**UNIVERSITI TEKNOLOGI MARA**

# COMPARATIVE ANALYSIS OF SPEECH DETECTION MODELS WITH A FOCUS ON THE JAPANESE LANGUAGE

**MUHAMMAD ALIFF AIMAN BIN ZOLKIFELI**

**MSc**

**March 2025**

**UNIVERSITI TEKNOLOGI MARA**

# COMPARATIVE ANALYSIS OF SPEECH DETECTION MODELS WITH A FOCUS ON THE JAPANESE LANGUAGE

**MUHAMMAD ALIFF AIMAN BIN ZOLKIFELI**

Dissertation submitted in partial fullfillment
of the requirements for the degree of
**Master of Computer Science**

**College of Computing, Informatics and Mathematics**

**March 2025**

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER ONE
# INTRODUCTION

## 1.1 Research Background

Human-computer interaction has been evolved from manual input into more natural interace and one of the natural interface is Automatic Speech Recognition (ASR). ASR have the capibilities to convert spoken language into text which is really useful in in application such as captioning, meeting minutes, media archiving, and voice-enabled search(**Xu**). In the context of japanese language, the quality of the transcription is still a challenge because of the multiple writing system that is kanji, hiragana and katakana and also context-sensitive readings that exist in the language (**Koenecke2020**).

Most of the ASR pipelines is focusing on the acoustic or end-to-end model (**xu2023recent**). However the real-world performance really depends on the preprocessing and feature extraction decision made before the data is fed into the model. The important preproccessing step is noise and reverberation reduction, voice activity detection (VAD), segmentation, resampling, channel and gain normalization, and cepstral mean/variance normalization (CMVN). Meanwhile the important feature extraction that need to be carried out is MFCC for traditional systems, and log-Mel filterbanks with augmentation for modern neural models. The preprocessing and feature extraction step can gave a big impact to the accuracy and stability of the model (**latif2020**).

In addition to the preprocessing steps, the model's output also must be clean for easy reading, quoting, and storing. This creates additional considerations, such as post-processing and text formalization. For example, inconsistent punctuation and stopwords must be handled during post-processing. This step is important to convert raw ASR output into a standardized format that can be utilized for lecture notes, reports, or subtitles (**ando2021**).

This paper will be focusing on formal Japanese speech transcription and compare three representative modeling approaches under controlled pre-processing and

feature extraction. The first model is GMM-HMM pipeline that is traditional model, the second model is CRDNN-CTC model that is hybrid model based on neural network, and the last model is fine-tuned Whisper model that is a transformer model. This paper will also evaluate the model performance based on Character Error Rate (CER) as the primary metric, and Word Error Rate (WER) and Real-Time Factor (RTF) as secondary metrics.

### 1.2 Problem Statement

Despite rapid progress in ASR, the comparative performance of different model families for formal Japanese transcription remains unclear. Many studies have explored various architectures, including traditional GMM–HMM systems (**huang2001**), hybrid neural models like CRDNN–CTC (**graves2013**), and large-scale transformer models such as Whisper (**radford2022robust**). However, most of the studies focus on English or multilingual datasets, with relatively fewer investigations dedicated to Japanese (**ando2021**; **xu2023recent**). From that number, only a few is comparing models from different model family. To get the clear comparision between these model afamilies also challenging as different studies is using differnt dataset and preproccessing step. As the result, it is difficult to draw definitive conclusions about which model type performs best for formal Japanese transcription tasks.

This study addresses that gap by conducting a controlled, apples-to-apples comparison of three representative traditional statistical GMM–HMM, hybrid CRDNN–CTC, and fine-tuned Whisper transformer models by standardizing preprocessing, feature extraction, and evaluation protocols. All models will be trained and evaluated on the same formal Japanese speech dataset with matched preprocessing configurations such as segmentation, resampling, CMVN and the feature types such as MFCC and log-Mel. Then the outputs will be post-processed uniformly to ensure readability and standardization. Finally, the same evaluation metrics CER, WER and RTF will be computed consistently to quantify both accuracy and usability. This systematic comparison will clarify which model family and preprocessing/feature setup is most effective for formal Japanese transcription under matched conditions, filling a critical gap in the literature and informing best practices for ASR system design in Japanese contexts.

## 1.3 Research Objectives

1. To identify the key requirements for constructing speech-to-text model within the context of Japanese language.

2. To analyze speech-to-text models to determine the most effective pre-processing and training setup to reduce CER and RTF in Japanese language processing.

3. To evaluate the CER and the transcription latency using RTF of different speech-to-text model when transcribing Japanese formal and informal language.

## 1.4 Research Questions

1. What is the key requirements for constructing speech-to-text model within the context of Japanese language.

2. What is the most effective pre-processing and training setup to reduce CER and RTF in Japanese language processing.

3. How to calculate the performance and effectiveness using CER and RTF of different speech-to-text model in context of Japanese language?

## 1.5 Scope of Study

This study will be focusing on speech-to-text transcription of Japanese language using only formal speech. The dialect speech will not be included in this study. The main focus of this study is to produce a readable and standardized text rather than detect dialects or classify speaking styles. For the preproccessing and feature extraction, this study will be focusing on segmentation, resampling and normalization, and two feature families of MFCC and log-Mel filterbanks.

The models that will be compared in this study are GMM-HMM, CRDNN-CTC, and fine-tuned Whisper. The evaluation metrics that will be used in this study are Character Error Rate (CER) as the primary metric, and Word Error Rate (WER) and Real-Time Factor (RTF) as secondary metrics. The RTF will be calculated by

dividing the total decoding wall-clock time by the total audio duration, using a fixed hardware/software setup. For the text post-processing, this study will be focusing on normalizing the output for formal use by removing fillers words, numeric and punctuation normalization, and consistent script conventions. Semantic editing and translation are out of scope for this study.

For the dataset, this study will only be using formal Japanese speech with available transcripts. The data that will be used is TEDx talks in Japanese language from YouTube that is scraped using *yt-dlp* tool and then the audio is extracted using *ffmpeg* tool.

## 1.6 Significance of Study

This study is aimed to address the gap of effective speech-to-text solution that focusing on Japanese language. Most of the developed models is focusing on English language or a generic transcribe model that is developed for multi-language. Organization that rely on accurate transcript like broadcasters, government agencies, and archives rely on this kind of technology. This thesis will be a guidance to determine which pre-processing and feature configurations most improve outcomes for formal Japanese across different model types.

This study also contributes to the research by providing a systematic comparison of model choices in ASR across traditional, hybrid, and fine-tuned transformer models in the context of formal Japanese transcription. By filling this gap, the findings will tell best practices for ASR system design in Japanese, supporting both academic research and practical applications in industries that depend on accurate speech-to-text conversion.

## 1.7 Conclusion

In this chapter, the advancement in machine learning and artificial intelligence that made the computer can understand human better by improving the speech to text model accuracy and speed has been discussed. However, there is still challenges to transcribe a language that has complex structure like Japanese that include syllable-based formation and the use of multiple writing systems. Because of this, a study to

find which implementation and which model is the most performance for handling Japanese language. The finding from this study is very important to answer the question of which model is the best for speech-to-text solution in Japanese language. By identifying the specific linguistic challenges and comparing these models, this study will provide a valuable information that will be able to guide future advancements in speech-to-text technology in Japanese language and ultimately will be able to support its broader application across the industries that rely heavily on precise and efficient transcription.

# CHAPTER TWO
# LITERATURE REVIEW

## 2.1 Introduction

The technology for Automatic Speech Recognition (ASR) has advanced rapidly in these years. Early traditional models like GMM and HMM into more sophisticated deep learning approaches such as DNN, CNN, RNN, and Transformer-based architectures.

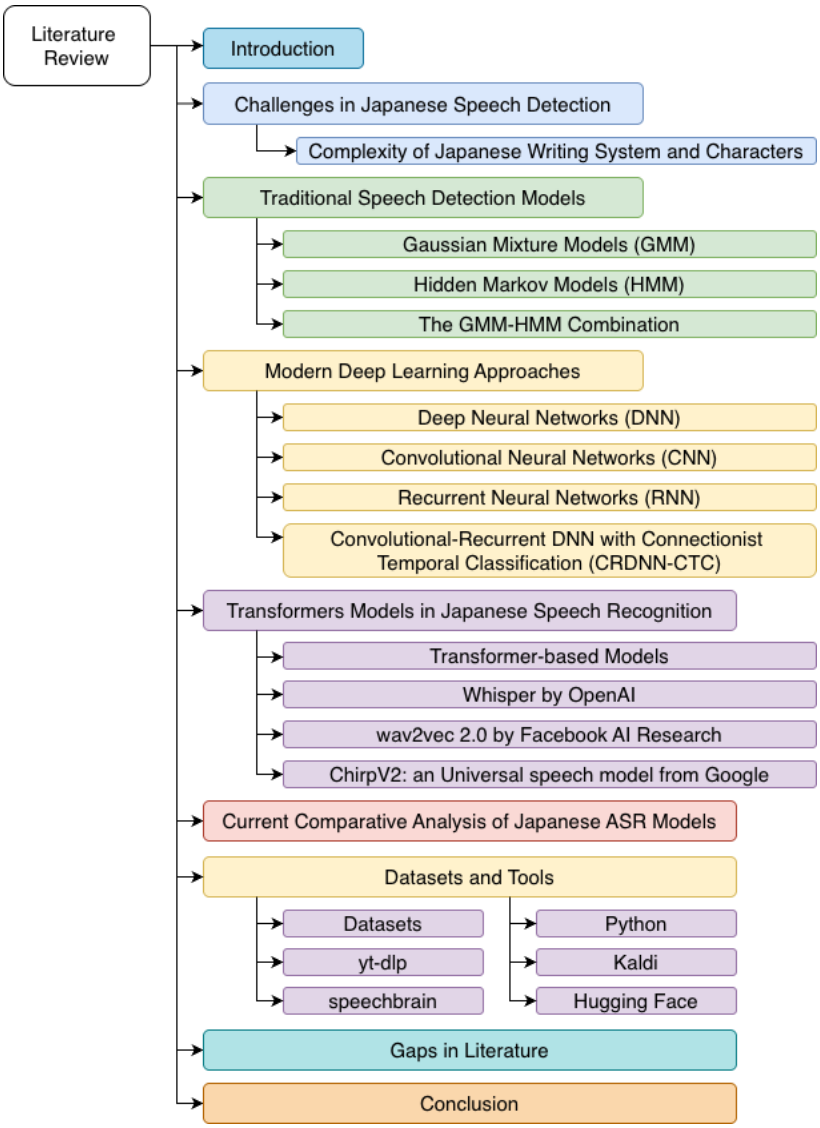


Figure 2.1 Literature Review Mind Map

However, to apply these technologies to the Japanese language may pose few

challenges due to its complex writing systems. In this chapter, the challenges of Japanese speech detection and the traditional and modern ASR models will be reviewed. The state-of-the-art model like Whisper, wav2vec, and Chirp will also be discussed based on their applicability to Japanese. By identifying the key challenges and gaps in existing research, this chapter prepared for a focused analysis of Japanese-specific ASR systems.

## 2.2 Challenges in Japanese Speech Detection

### 2.2.1 Complexity of Japanese Writing System and Characters

The complexity of Japanese writing system and character can cause challenge in ASR system especially in end-to-end neural network architectures. Japanese writing system is a combination of multiple character sets, such as the Hiragana, Katakana, Kanji (ideographic characters), Roman letters and various symbols, leading to a considerably larger and more varied character (**rose2019unique**). As mentioned by **Ito2016End-to-end**; **ito2017<empty citation>**, the number of possible Japanese character labels can exceed several thousand.

A single character of kanji may have a few ways to pronounce it because each character of kanji has Onyomi (Chinese derived) and Kunyomi (native Japanese) readings, and these readings can change depending on the word context (**curtin2020japanese**). Because of this ambiguity, the ASR system must be able to model and distinguish numerous acoustic differences in the speech data with same sound. The training model must be able to handle thousands of character and each of the character is potentially linked to multiple context dependent phonetic outcome which require a significant computational resources and large scale training data to ensure adequate coverage (**Ito2016End-to-end**; **ito2017**).

## 2.3 Traditional Speech Detection Models

### 2.3.1 Gaussian Mixture Models (GMM)

GMM have been the earliest technology used for developing Japanese speech detection and recognition systems because of their capability in capturing

the statistical distribution of speech features very well (**Imaishi2022Examination**). Because of the absence of word boundaries and the nuances of pitch accent in the Japanese language, it is really complicated to understand the context of the spoken words. However, GMM would be useful by using probabilities to manage and characterize intricate patterns (**sun2020subspace**). For example, **povey2011subspace<empty citation>** were able to use GMM to model phoneme-based acoustic features, and this approach led to a good performance of speech recognition systems.

**Imaishi2022Examination<empty citation>** developed an approach within the EM algorithm that leads to the stabilization of the GMM parameters as well as increasing the discriminative power of the model in cases where there is not much evaluation data available. In other work, **povey2011subspace<empty citation>** point out that it is possible to represent the distribution of speech features in GMM mode by employing a combination of several Gaussian components. **Takami2020Performance<empty citation>** emphasized a different direction which starts with the creation of the Universal Background Model, which is a Gaussian Mixture Model calculated from the collection of a large number of speech samples.

### 2.3.2 Hidden Markov Models (HMM)

HMM is working quite well with Japanese speech detection because of the incorporation of the acoustic and temporal characteristics of speech, including the difficulties found in the encoding of Japanese speech (**Tokuda1999Application**). ASR systems incorporated with HMM are more superior in portraying Japanese speech characteristics' rhythm and tone including essential features like pitch accent and moraic timing which features will enhance the performance of the systems on the phonology aspects of the language (**Tokuda2000HMM**).

To further Increase Japanese ASR capability, few other models is used along with HMM which is context-sensitive such as Tri-phone method. Tri-phone method is a phonetic expansion that employs phonetics of the neighbor sounds to the phoneme as context in order to increase the recognition accuracy by taking into account the co-articulation that takes place during fast speech production (**Tokuda2000HMM**).

### 2.3.3 The GMM-HMM Combination

In the Kaldi framework, Japanese large-vocabulary ASR has commonly been implemented as a GMM–HMM pipeline where HMM states capture temporal dynamics and GMMs model frame-level acoustic likelihoods, typically paired with context-dependent decision-tree clustered triphones. Classic Kaldi recipes also report stepwise gains when moving from simpler triphone systems to stronger feature-space modeling. For example, the Kaldi CSJ recipe reports that performance improves from early triphone stages to later stages, with eval1 WER decreasing from 22.67% (tri1) to 21.49% (tri2) and further to 17.49% (tri3) and 15.26% (tri4), demonstrating the typical benefit of successive refinements in the conventional HMM-based pipeline. (**KaldiCSJResults2015**)

More recently, **ando2021<empty citation>** evaluated Kaldi-style HMM-based systems using the official CSJ recipe as a baseline training procedure and reported CER on both an out-of-domain TEDx-style test set and the standard CSJ evaluation sets. On the TEDxJP-10K evaluation set, their best reported configuration achieved 17.92% CER, while simpler configurations such as CSJ acoustic model with CSJ language model achieved 23.41% CER. These results provide concrete reference points for conventional HMM-based Japanese ASR performance under both matched (CSJ) and more realistic, diverse conditions (TEDxJP-10K). (**ando2021**)

Table 2.1
ASR decoding results (CER%) on the TEDxJP-10K dataset (adapted from **ando2021<empty citation>**).

| LM | CSJ | TV | CSJ+TV |
|---|---|---|---|
| CSJ | 23.41 | 21.83 | 20.61 |
| TV | 22.14 | 18.98 | 18.22 |
| OSCAR | 23.52 | 19.28 | 18.71 |
| TV+OSCAR | 21.89 | 18.29 | 17.92 |

## 2.4 Modern Deep Learning Approaches

### 2.4.1 Deep Neural Networks (DNN)

The use of DNN in conjunction with HMM, also known as DNN-HMM has been shown to improve performance in Japanese ASR.

**seki2014comparison<empty citation>** compared syllable-based and phoneme-based DNN-HMM and found that the syllable-based DNN-HMM was better, as its parameter space is less coupled with the context of the syllables. They reported that an 11% relative decrease in the WER for triphone DNN-HMMs over syllable-based DNN-HMMs when used on large databases such as ASJ+JNAS. The multilayered structure of DNNs makes it much suitable for developing models of contextual dependencies for speech signals (**hojo2018dnn**).

**mu2020japanese<empty citation>** developed a double-deep neural network for the evaluation of Japanese pronunciation to address the problems of text-to-speech alignment and scoring. The DDNN integrated CNN and RNNs with attention and it is effective for detecting pronunciation mistakes. **lin2017dnn<empty citation>** noted the importance of addressing the particular problem of the lack of annotated Japanese speech corpora by emphasizing the use of transfer learning with DNN.

### 2.4.2 Convolutional Neural Networks (CNN)

(**noda2014lipreading**) conducted a research using elastic net regression on a seven-layer CNN structure and 58% of phoneme recognition accuracy was obtained for Japanese datasets. Building upon this work, **yalta2019cnn<empty citation>** constructed a functional speech recognition framework inclusive of several types of words spoken intended for tight spots like houses.

**noda2014lipreading<empty citation>** investigated the use of CNNs for solving the problem of creating a Japanese speech acoustic model. CNN used to encode the frequency-time domain audio and properly exploit the spatial and temporal aspects. The combination of CNNs with attention mechanisms has been beneficial in increasing accuracy and interoperability during the detection of long utterances and multi-speaker datasets (**Mukohara2015Emotion**).

### 2.4.3 Recurrent Neural Networks (RNN)

In the work of **takeuchi2020real<empty citation>**, RNN is introduced, which enables the processing of input speech while removing noise and help to reduce exploding gradient problems often seen in RNNs. **Kida2016LSTM<empty citation>** investigated linear prediction filters based on LSTM which did not require direct ac-

cess to raw information and thus can extract features from distorted signals, as an LSTM estimated linear prediction coefficients.

**Kubo2014DeepLearning<empty citation>** broadened approaches incorporating RNNs into synthesizing speech for Japanese, focusing on improving prosody and intonation. **takeuchi2020real<empty citation>** took advantage of the RNN-based architectures for the acoustic modelling for Japanese ASR showed that even though GRUs have a simpler gating strategy than LSTMs, they could achieve a similar level of classification accuracy with lower compute requirements. Then, the studies on bidirectional LSTMs (BLSTMs) by **imaizumi2022<empty citation>** revealed that the past context and the future context of the signal can be utilized for better performance of ASR.

### 2.4.4 Convolutional-Recurrent DNN with Connectionist Temporal Classification (CRDNN-CTC)

The CRDNN-CTC family is part of the broader end-to-end ASR direction that replaces the conventional lexicon-driven alignment pipeline with direct sequence learning. In this approach, acoustic features are mapped to output symbols using a neural encoder, and the Connectionist Temporal Classification (CTC) objective provides a monotonic alignment between the input frames and the target transcription without requiring frame-level labels. This has been used in Japanese ASR because Japanese can be transcribed naturally at the character or kana level, while word boundaries are not explicitly marked in continuous speech, making forced alignment and word-level tokenization less straightforward (**watanabe2017hybrid**; **watanabe2018espnetendtoendspeechprocessing**).

A consistent finding in prior work is that CTC-based models can achieve strong Japanese recognition accuracy while maintaining a simple training pipeline. For example, **chen2020streaming<empty citation>** proposed an end-to-end streaming Japanese speech recognition method that combines CTC with local attention and reported a best CER of 9.87%, demonstrating that competitive character-level performance is possible even under streaming constraints where future context is limited. This line of work also supports the practicality of CTC-style training for Japanese, where stable alignments can be learned directly from paired audio-transcript data

without the needs of pronunciation dictionaries (**chen2020streaming**).

Table 2.2
Streaming Japanese ASR results (Avg. CER% and latency) using CTC and local attention (from **chen2020streaming<empty citation>**).

| Model no. | CNN | Attn. | Latency (ms) | Avg. CER (%) |
|---|---|---|---|---|
| 1 | | | 40 | 12.60 |
| 2 | | | 60 | 13.67 |
| 3 | | ✓ | 240 | 10.37 |
| 4 | | ✓ | 360 | 10.40 |
| 5 | ✓ | | 40 | 10.03 |
| 6 | ✓ | | 60 | 12.27 |
| 7 | ✓ | ✓ | 240 | 9.87 |
| 8 | ✓ | ✓ | 360 | 10.27 |
| 9 | ✓ | ✓ | 360 | 10.23 |

## 2.5 Transformers Models in Japanese Speech Recognition

### 2.5.1 Transformer-based Models

**taniguchi2022transformer<empty citation>** propose a series of Transformer-based ASR models aimed at improving Japanese speech recognition, particularly in the context of simultaneous interpretation. They investigate the possibility of utilizing auxiliary input like the source language text to resolve issues such as disfluencies, hesitations, and self-repairs commonly observed in the interpreter speech which helps to improve the transcription quality (**Futami2020Bidirectional**). The models combined audio and text data via multi-modal transformer encoders and decoders, which offers a broader scope of recognition by using previously provided source language text for interpreter training programs (**taniguchi2022transformer**).

A wide range of datasets for source text and simultaneous interpretation speech are however not readily available, so the authors use a adapted speech translation corpora from MuST-C and CoVoST 2 while also introducing TED based Japanese texts for evaluation purposes (**taniguchi2022transformer**). With an additional goal of enhancing performance, the authors fine-tune the source language text encoder by using large machine translation corpora which helps in lowering the word error rates during translation of English, Dutch, German and Japanese (**Taniguchi2024Pretraining**). Results consistently demonstrate that incorporat-

ing source language text into Transformer-based ASR models significantly improves recognition performance, with the greatest impact observed when auxiliary input is introduced at later stages of the audio encoding and decoding process (**Futami2020Bidirectional**).

### 2.5.2 Whisper by OpenAI

Large scale weak supervision has emerged as one of the major approaches in speech recognition as noted by **radford2023robust<empty citation>** in their development of whisper model that has been trained on multilingual and multitask audio datasets that has a combined duration of 680,000 hours. This work is a continuation to the self-supervised methods such as Wav2Vec 2.0 (**baevski2020wav2vec**), which demonstrated learning without supervision from audio without any human-provided labels. However, dataset-specific fine-tuning is often necessary to obtain good performance, whereas with Whisper such reliance is reduced because of the efficacy of weak supervision.

By scaling weak supervision across diverse datasets, Whisper able to bypass the need for dataset-specific adaptation while able offer a robust zero-shot performance across languages and tasks. This method also resulting in the models to have similar trends with other state of the art model in machine learning where a large, diverse datasets will improve model resilience which is align the with the advancements in computer vision (**kolesnikov2020big**) and NLP (**radford2019language**).The Whisper model's architecture, a simple encoder-decoder Transformer, reinforces the effectiveness of minimal preprocessing and sequence-to-sequence training, simplifying the transcription pipeline.

**bajo2024efficient<empty citation>** detail their work on adapting OpenAI's Whisper model to enhance its performance in ASR for the Japanese language. The research draws attention to the dilemma faced in balancing the multilingual being and the accuracy of an English-only product, ReazonSpeech, that seeks to maximize on the Japanese language ASR, which is monolingual in nature. By using a Japanese dataset while utilizing Low-Rank Adaptation (LoRA) and fine-tuning methods, they were able to lower Whisper-Tiny's CER from 32.7% to 14.7%. This fine tuning method showed that smaller multilingual models give more promising result,

after being tuned for the desired language outperform their larger baseline models, for example the case of the Whisper-Base model (**bajo2024efficient**).

Table 2.3
WER and CER performance of Whisper models. Reproduced from **bajo2024efficient**.

| Model | WER (%) | CER (%) |
|---|---|---|
| Whisper Tiny | 47.48 | 32.74 |
| Whisper Base | 29.81 | 20.20 |
| Whisper Small | 16.14 | 9.89 |
| Whisper Medium | 10.84 | 6.86 |
| Whisper Large | 7.41 | 4.77 |
| Whisper Tiny + LoRA | 33.16 | 20.83 |
| Whisper Base + LoRA | 23.36 | 14.50 |
| Whisper Small + LoRA | 14.90 | 9.16 |

## 2.6 Current Comparative Analysis of Japanese ASR Models

A comparative analysis that carried out by **Karita2021<empty citation>** shows that Conformer-based models perform better than Conformer BLSTM architectures, as they obtained 4.1, 3.2, and 3.5 character error rates for CSJ in eval1, eval2, and eval3 tasks respectively. It is noted that both the BLSTM and Conformer models have character error rates below 7% and the character error rate is lower when using Conformer Itself. Conformer encoders also offer increased accuracy and efficiency, with a throughput of 628.4 utterances processed per second and 430.0 for the BLSTM models (**Karita2021**).

Table 2.4
Character error rates on CSJ dev/eval1/eval2/eval3 sets cited from **Karita2021**.

| Encoder | Decoder | Param | Utt/sec | CER [%] |
|---|---|---|---|---|
| BLSTM | CTC | 258M | 430.0 | 3.9 / 5.2 / 3.7 / 4.0 |
| BLSTM | attention | 309M | 365.5 | 3.8 / 5.3 / 3.7 / 3.7 |
| BLSTM | transducer | 274M | 297.6 | 3.8 / 5.1 / 3.7 / 4.0 |
| Conformer | CTC | 117M | **628.4** | 3.1 / 4.1 / 3.2 / 3.5 |
| Conformer | attention | 124M | 534.8 | 3.3 / 4.5 / 3.3 / 3.5 |
| Conformer | transducer | 120M | 376.1 | **3.1 / 4.1 / 3.2 / 3.5** |

Another comparative analysis in ASR for Japanese is presented by **takahashi2024comparison<empty citation>**, focusing on the accuracy of speech recognition for different dialects. The study evaluates three models: Whisper, XLSR, and XLS-R, which are self-supervised learning frameworks. The Whisper model

significantly underperformed for any Japanese outside of standard Japanese, recording a 4.1% CER only after it has gone through fine tuning. However, when the accuracy is low when the language identification marker is absent where some instances of being higher than 100%.This marks the weakness of Whisper in terms of its application for wide ranging applications in different dialects of Japanese (**takahashi2024comparison**).

Table 2.5
Comparison of ASR accuracy on two datasets, Standard Japanese (CSJ) and Japanese dialects (COJADS) cited from **takahashi2024comparison**.

| Pre-Trained Model Name | Adaptation Method | CER [%] CSJ | CER [%] COJADS |
|---|---|---|---|
| Whisper-medium (zeroshot) | - | 25.6* | 116.0* |
| Whisper-medium | full finetuning | **4.1** | 32.9 |
| XLSR | full finetuning | 6.5 | 34.1 |
| XLSR | 3-steps finetuning | - | 30.0 |
| XLS-R | full finetuning | 6.1 | 32.6 |
| XLS-R | 3-steps finetuning | - | **29.2** |

*After post-processing with kanji-to-kana conversion.

## 2.7 Current Comparative Analysis of Japanese ASR Models

A recent comparative study by **hono2024integratingpretrainedspeechlanguage<empty cita** proposed an end-to-end Japanese ASR framework that integrates a pre-trained speech encoder (HuBERT) with a decoder-only large language model (GPT-NeoX) via a bridge network, aiming to avoid explicit LM-fusion and keep decoding simple. The evaluation compared the proposed model against publicly available baselines across several Japanese corpora. For efficiency, the authors reported that applying DeepSpeed-Inference improved inference speed by about 1.4× while keeping recognition accuracy unchanged. In terms of CER, the proposed model outperformed reazonspeech-espnet-v1 across all beam settings on the ReazonSpeech test set (in-domain), while showing mixed outcomes on JSUT and CV8.0 compared to larger-beam decoding.

Table 2.6
ASR results reported by
**hono2024integratingpretrainedspeechlanguage<empty citation>** on
ReazonSpeech, JSUT, CV8.0, and CSJ eval sets (beam size 1 = greedy decoding).

| Model | Params | Beam | Reazon | JSUT | CV8.0 | CSJ E3 |
|---|---|---|---|---|---|---|
| reazonspeech-espnet-v1 | 90M | 1 | 12.0 | 8.7 | 10.5 | 15.3 |
| reazonspeech-espnet-v1 | 90M | 20 | 8.7 | 7.5 | 8.9 | 15.5 |
| Whisper-large-v3 | 1,541M | 1 | 12.4 | 7.1 | 8.2 | 14.8 |
| Proposed (w/o DS-Inf.) | 3,708M | 1 | 8.4 | 8.6 | 9.1 | 22.9 |
| Proposed (w/ DS-Inf.) | 3,708M | 1 | 8.4 | 8.6 | 9.2 | 22.9 |

Current comparative analysis from these studies shows that several challenges need to be addressed. A study to compare the existing models in Japanese ASR is needed because of the unique characteristics of the Japanese language. The comparative analysis from **Karita2021<empty citation>** and **takahashi2024comparison<empty citation>** shows that while models like Conformer and Whisper have made significant strides in ASR, they still face challenges in handling the complexities of Japanese. Additionally, there is still lack of study that comparing the model from different model families because most of the existing comparative analysis only focusing on models from the same family.

## 2.8 Datasets and Tools

### 2.8.1 Datasets

Dataset is a crucial component in training and evaluating ASR models. In this study, a curated dataset collected from Japanese TEDx talks on YouTube with manual Japanese subtitles is used. TEDx talks is a rich source of diverse speech data, covering a wide range of topics and speakers, making it suitable for training ASR models. The talks are delivered by various speakers with different accents, speaking styles, and speech rates, which helps in creating a comprehensive dataset that captures the variability in real-world Japanese speech (**ando2021**).

### 2.8.2 Python

Python was used as the main programming language to implement the data pipeline, training utilities, and evaluation scripts. It supported preprocessing

tasks such as transcript cleaning, manifest generation, and training split preparation (**python3.14documentation_2025**). In addition, Python libraries were used to support ASR-related workflows such as audio processing, metadata handling, diarization-based filtering, and evaluation.

### 2.8.3 yt-dlp

Yt-dlp is a command-line tool and Python library that allows downloading videos, audio, and subtitle tracks from YouTube. In this study, yt-dlp was used to query YouTube metadata to verify the availability of manual Japanese subtitles before downloading. Then, audio and subtitle files were downloaded for selected Japanese TEDx talks to form a consistent dataset for ASR training and evaluation (**yt-dlp**).

### 2.8.4 Audio Processing Tools

To standardize the dataset and ensure compatibility across model families, audio was converted into a consistent format (16 kHz, mono, 16-bit PCM). Audio resampling and waveform writing were performed using Python audio libraries `librosa` and `soundfile` (**mcfee_2025_15006942**; **soundfile**). In addition, WebRTC voice activity detection was used to segment long TEDx recordings into shorter utterances suitable for training and evaluation (**webrtc**).

### 2.8.5 Automatic Audio Filtering Tools

A filtering step was applied to remove non-speaker or low-quality segments such as applause and laughter. Speaker structure filtering used `pyannote.audio` diarization to prefer clips with a dominant speaker. Acoustic event filtering used a Hugging Face zero-shot audio classification pipeline with the CLAP model (`laion/clap-htsat-fused`) to identify and remove segments with high confidence of applause or laughter (**clap**).

### 2.8.6 Kaldi

Kaldi is an open-source toolkit for speech recognition that provides a complete pipeline for feature extraction, acoustic modeling, and decoding (**kaldi2011**).

In this study, Kaldi was used to implement the traditional GMM–HMM baseline system. The workflow included MFCC feature extraction with CMVN normalization, staged acoustic model training (mono, tri1, tri2, tri3), alignment between stages, and decoding graph generation for evaluation.

### 2.8.7 KenLM

KenLM was used to build an external *n*-gram language model to support decoding in the Kaldi GMM–HMM systems. In this work, `lmplz` was used to produce an ARPA-format language model, which was then converted into decoding resources within the Kaldi graph construction pipeline (**kenlm**).

### 2.8.8 Lexicon Generation Tools

A grapheme-based lexicon was generated using a custom Python script. The script extracted vocabulary items from transcripts and converted Japanese tokens into Hepburn romanisation using `pykakasi`. The romanised output was normalized and mapped into character-level pronunciation units used to build Kaldi language resources (**pykakasi**).

### 2.8.9 SpeechBrain

SpeechBrain is an open-source toolkit for speech processing that supports end-to-end ASR training and evaluation (**ravanelli2021speechbraingeneralpurposespeechtoolkit**). In this study, SpeechBrain was used to implement the CRDNN–CTC model. It used the prepared CSV manifests and a character-level vocabulary, and training checkpoints were managed using SpeechBrain's checkpointer mechanism (**speechbrain_v1**).

### 2.8.10 Hugging Face

Hugging Face was used to access pretrained Whisper models and supporting utilities for fine-tuning. The Transformers library was used to load Whisper checkpoints, configure Japanese decoding prompts, and generate hypotheses using `model.generate()`. Hugging Face Datasets was used to load

train/validation/test CSV splits efficiently and support reproducible fine-tuning experiments (**wolf2020huggingfacestransformersstateoftheartnatural**).

### 2.8.11 Evaluation Tools

Model evaluation used character error rate (CER), word error rate (WER), and real-time factor (RTF). CER and WER were computed using normalized edit distance between hypothesis and reference transcripts. In this study, the `jiwer` Python library was used to compute WER and to support consistent error-rate scoring across systems. RTF was computed by dividing total decoding wall time by total audio duration to measure decoding efficiency and latency behaviour (**jiwer_2025**).

## 2.9 Gaps in Literature

Based on the literature review, shows that there is several gaps in the research on Japanese ASR. First, while there has been significant progress in ASR models, there is still a lack of comprehensive comparative studies that evaluate different model architectures specifically for Japanese. Most existing studies focus on individual models or specific families of models, making it difficult to draw broad conclusions about the best approaches for Japanese ASR. Second, many studies are using datasets that already being throughly studied, such as CSJ, which has been cleaned and preprocessed extensively. There is a need for more research using diverse and real-world datasets, such as TEDx talks, to better understand model performance in practical scenarios.

## 2.10 Conclusion

This chapter has reviewed the existing literature on Japanese ASR, covering traditional models like GMM–HMM and modern deep learning approaches including CRDNN-CTC, and Transformer-based models like Whisper. The unique challenges posed by the Japanese language, such as its complex writing system and character pronunciations, were discussed. Tools and datasets commonly used in Japanese ASR research were also highlighted and gaps in the literature were also being identified.

# CHAPTER THREE
# RESEARCH METHODOLOGY

## 3.1 Introduction

This chapter explained the methodology used to evaluate Japanese automatic speech recognition (ASR) across three model families: a traditional Kaldi-style GMM–HMM system, an end-to-end CRDNN–CTC model, and a fine-tuned transformer encoder–decoder model based on Whisper model from OpenAI (**kaldi2011**; **speechbrain_v1**; **radford2023robust**). This chapter described the data pipeline such as data collection, audio and transcript cleaning, and audio splits. The preprocessing, training, decoding process and the scoring protocol used for each model family was also discussed in this chapter.

## 3.2 Research Design

This study was organised into six phases where started from planning and followed by reviewing prior work. And then moved to data collection and preprocessing phase where dataset was compiled and processed. After that, the selcted model were trained and fine tuned using the data. Then the model was scored using selected metrics and the the data finally were analyzed and discussed in the later phase. Table 3.1 below showed an overview of the phases, activities and deliverables.

Table 3.1
Overview of Research Project

| Phase | Activities | Deliverables |
|---|---|---|
| **Planning** | Defined below items:<br>• Project title<br>• Problem statements<br>• Objectives<br>• Scopes<br>• Significance | Chapter 1<br>• Title defined<br>• Problems defined<br>• Objectives defined<br>• Scope defined<br>• Significance defined |
| **Prior Work** | • Reviewed Japanese ASR studies<br>• Listed preprocessing methods<br>• Listed model families<br>• Listed evaluation metrics<br>• Summarized key gaps | Chapter 2<br>• Literature review written<br>• Preprocessing chosen<br>• Models selected<br>• Metrics defined<br>• Gaps summarized |
| **Data Collection and preprocessing** | • Collected Japanese TEDx talks<br>• Downloaded audio and subtitles<br>• Cleaned and normalized transcripts<br>• Resampled audio to 16 kHz mono<br>• Split audio using VAD<br>• Created train/valid/test splits<br>• Exported manifests and metadata | Chapter 3<br>• Dataset compiled<br>• Collection method documented<br>• Transcripts normalized<br>• Segments generated<br>• Splits finalized<br>• Manifests prepared |
| **Model Training and Fine Tune** | • Trained GMM-HMM<br>• Trained CRDNN-CTC<br>• Fine-tuned Whisper variants<br>• Set decoding parameters<br>• Saved configs and checkpoints | Chapter 4<br>• Models trained<br>• Checkpoints saved<br>• Settings recorded<br>• Outputs generated |
| **Evaluation** | • Scored with CER and WER<br>• Measured speed using RTF<br>• Compared models and decoders | Chapter 4<br>• Results tables and plots<br>• Best model identified |
| **Discussion** | • Explained main findings<br>• Linked to prior work<br>• Noted limitations<br>• Suggested future work | Chapter 5<br>• Findings discussed<br>• Limitations stated<br>• Recommendations given<br>• Future work proposed |

## 3.3 Planning Phase

Table 3.2 showed the first phase of this project where most of the planning and decision about this projectis done. The activities carried out and the outcome

deliverables were described in the table shown below.

Table 3.2
Planning Phase

| Phase | Activities | Deliverables |
|---|---|---|
| **Planning** | Defined below items:<br>• Project title<br>• Problem statements<br>• Objectives<br>• Scopes<br>• Significance | Chapter 1<br>• Title defined<br>• Problems defined<br>• Objectives defined<br>• Scope defined<br>• Significance defined |

The starting point of this project began in the planning phase. This phase was important because it served as the blueprint and became guidance to the project direction (**TODO_planning_methodology**). This phase purpose was to identify the problem in the domain and found a way to solve the problem stated (**TODO_planning_methodology**). The activities that was carried out in this pahse was defining key aspect that was the title of the project, problem statements, objective, scope and significance of the project. The outcome from this phase was the title of the project, problems that has been defined, objective of the project, scope of the project which explained what was covered and what was not covered in this project and last deliverables in this phase was the significance that has been identified. During this planning phase, things like time, cost, resource, benefit and difficulty level also have been take into consideration (**TODO_planning_methodology**).

## 3.4 Prior Work Phase

Table 3.3 showed the second phase of this project where prior work in the domain of ASR was reviewed and some decision was made. The activities carried out and the outcome deliverables were described in the table shown below.

Table 3.3
Prior Work Phase

| Phase | Activities | Deliverables |
|---|---|---|
| **Prior Work** | • Reviewed Japanese ASR studies<br>• Listed preprocessing methods<br>• Listed model families<br>• Listed evaluation metrics<br>• Summarized key gaps | Chapter 2<br>• Literature review written<br>• Preprocessing chosen<br>• Models selected<br>• Metrics defined<br>• Gaps summarized |

After planning phase was completed and the reasearch domain has been set, next phase was reviewing prior work phase. During this phase, multiple sources such as related article, journal, conference paper and textbook was reviewed (**TODO_literature_review_method**). This resource was used to evaluate and identified what has been discussed or discovered (**TODO_literature_review_method**). In this phase, the activity involved was list down the preprocessing mrthod that currently being used in speech recognition and a few preprocessing technique was choosen based on the requirements of the models as the deliverables. The next activity was to choose which model to compared to, instead of comparing model from same family, this research has conclude to compare 3 model from different model architecture and these 3 models was the deliverables for this phase (**kaldi2011**; **speechbrain_v1**; **radford2023robust**). The next activity was to conclude which metrics was used to compare these models. As the deliverables, three metrics has been choose to do comparitive analysis between each models (**jiwer_2025**; **TODO_rtf_reference**). In this phase, the gaps in the literature was also defined and as the deliverables the gaps in the literature was summarized (**TODO_literature_review_method**).

## 3.5 Data Collection and preprocessing Phase

Table 3.4 showed the third phase of this project where the source of the data was selected, aqquired and then the data went through preprocessing to make sure data was ready to be input into models. The activities carried out and the outcome deliverables were described in the table shown below.

Table 3.4
Data Collection and preprocessing Phase

| Phase | Activities | Deliverables |
|---|---|---|
| **Data Collection and preprocessing** | • Collected Japanese TEDx talks<br>• Downloaded audio and subtitles<br>• Cleaned and normalized transcripts<br>• Resampled audio to 16 kHz mono<br>• Split audio using VAD<br>• Created train/valid/test splits<br>• Exported manifests and metadata | Chapter 3<br>• Dataset compiled<br>• Collection method documented<br>• Transcripts normalized<br>• Segments generated<br>• Splits finalized<br>• Manifests prepared |

The focus of this phase was to build a cleaned and useable dataset from publicly available dataset into a consistent training and evaluation format so that all three models from different family can use the sama dataset for traiing and testing (**ando2021**; **TODO_dataset_pipeline_reference**). The pipeline for this phase could be seperated into four section. The first task was collecting TEDx Japanese talks from YouTube with only Japanese subtitles that came with manual subtitles by human (**yt-dlp**). Then the audio was downloaded with the subtitle files (**yt-dlp**). The second task was to clean and normalize subtitle text into reference transcripts and standardized the audio into 16 kHz mono PCM (**mcfee_2025_15006942**; **soundfile**). Third task was to split the audio into chunks using voice activity detection (VAD) and the audio that did not have audio activity such as intro music was discarded (**webrtc**). The last task was to produce train/validation/test splits and manifests with audio paths, durations, and transcripts (**TODO_manifest_best_practice**).

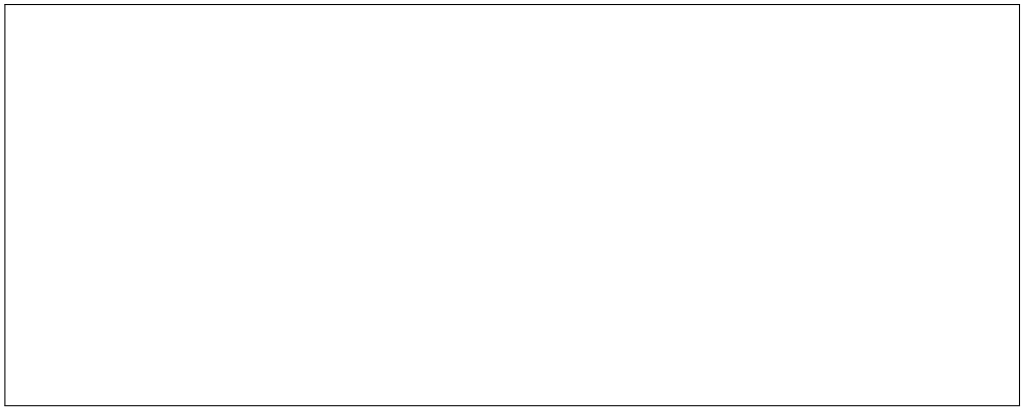

Figure 3.1 Placeholder: End-to-end data collection and preprocessing pipeline for TEDx Japanese talks (YouTube → cleaned transcripts → VAD chunks → manifests).

### 3.5.1 Data Source and Selection Criteria

The source of the data was collected from Japanese TEDx talks that was hosted on video sharing platform, Youtube. This particular source was choosen beacuse TEDx talks was a collection of public speaking where commonly the speaker had a clear speech with structured monologue-style delivery (**ando2021**). However, not all video that published came with manual Japanese subtitle by human, only part of the video had manual Japanese subtitle. The rest either did not have subtitle or the subtitle was in other language. To make sure that the transcripts were sufficiently reliable to serve as reference text during ASR training and evaluation, only videos that had Japanese manual subtitle was choosen and the rest was excluded (**TODO_reference_transcript_selection**). The videos that contained only auto-generated subtitles were also excluded to reduce transcription noise and alignment inconsistencies (**TODO_reference_transcript_selection**).

All audio and subtitle was downloaded using a python library name yt-dl (**yt-dlp**). The script first queried the video metadata to find if the video in playlist had Japanese subtitle or did not without downloading the video using `extract_info` function from yt-dl library (**yt-dlp**).

Listing 3.1: Audio splitting script

```
with yt_dlp.YoutubeDL(check_opts) as ydl:
    info = ydl.extract_info(video_url, download=False)

    # Check manual subtitles
    if 'subtitles' in info and 'ja' in info['subtitles']:
        has_jp_subs = True

    # Check auto-generated subtitles (ASR)
    elif ('automatic_captions' in info
          and 'ja' in info['automatic_captions']):
        pass
```

For the video that had Japenese subtitle, audio and transcript was downloaded (**yt-dlp**).

Listing 3.2: Audio splitting script

```
download_opts = {
    'subtitleslangs': ['ja'],
    'subtitlesformat': 'vtt',
}
if has_jp_subs:
    with yt_dlp.YoutubeDL(download_opts) as ydl:
        ydl.download([video_url])
```

### 3.5.2 Transcript Cleaning and Normalization

After the transcript was downloaded, the transcript needed to be cleaned since it could not be directly used as reference transcript (**ando2021**; **TODO_text_normalization_reference**). The raw downloaded trasncriptt contained non-speech markers, formatting tags, and artifacts introduced for readability rather than ASR training (**ando2021**; **TODO_text_normalization_reference**). All subtitle files were normalized into cleaned text using a custom script. The cleaning rules implemented included:

- Removing HTML and formatting tags (e.g., `<i>`, brace tags).
- Removing speaker labels (e.g., `Speaker 1:`, `NARRATOR:`).
- Removing non-speech annotations inside brackets or parentheses when they represent events such as `[laughter]`, `(applause)`, `[music]`, or `(inaudible)`.
- Normalizing whitespace and punctuation (e.g., collapsing multiple spaces, removing spaces before punctuation).

The script produced a transcript file where one cleaned line per subtitle cue with the timestamp which was useful for later alignment between subtitle cues and speech segments (**TODO_alignment_reference**). The timestamp was used to seperate the long audio format into chunks (**TODO_alignment_reference**).

### 3.5.3 Audio Standardization to 16 kHz Mono

The downloaded audio may came with different format based on the download option and video quality. To ensure the compatibility with all selected model, the audio was standardized into 16 kHz Mono (**mcfee_2025_15006942**;

**soundfile**). This was also to make sure the fairness across all ASR model families when compare to each other (**TODO_standardization_reference**). The standardized choosen was **16 kHz sample rate, mono channel, 16-bit PCM** because this format was a standard configuration used in Kaldi recipes and also aligned with typical front-end assumptions in many end-to-end ASR pipelines (**kaldi2011**; **TODO_end2end_frontend_reference**).

Listing 3.3: Audio resampling to 16 kHz

```
import librosa, soundfile as sf

def resample_16khz(in_wav, out_wav, target_sr=16000):
    y, sr = librosa.load(in_wav, sr=None)
    if sr != target_sr:
        y = librosa.resample(y, orig_sr=sr, target_sr=target_sr)
        sr = target_sr
    sf.write(out_wav, y, sr)
```

### 3.5.4 Speech Segmentation using WebRTC VAD

The downloaded audio was averaging between 20 minutes to 30 minutes long. This duration was not suitable to be used as trining data because it was chalenging for the model to learn effectively from very long utterances, and it also increased memory usage and training time (**webrtc**; **TODO_long_utterance_reference**). To deal with this issue, WebRTC voice activity detection (VAD) was used to split a long audio files into chunks audio that suitable to be used in training phase (**webrtc**). The segmentation strategy was:

- Audio was split into 30 ms frames (**webrtc**).
- Speech frames were grouped into continuous speech regions (**webrtc**).
- A region was closed when silence exceeded a threshold 500 ms (**webrtc**).
- Very short segments were removed $\leq$2 s (**TODO_vad_threshold_reference**).
- Very long segments were further sliced into $\leq$20 s chunks (**TODO_vad_threshold_reference**).

The created a lot of short audio files and to manage all the chunks of file, below

naming convention was used to make sure the original audio name still available

`<originalname>_chunk_000.wav`, `<originalname>_chunk_001.wav`, ...

This naming convention was later reused as utterance IDs in the transcript mapping and manifests.

### 3.5.5 Remove non speaker audio

To improve the quality of the dataset, a filtering step was applied to remove the audio that did not have any Japanese speaker in it (**TODO_audio_filtering_reference**). This was because this audio became noise and could cause the the quality of the training data become lower (**TODO_audio_filtering_reference**). The type of audio that removed was audio clips that dominated by applause/laughter, heavy background noise, or non-Japanese speech (**clap**; **TODO_audio_filtering_reference**). A python script was created to filter out this type of audio by applied three checks:

- **Speaker structure check (pyannote diarization):** clips were preferred when a single dominant speaker is present (high dominant speaker ratio) (**TODO_pyannote_reference**).
- **Acoustic event detection (CLAP zero-shot audio classification):** clips with high confidence of laughter or clapping were removed based on threshold scores (**clap**).

Listing 3.4: Automatic audio quality filtering

```
diar = Pipeline.from_pretrained("pyannote/speaker-diarization-3.1",
    use_auth_token=HF_TOKEN)
clap = hf_pipeline("zero-shot-audio-classification",
    model="laion/clap-htsat-fused",device=0 )

for audio_path in wav_files:
    # keep only clear speech (no laughter/clapping)
    dom_ratio = dominant_speaker_ratio(diar(audio_path))
    scores = clap(audio_path,
```

```
        candidate_labels=["people laughing", "applause or hand clap"],
        top_k=None)
    clear = ((dom_ratio >= 0.85) and (score(scores, "laugh") < 0.30)
        and (score(scores, "clap") < 0.30))

    if not clear:
        os.remove(audio_path)
```

Files failing the clarity or language criteria were deleted automatically, leaving a cleaner set of speech segments for the downstream ASR experiments (**TODO_audio_filtering_reference**).

### 3.5.6 Transcript-to-Audio Mapping and Manifest Preparation

The next step after splitting the audio into chunks and cleaning the subtitle was to align the audio with a matching reference transcript to enable supervised training and evaluation (**TODO_alignment_reference**). The cleaned subtitle text served as the transcript source and the final transcript mapping was stored in a simple format like this:

`utt_id: transcript`

where `utt_id` matched the chunk filename and the `transcript` was the clean subtitle file that has been extracted the value and aligned based on the timestamp in the subtitle file (**TODO_alignment_reference**). After reference transcript was aligned with audio file, for training and evaluation reproducibility, metadata manifests were generated containing:

- `utt_id` (utterance identifier),
- `wav` (absolute path to the audio file),
- `duration` (in seconds),
- `transcript` (cleaned reference text).

### 3.5.7 Train/Validation/Test Splits

The final dataset was divided into three splits to support model development and unbiased evaluation (**TODO_dataset_splitting_reference**). The split ratio used

was 80% training, 10% validation, and 10% testing. The script implemented a two-step split using train test split (**TODO_sklearn_train_test_split**):

1. Split the full set into 80% train and 20% temporary.
2. Split the temporary set into 50% validation and 50% test (10% each).

The data was split like this beacuse the training set must be large enough for the models to learn the language patterns, while the validation set was required to tune hyperparameters and select checkpoints without leaking information from the test set (**TODO_dataset_splitting_reference**). The test set was kept completely unseen until the final evaluation to provide an unbiased estimate of real-world transcription performance (**TODO_dataset_splitting_reference**).

### 3.6 Model Training and Fine Tune Phase

Table 3.5 showed the fourth phase of this project where cleaned audio and transcript earlier was used to train or fine tune models. The main objective of this phase was to make sure each model was trained using the same data split so that the evaluation in the next phase was fair and consistent (**TODO_controlled_comparison_reference**).

Table 3.5
Model Training and Fine Tune Phase

| Phase | Activities | Deliverables |
|---|---|---|
| **Model Training and Fine Tune** | • Trained GMM-HMM<br>• Trained CRDNN-CTC<br>• Fine-tuned Whisper variants<br>• Set decoding parameters<br>• Saved configs and checkpoints | Chapter 4<br>• Models trained<br>• Checkpoints saved<br>• Settings recorded<br>• Outputs generated |

There model was used in this phase that was which was a traditional Kaldi GMM–HMM, an end-to-end CRDNN–CTC using SpeechBrain framework that was trained from zero while a transformer encoder–decoder Whisper model from OpenAI was fine tuned using the same dataset (**kaldi2011**; **ravanelli2021speechbraingeneralpurposespeechtoolkit**; **speechbrain_v1**; **radford2023robust**; **wolf2020huggingfacestransformersstateoftheartnatural**).

### 3.6.1 GMM–HMM Training (Kaldi)

The first model family was the traditional Kaldi-style GMM–HMM system (**kaldi2011**). This model was trained using a standard Kaldi recipe flow which started from feature extraction, followed by acoustic model training in multiple stages (mono, tri1, tri2, tri3) (**kaldi2011**). In this experiment, MFCC features with CMVN normalization were used because it was a common baseline setup in Kaldi and suitable for classical HMM-based modelling (**kaldi2011**; **TODO_mfcc_cmvn_reference**).

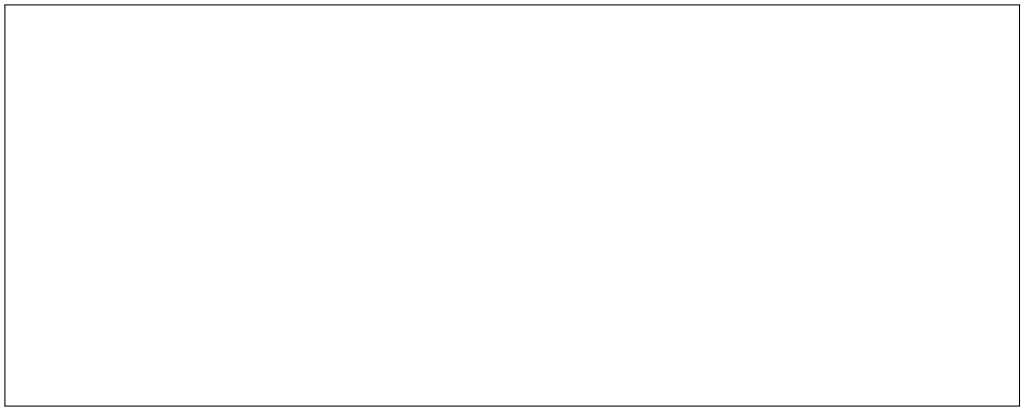

Figure 3.2 Placeholder: Model training workflow for GMMHMM

#### 3.6.1.1 Lexicon, Dictionary, and Language Preparation

A pronunciation dictionary and lexicon were prepared using a custom script, `prepare_dict.py`. The script extracted a vocabulary list from the training transcripts and generated a grapheme-based lexicon by converting each Japanese token into Hepburn romanisation using `pykakasi` (**pykakasi**). The romanised output was normalised to lowercase and filtered to `a-z` characters only. Each word was then mapped to a sequence of letters, which served as the basic pronunciation units (phones) (**TODO_grapheme_lexicon_reference**).

Listing 3.5: Grapheme-based lexicon generation (Japanese → Hepburn letters)

```
from pykakasi import kakasi

k = kakasi()
k.setMode('J','a'); k.setMode('K','a'); k.setMode('H','a'); k.setMode('r
    ','Hepburn')
```

```
conv = k.getConverter()

for w in vocab:  # unique tokens from training transcripts
    roma = "".join(x["hepburn"] for x in conv.convert(w)).lower()
    phones = list(re.sub(r"[^a-z]", "", roma))      # keep a-z only
    if phones:
        lexicon.write(f"{w} {' '.join(phones)}\n")  # lexicon
        phone_set.update(phones)                    # nonsilence_phones
```

Next, an unknown token <unk> was inserted into the lexicon to handle out-of-vocabulary (OOV) terms during decoding (**TODO_oov_reference**). This was for silent between speech or unknown features when decoding. After that, a language directory `data/lang` which contained symbol tables and language resources, was created using kaldi internal receipe (**kaldi2011**). The output of this step was a complete language directory that included `words.txt`, `L.fst`, and other required files used by Kaldi for graph construction (**kaldi2011**).

#### 3.6.1.2 Feature Extraction (MFCC + CMVN)

MFCC (Mel-Frequency Cepstral Coefficients) was a compact numeric features that extracted from audio to represent the way human hear sound (**TODO_mfcc_reference**). It converted audio into numerical value and used as input feature in the ASR system (**TODO_mfcc_reference**). MFCC features were extracted for train, dev, and test sets using `steps/make_mfcc.sh` from kaldi receipe (**kaldi2011**).

After that, CMVN (Cepstral Mean and Variance Normalization) statistics were computed using `steps/compute_cmvn_stats.sh` (**kaldi2011**). CVMN was a normalization step that applied to the MFCC to make the features more consistent by removing channel/mic/loudness differences (**TODO_cmvn_reference**). It also applied variance normalization for scale the unit variance and reduced scale differences (**TODO_cmvn_reference**).Below was the script to run MFCC and CMVN:

Listing 3.6: Grapheme-based lexicon generation (Japanese → Hepburn letters)

```
echo "=== Extracting MFCCs & CMVN ==="
```

```
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj "$NJ_MFCC_TRAIN"
  --cmd run.pl data/train exp/make_mfcc/train mfcc
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj "$NJ_MFCC_DEV"
  --cmd run.pl data/dev   exp/make_mfcc/dev   mfcc
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj "$NJ_MFCC_TEST"
  --cmd run.pl data/test  exp/make_mfcc/test  mfcc

steps/compute_cmvn_stats.sh data/train exp/make_mfcc/train mfcc
steps/compute_cmvn_stats.sh data/dev   exp/make_mfcc/dev   mfcc
steps/compute_cmvn_stats.sh data/test  exp/make_mfcc/test  mfcc
```

#### 3.6.1.3 Acoustic Model Training Stages

To train the acoustic model, the training was done gradually to improve the performance (**kaldi2011**). The acoutic model was train using 4 stages as described below:

- **Monophone (mono):** a baseline context-independent model trained using `train_mono.sh` (**kaldi2011**).
- **Triphone 1 (tri1):** a delta-based triphone model trained using `train_deltas.sh` (**kaldi2011**).
- **Triphone 2 (tri2):** a triphone model with LDA+MLLT transforms trained using `train_lda_mllt.sh` (**kaldi2011**).
- **Triphone 3 (tri3):** a speaker-adaptive training (SAT) model trained using `train_sat.sh` (**kaldi2011**).

Between each stage, the training set was aligned using `align_si.sh` to produce improved alignments for the next model (**kaldi2011**). This staged approach was important in Kaldi because better alignments usually led to better acoustic models (**kaldi2011**). Below was the snippet of the bash script used to run the training receipe from Kaldi library.

Listing 3.7: Audio resampling to 16 kHz

```
echo "=== Training mono ==="
steps/train_mono.sh --nj "$NJ_TRAIN" --cmd run.pl \
  data/train data/lang exp/mono
```

```
4
5  echo "=== Aligning with mono -> mono_ali ==="
6  steps/align_si.sh --nj "$NJ_ALIGN" --cmd run.pl \
7    data/train data/lang exp/mono exp/mono_ali
8
9  echo "=== Training tri1 (deltas) ==="
10 steps/train_deltas.sh 2000 10000 \
11   data/train data/lang exp/mono_ali exp/tri1
12
13 echo "=== Aligning with tri1 -> tri1_ali ==="
14 steps/align_si.sh --nj "$NJ_ALIGN" --cmd run.pl \
15   data/train data/lang exp/tri1 exp/tri1_ali
16
17 echo "=== Training tri2 (LDA+MLLT) ==="
18 steps/train_lda_mllt.sh 2500 15000 \
19   data/train data/lang exp/tri1_ali exp/tri2
20
21 echo "=== Aligning with tri2 -> tri2_ali ==="
22 steps/align_si.sh --nj "$NJ_ALIGN" --cmd run.pl \
23   data/train data/lang exp/tri2 exp/tri2_ali
24
25 echo "=== Training tri3 (SAT) ==="
26 steps/train_sat.sh 3500 20000 \
27   data/train data/lang exp/tri2_ali exp/tri3
```

#### 3.6.1.4 Language Model Construction and Decoding Graph

For decoding using a 3-gram, a language model was created using KenLM (**kenlm**). An external language model from wikipedia dump was used because it had a huge number of vocabluary to cover huge amount of language pattern (**TODO_wikipedia_lm_reference**). After that `lmplz` was used to produce an ARPA LM, and Kaldi tools were used to convert the ARPA file into `G.fst` (**kenlm**; **kaldi2011**).

Finally, decoding graphs were created for each acoustic model using `utils/mkgraph.sh` (**kaldi2011**). This produced graphs such as `exp/mono/graph`, `exp/tri1/graph`, and so on, which were used for decoding the test set (**kaldi2011**).

Listing 3.8: Audio resampling to 16 kHz

```
echo "=== Making decoding graphs for each system ==="
utils/mkgraph.sh data/lang exp/mono exp/mono/graph
utils/mkgraph.sh data/lang exp/tri1 exp/tri1/graph
utils/mkgraph.sh data/lang exp/tri2 exp/tri2/graph
utils/mkgraph.sh data/lang exp/tri3 exp/tri3/graph
```

### 3.6.2 CRDNN–CTC Training (SpeechBrain)

The second family model was from neural network model and CRDNNCTC model has been choosen based on the literature review (**chen2020streaming**; **ravanelli2021speechbraingeneralpurposespeechtoolkit**). This model was trained using SpeechBrain framework with the goal was to learn the direct mapping from acoustic feature and output character sequesnce (**speechbrain_v1**). For the decoder setup, CTC was used because Japanese transcripts could be handled naturally as a sequence of characters, and it avoided the need for word segmentation (**chen2020streaming**).

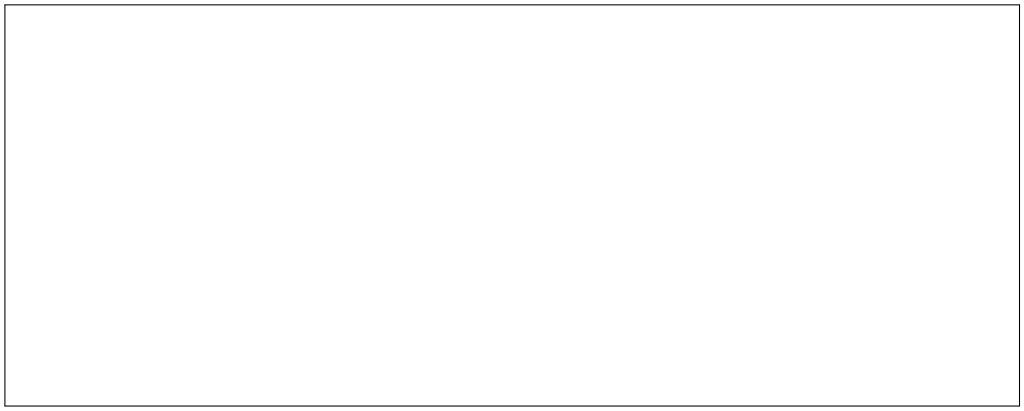

Figure 3.3 Placeholder: Model training workflow for CRDNN

#### 3.6.2.1 Manifest Preparation

Similar to GMM-HMM setup earlier, CRDNN-CTC setup also utilized the same CSV manifests that included `ID`, `wav`, `duration`, and `transcript` (**TODO_manifest_best_practice**). The manifests were generated from the pre-

pared wav directory and transcript mapping file. The same 80/10/10 split strategy train/valid/test was used (**TODO_dataset_splitting_reference**).

#### 3.6.2.2 Character Vocabulary (CTC Token Set)

Next step was to create a character vocabuluary from the training script by scanning all characters in the training set and writing them into a charset file (**TODO_ctc_vocab_reference**). The vocabulary file began with a special `<blank>` token to represent the CTC blank symbol (**graves2013**). After that this charset was loaded into SpeechBrain using `CTCTextEncoder` (**speechbrain_v1**). This step was important because CTC required a fixed label set, and the model output layer size depended directly on the number of characters in this vocabulary (**graves2013**). By generating the charset from the training transcripts, the label set stayed consistent with the dataset and reduced unexpected errors when unseen symbols appeared during training (**TODO_ctc_vocab_reference**).

Listing 3.9: Grapheme-based lexicon generation (Japanese → Hepburn letters)

```
# (1) Build charset.txt (CTC vocab)
import pandas as pd, unicodedata as ud
from collections import Counter
from pathlib import Path

df = pd.read_csv("manifests/jsut_train.csv")
chars = Counter(ch for t in df.transcript.astype(str)
                for ch in ud.normalize("NFKC", t).strip() if ch != " ")

Path("tokenizer").mkdir(exist_ok=True)
with open("tokenizer/charset.txt", "w", encoding="utf-8") as f:
    f.write("<blank>\n" + "\n".join(sorted(k for k in chars if k.strip()
       )) + "\n")
```

#### 3.6.2.3 Model Configuration and Hyperparameters

The CRDNN model hyperparameters were defined in `hyperparams.yaml` (**speechbrain_v1**). In this experiment, filterbank (FBank) features with mean-

variance normalization were used (**speechbrain_v1**; **TODO_fbank_reference**). The architecture consisted of CNN blocks, followed by bidirectional LSTM layers, and a DNN block before a final linear layer that projected to the CTC vocabulary size (**speechbrain_v1**). The optimizer used was Adam with a learning rate of $1 \times 10^{-3}$, and learning rate scheduling was handled using NewBob scheduling based on validation improvement (**speechbrain_v1**; **TODO_newbob_reference**). Dropout was also applied to reduce overfitting because the dataset size was limited compared to large-scale ASR corpora (**TODO_regularization_reference**). In addition, batch sizes for training and validation were configured separately to make sure validation could run stably under limited GPU memory (**TODO_batch_size_reference**).

#### 3.6.2.4 Training Procedure and Checkpointing

A custom SpeechBrain `Brain` class (`ASRBrain`) was implemented to define forward computation, CTC loss, and CER tracking during validation (**speechbrain_v1**). During training, checkpoints were saved using SpeechBrain checkpointer so that the best model could be recovered and used during evaluation (**speechbrain_v1**). The model was trained for a fixed number of epochs, and validation CER was monitored across epochs. The learning rate was automatically adjusted when validation improvement became small, which helped stabilize training and avoided over-updating the model (**speechbrain_v1**; **TODO_lr_schedule_reference**). At the end of training, the best checkpoint was selected based on validation performance (**speechbrain_v1**).

### 3.6.3 Whisper Fine-Tuning (Transformer Encoder–Decoder)

The third model family was Whisper, a transformer encoder–decoder ASR model from OpenAI (**radford2023robust**). In this research, multiple Whisper variants were fine tuned (tiny, base, small, medium, and large-turbo) (**radford2023robust**). Whisper was fine tuned using supervised learning by providing audio features as inputs and reference transcripts as labels (**radford2023robust**).

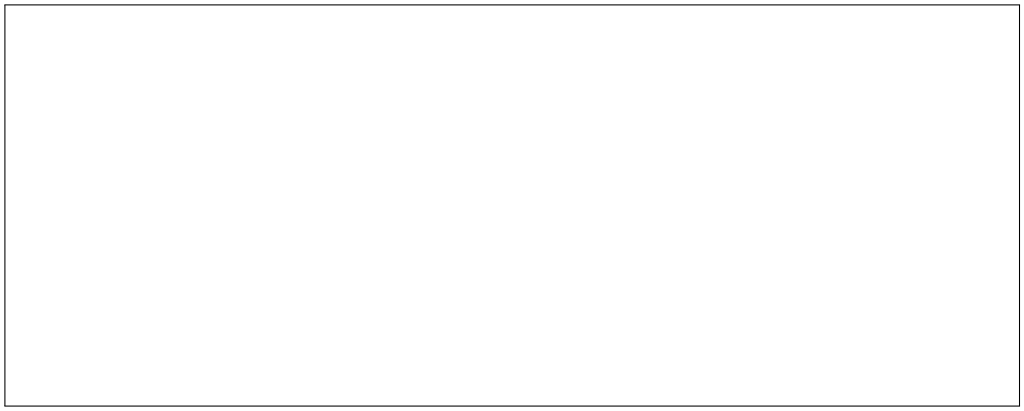

Figure 3.4 Placeholder: Model training workflow for Whisper

#### 3.6.3.1 Dataset Preparation for Fine-Tuning

The dataset was prepared by converting `transcript_utf8.txt` into a DataFrame that mapped each utterance ID to its corresponding `.wav` path and transcript. The data was shuffled using a fixed random seed for reproducibility and split into training, validation, and testing sets (**TODO_dataset_splitting_reference**). Each split was stored as CSV files (`train.csv`, `valid.csv`, `test.csv`) and later loaded using HuggingFace `datasets` (**wolf2020huggingfacestransformersstateoftheartnatural**).

#### 3.6.3.2 Feature Extraction and Label Encoding

Whisper used log-Mel filterbank features extracted by the Whisper feature extractor (**radford2023robust**). In preprocessing, each audio sample was resampled to 16 kHz if needed and converted into `input_features` (**radford2023robust**). The transcript text was tokenized using Whisper tokenizer to generate `labels` (**radford2023robust**). To avoid extremely long samples affecting training stability, audio longer than a fixed threshold (example: 30 seconds) was filtered out before training (**TODO_long_sample_filter_reference**).

#### 3.6.3.3 Fine-Tuning Configuration

For fine tuning, a pretrained Whisper checkpoint (example: `openai/whisper-base`) was loaded (**radford2023robust**). The decoder was configured with Japanese language prompts using

forced decoder IDs (`language="japanese", task="transcribe"`) (**wolf2020huggingfacestransformersstateoftheartnatural**). Gradient checkpointing-related settings were applied by setting `use_cache=False` (**wolf2020huggingfacestransformersstateoftheartnatural**). Training used AdamW optimizer with a small learning rate (example: $1 \times 10^{-5}$) and cosine learning rate scheduling with warmup (**TODO_adamw_cosine_reference**). Mixed precision (FP16) training was enabled when running on GPU to speed up training and reduce memory usage (**TODO_fp16_reference**).

Listing 3.10: Grapheme-based lexicon generation (Japanese → Hepburn letters)

```
from transformers import WhisperProcessor,
    WhisperForConditionalGeneration

MODEL_NAME = "openai/whisper-medium"  # later you can try tiny/base/
    medium/etc.

processor = WhisperProcessor.from_pretrained(MODEL_NAME)
model = WhisperForConditionalGeneration.from_pretrained(MODEL_NAME)

# configure language + task
forced_decoder_ids = processor.get_decoder_prompt_ids(
    language="japanese",
    task="transcribe",
)
model.config.forced_decoder_ids = forced_decoder_ids
model.config.suppress_tokens = []    # often helps for fine-tuning
model.config.use_cache = False       # must be False for gradient
    checkpointing
```

#### 3.6.3.4 Training Loop and Validation

Training was performed for 10 epochs using mini-batch gradient updates (**TODO_training_loop_reference**). During training, the loss value was monitored at each step and average loss was computed at the end

of each epoch. After training, validation decoding was performed using `model.generate()` and metrics were computed using WER and CER (**wolf2020huggingfacestransformersstateoftheartnatural**; **jiwer_2025**). The model and processor were then saved for later decoding and test evaluation (**wolf2020huggingfacestransformersstateoftheartnatural**).

#### 3.6.3.5 Model Saving and Inference Check

After fine-tuning completed, the model weights and processor configuration were saved using `save_pretrained` (**wolf2020huggingfacestransformersstateoftheartnatural**). A quick inference check was performed by decoding a sample from the test split and comparing the predicted transcription against the ground truth reference. This step was important to confirm that the model and tokenizer were saved correctly and could be loaded again for the evaluation phase (**wolf2020huggingfacestransformersstateoftheartnatural**).

## 3.7 Evaluation Phase

Table 3.6 showed the fifth phase of this project where all trained models were evaluated using the same test split and the same scoring protocol. The main objective of this phase was to measure transcription accuracy and decoding efficiency in a consistent and reproducible way so that comparisons across the three model families were fair (**jiwer_2025**; **TODO_rtf_reference**; **TODO_controlled_comparison_reference**). The activities carried out and the outcome deliverables were described in the table shown below.

Table 3.6
Evaluation Phase

| Phase | Activities | Deliverables |
|---|---|---|
| **Evaluation** | • Scored with CER and WER<br>• Measured speed using RTF<br>• Compared models and decoders | Chapter 4<br>• Results tables and plots<br>• Best model identified |

In this phase, each trained system produced hypotheses on the test split and those hypotheses were scored against the same reference transcripts. The evalu-

ation used three metrics that matched the research objectives that is character error rate (CER), word error rate (WER), and real-time factor (RTF) (**jiwer_2025**; **TODO_rtf_reference**).

### 3.7.1 Decoding and Hypothesis Generation

For the Kaldi GMM–HMM systems, decoding was performed using the test set features and the corresponding decoding graphs created earlier. Hypotheses were produced for each acoustic model stage (mono, tri1, tri2, tri3) using the standard Kaldi decoding pipeline (**kaldi2011**). For the CRDNN–CTC system, decoding was performed using CTC decoding and hypotheses were generated for the test set, including the beam decoding variant when configured (**speechbrain_v1**; **graves2013**). For Whisper variants, hypotheses were generated using the HuggingFace `model.generate()` decoding procedure, and the output token sequences were converted back into Japanese text using the Whisper tokenizer (**wolf2020huggingfacestransformersstateoftheartnatural**).

### 3.7.2 Scoring Protocol (CER and WER)

After hypotheses were generated, each prediction was aligned against the reference transcripts and scored using CER and WER (**jiwer_2025**). CER was computed by measuring the normalized Levenshtein distance between predicted and reference character sequences (**jiwer_2025**; **TODO_levenshtein_reference**). WER was computed using the same edit distance approach but applied at the word level after tokenization (**jiwer_2025**). The scoring protocol was applied consistently across all three model families so that differences in results reflected model performance rather than differences in evaluation handling (**jiwer_2025**; **TODO_controlled_comparison_reference**).

### 3.7.3 Speed Measurement (RTF)

Decoding speed was evaluated using real-time factor (RTF), where total decoding wall time was divided by total audio duration (**TODO_rtf_reference**). RTF values below 1.0 indicated faster-than-real-time decoding, while values above 1.0 indicated slower-than-real-time decoding (**TODO_rtf_reference**). RTF was recorded

for each system to compare computational efficiency, and the same measurement approach was applied across models under the same hardware environment to support fair comparisons (**TODO_rtf_reference**).

### 3.8 Discussion Phase

Table 3.7 showed the sixth phase of this project where the evaluation outcomes were interpreted and discussed in relation to the research questions and prior work. The main objective of this phase was to analyze the trends observed across the three model families, explain the trade-offs between accuracy and speed, and outline the limitations and future improvements (**TODO_discussion_method_reference**). The activities carried out and the outcome deliverables were described in the table shown below.

Table 3.7
Discussion Phase

| Phase | Activities | Deliverables |
|---|---|---|
| **Discussion** | • Explained main findings<br>• Linked to prior work<br>• Noted limitations<br>• Suggested future work | Chapter 5<br>• Findings discussed<br>• Limitations stated<br>• Recommendations given<br>• Future work proposed |

In this phase, the results from CER, WER, and RTF were analyzed to explain how each model family behaved under the same dataset and evaluation setup (**TODO_discussion_method_reference**). The discussion compared classical and neural approaches by focusing on how performance improved from mono to tri3 in the Kaldi pipeline, how CRDNN–CTC handled Japanese character sequences under end-to-end training, and how Whisper variants achieved different accuracy and speed trade-offs depending on model size (**kaldi2011**; **speechbrain_v1**; **radford2023robust**). The outcomes were linked back to prior work to show whether observed trends matched or differed from findings in the literature, especially regarding the advantages of pretrained transformer-based models for low-resource or domain-mismatched settings (**radford2023robust**; **bajo2024efficient**; **TODO_domain_mismatch_reference**).

## 3.9 Summary

This chapter described the methodology for comparing Japanese ASR across three model families. The study followed six phases (planning, prior work, data preparation, training, evaluation, and discussion) and used a single, consistent dataset pipeline to ensure fair comparison (**TODO_controlled_comparison_reference**). Japanese TEDx talks with manual subtitles were collected, transcripts were cleaned, audio was standardised to 16 kHz mono, and long recordings were segmented using WebRTC VAD, with low-quality non-speech clips filtered out (**yt-dlp**; **mcfee_2025_15006942**; **soundfile**; **webrtc**; **clap**; **TODO_pyannote_reference**). All models were trained on the same 80/10/10 splits and evaluated on the same test set using CER, WER, and RTF to compare accuracy and decoding speed (**jiwer_2025**; **TODO_rtf_reference**).

# CHAPTER FOUR
# RESULTS AND DISCUSSION

## 4.1 Introduction

In this chapter, the result of the workflow explained in Chapter 3 is presented. The result encompasses the steps from data collection and preprocessing, model training and fine-tuning, and finally evaluation on a held-out test split. Then, the result is organized based on the methodology section and the quantitative results from testing the trained and fine-tuned models.

The performance is measured using character error rate (CER), word error rate (WER), and real-time factor (RTF). Following the research objective explained in Chapter 1 and the gap identified in Chapter 2, the analysis in this chapter focuses on two goals. The first goal is measure and compare the accuracy of the selected models, and secothe nd goal is to measure how fast the model can generate output based on the inputted audio (**ando2021**).

## 4.2 Data Collection and Preprocessing Results

### 4.2.1 Data Source Selection Outcomes

This study gathered data from publicly available Japanese TEDx talks and only videos that contain manual subtitle will be selected from the playlist. Videos with auto-generated subtitles or subtitles in languages other than Japanese will be excluded to reduce transcript noise.

Table 4.1
Video selection outcomes for TEDx Japanese data collection.

| Item | Count |
|---|---|
| Playlist items queried | 1574 |
| Manual Japanese subtitles found | 862 |
| Auto-generated only / none | 712 |
| Download completed | 862 |

Since the data source only utilizes data from manual transcription, it is align with the testing objective that is to produce a readable and standardized transcript

while at the same time reducing the noise during training and evaluation. This is important because Japanese language can have words with different meaning with same pronunciation. If this kind of data used as training data, it will resulting in model performance to become worsen.

(**Koenecke2020**).

### 4.2.2 Transcript Cleaning and Normalization Outcomes

After the subtitle is downloaded, the raw data from the subtitle was normalized into a reference transcript that acts as the source of truth. This cleaning process involves removing formatting tags, removing speaker labels, and discarding non-speech annotations such as `[laughter]`, `(applause)`. The output from this process is a clean transcript with a timestamp. Another benefit of removing the labels is to reduce noise as things like bracket events will increase the error rate because when calculating the error number, the bracket event will be treated as a deletion from the reference transcript.

```
Timestamp: 00:00:01.000 --> 00:00:03.200
RAW      : [applause] みなさん、こんにちは。
CLEANED  : みなさん、こんにちは。

Timestamp: 00:00:03.200 --> 00:00:06.000
RAW      : Speaker 1: 今日はAIについて話します。
CLEANED  : 今日はAIについて話します。

Timestamp: 00:00:06.000 --> 00:00:09.000
RAW      : (笑) それでは始めましょう。
CLEANED  : それでは始めましょう。

Timestamp: 00:00:09.000 --> 00:00:12.000
RAW      : <i>重要なのは</i> 継続です。
CLEANED  : 重要なのは 継続です。

Timestamp: 00:00:12.000 --> 00:00:16.000
RAW      : [music] (拍手) 本日はありがとうございます。
CLEANED  : 本日はありがとうございます。
```

Figure 4.1 Raw vs cleaned transcript after preprocesing

As shown in Figure 4.1, the raw subtitle contains non-speech items like formatting symbols and additional annotations that will be very useful for human viewing the video but since it is not part of the intended script, it will cause noise. Another

thing to be taken into consideration is that the post-processing also needs to clean these kinds of annotations in the hypothesis transcript. This will improve the fairness when comparing these models because the CER/WER is measured based on the speech content rather than subtitle formatting. (**ando2021**)

### 4.2.3 Audio Standardization Outcomes

After the audio is downloaded, it will be converted into a consistent 16 kHz, mono, PCM16 format. The audio needs to be formatted to ensure compatibility when used in Kaldi, SpeechBrain, and Whisper training pipelines. By standardizing the sampling rate, it will reduce the differences caused by audio encoders and also 16 kHz, mono, PCM16 is the standard data expected from traditional pipelines (MFCC + CMVN) and modern neural pipelines (log-Mel filterbanks) (**kaldi2011**; **mcfee_2025_15006942**). This step also will support the "apples-to-apples" comparison goal stated in Chapter 1 by using the same data format across all models (**xu2023recent**). Figure 4.2 below shows the difference of downloaded and resampled audio.

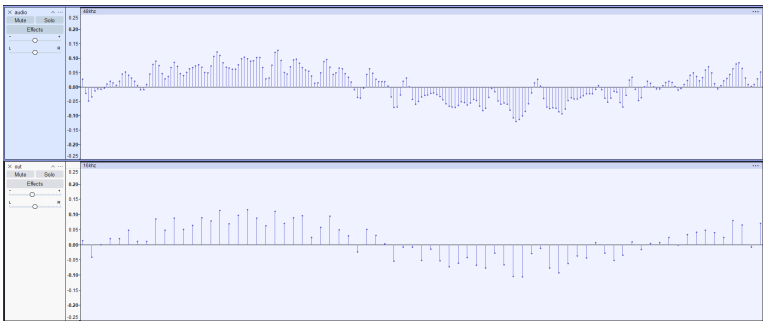

Figure 4.2 Example of audio standardization via resampling (44.1 kHz to 16 kHz).

### 4.2.4 VAD Segmentation Outcomes

The downloaded audio length is approximately 20 minutes long, which is not suitable for direct use as training data. By segmenting the long audio into shorter utterances, it will be more practical to use as training data. Additionally, models like Whisper, which is based on transformers, will cause memory issues because long sequences of training data will increase memory usage and decrease the stability of the models during training and fine-tuning. The segmentation process is done using a

lightweight segmentation mechanism widely used in speech preprocessing pipelines named WebRTC VAD (**webrtc**).

Table 4.2
VAD segmentation results.

| Item | Count / Value | Notes |
|---|---|---|
| Total chunks produced | 74802 | After segmentation and slicing |
| Chunks removed (too short) | 11881 | $\leq$2 s |
| Avg chunk duration (s) | 6.605 | After filtering short clips |
| Max chunk duration (s) | 20 | Enforced by slicing |

Table 4.2 above showed the summary of the data after segmentation. The average audio length became 6.6 seconds, which are closer to utterance-level speech than long-form audio. This will eliminate the memory usage problem mentioned earlier, as the longest audio is only 20 seconds. Another reason to chunk the audio is to prevent the RTF meafrom surement creating a spike because of limits when decoding the audio.

### 4.2.5 Non-Speaker / Low-Quality Audio Filtering Outcomes

The filtering process is carried out based on the requirement in Chapter 1, which states that the preprocessing method will heavily impact performance, especially for audio that contains too much noise, such as non-linguistic events or multi-speaker overlap (**latif2020**). This type of data needs to be removed because it may contain a weak or wrong subtitle alignment even when the segment has manual subtitles.

Table 4.3
Audio filtering outcomes (diarization + CLAP).

| Item | Count | Notes |
|---|---|---|
| Chunks before filtering | 62921 | Output from VAD |
| Removed by speaker-structure check | 13758 | Dominant speaker ratio threshold |
| Removed by clap/laughter threshold | 5596 | CLAP score thresholds |
| Chunks after filtering (final) | 43567 | Used to build manifests |

The result in the table above shows that after removing non-speaker and low-quality audio, only 43567 chunks remain from 62921. This shows that it is a trade-off between the quantity and the quality of the audio, where filtering will reduce the

number of dataset size but increase the segment with clean Japanese speech. The reliability of CER/WER is also increased because the test reference is less affected by non-speech artifacts and misalignment noise (**ando2021**).

### 4.2.6 Transcript-to-Audio Mapping and Manifest Outcomes

After the audio is segmented and cleaned, each chunk will be paired with the reference transcript from the cleaned subtitle earlier. From there, the manifest containing the audio and transcript data is created. The objective of creating the manifest is to improve reproducibility when splitting the data into train and test splits. This will also ensure that the differences in the end result are only due to the model itself and not on inconsistent data splits (**ravanelli2021speechbraingeneralpurposespeechtoolkit**).

| | ID | path | duration | transcript |
|---|---|---|---|---|
| 0 | 1367_Why_a_city_... | /content/data/wa... | 2.10 | なぜそこの女川町を選んだのか。 |
| 1 | 398_Career_educa... | /content/data/wa... | 3.69 | あんなに堂々と 生き生きと発表す... |
| 2 | 1250_1 _TEDxWakay... | /content/data/wa... | 3.81 | こういう救出はこの到着部隊から考... |
| 3 | 282_The_Regions_... | /content/data/wa... | 10.89 | これ人工衛星の話じゃないですか。... |
| 4 | 1336_Bringing_a_... | /content/data/wa... | 4.92 | 庭園 建物 様々なものづくりを発... |
| 5 | 1348_A_face_to_f... | /content/data/wa... | 3.90 | たくさんの人を山に案内する中で ... |
| 6 | 611_TEDxUSH_chun... | /content/data/wa... | 3.99 | 何か自分の変わる人生が 変わるき... |
| 7 | 1348_A_face_to_f... | /content/data/wa... | 5.13 | 森は仕事として関わる一部の人間に... |
| 8 | 1269_TEDxHaneda_... | /content/data/wa... | 11.82 | でも これって誰が悪いとか そう... |
| 9 | 658_The_challeng... | /content/data/wa... | 14.28 | ハイビスカスをつけて 踊りながら... |

Figure 4.3 Data after mapping audio and transcript into manifest format

Figure above shows some of the data from the manifest data. The manifest data contains four column that is ID, path, duration, and transcript. The ID data is used to differentiate each chunk from the main audio file name. Duration and path are the data of the audio file, keeping records of where the audio is stored and the duration of the audio file. Lastly, the transcript is the reference transcript that will be used as labelled data when training models and as a reference transcript when testing the models.

### 4.2.7 Train/Validation/Test Split Results

The final dataset was split into 80% training, 10% validation, and 10% testing. Table 4.4 reported split sizes.

Table 4.4
Dataset statistics and train/validation/test split summary.

| Split | #Utterances | Duration (hrs) | Average (s) |
|---|---|---|---|
| Train | 32852 | 63.3 | 6.941 |
| Valid | 4357 | 8.3 | 6.861 |
| Test | 4358 | 8.1 | 6.731 |

The split statistics show comparable average durations across train/valid/test, which reduces the risk that one split systematically contains longer or harder utterances. Keeping the test split fully held out supports a cleaner interpretation of generalization and makes the evaluation consistent with comparative reporting in prior Japanese ASR studies (**ando2021**; **Karita2021**).

## 4.3 Model Training and Fine-Tuning Results

### 4.3.1 Kaldi GMM–HMM Training Outcomes

Kaldi training consisted of four stages of training, which are mono, tri1, tri2, and tri3. The later stages were trained based on the previous stages' aligned data. This convention follows the Kaldi recipe logic, where increasing the context-dependent model will improve the alignment and feature-space transform. The consistent improvement aligns with established reports with other training recipes, such as CSJ, where later triphone and SAT will produce better accuracy gain (**kaldi2011**; **KaldiCSJResults2015**)

Table 4.5
Kaldi GMM–HMM training dynamics based on objective function improvement per frame. Peak improvements occur early and decrease toward near-zero values, indicating convergence.

| Stage | Peak impr./frame | Peak iter | Final impr./frame | Stable iter ($< 0.02$) |
|---|---|---|---|---|
| mono | 0.4289 | 1 | 0.0110 | 32 |
| tri1 | 0.2022 | 3 | 0.0026 | 30 |
| tri2 | 0.4820 | 1 | 0.0034 | 30 |
| tri3 | 0.2975 | 3 | 0.0034 | 30 |

The trend line in Table 4.5 above shows that most of the huge parameter updates occurred early in the training, then followed by small improvements until it hit a plateau and convergence was approached. This pattern is expected for GMM–HMM training and aligns with typical Kaldi diagnostics used to detect training stability and saturation (**kaldi2011**).

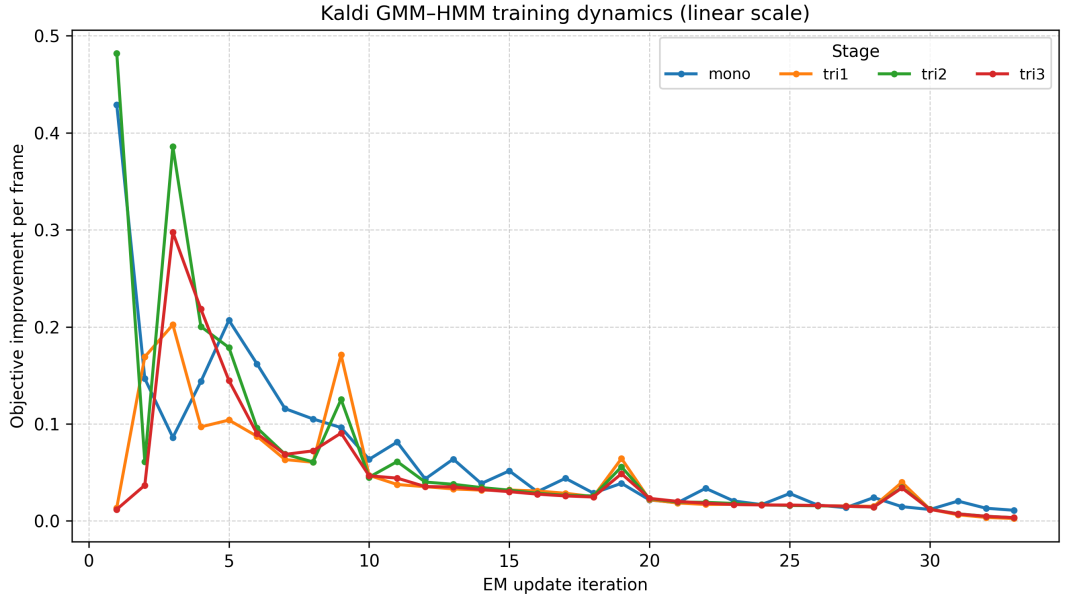


Figure 4.4 Kaldi GMM-HMM training dynamics in linear scale

### 4.3.2 CRDNN–CTC Training Outcomes

The CRDNN-CTC model was trained using the manifest prepared in the preprocessing phase and a character vocabulary derived from the training script. The training was done over 70 epochs, where the starting training loss was 5.44 at the start of the training and became 0.0775 at the end, with the lowest CER at 0.2272 during epoch 68. After that, the best checkpoint was selected using validation tracking and later evaluated on the test split (reported in Table 4.10). Table 4.6 summarized the epoch-by-epoch training dynamics from `data.txt`.

For the decoding part using CTC, it is well aligned with Japanese transcription due to its ability to use monotonic alignment without needing a pronunciation dictionary. The reference transcript also does not need to be tokenized into words, which is very useful for languages that do not have space between words in a sentence (**watanabe2018espnetendtoendspeechprocessing**).

Table 4.6
CRDNN–CTC training metrics across epochs.

| Epoch | Train loss | Valid loss | CER |
|---|---|---|---|
| 1 | 5.4400 | 5.2833 | 0.9983 |
| 2 | 4.0100 | 3.0769 | 0.6428 |
| 5 | 1.9100 | 1.6922 | 0.4555 |
| 10 | 1.1200 | 1.3025 | 0.3372 |
| 15 | 0.7760 | 1.0662 | 0.2906 |
| 20 | 0.5770 | 1.0610 | 0.2783 |
| 25 | 0.3130 | 1.0613 | 0.2629 |
| 30 | 0.2250 | 1.0297 | 0.2483 |
| 35 | 0.1740 | 1.0951 | 0.2404 |
| 40 | 0.1360 | 1.1158 | 0.2363 |
| 50 | 0.2050 | 1.0767 | 0.2437 |
| 60 | 0.1090 | 1.1674 | 0.2352 |
| 70 | 0.0775 | 1.2433 | 0.2309 |

Training curves below show that learning occurs very strongly in the early stages and then followed by small changes later in the training as the epochs increase. The error rate value starting from a very high number that is 0.9983 which shows that the prediction is basically random at this point of training. As the train loss and valid loss decrease, the CER also decreased. The figure below shows that training and validation loss as the epoch increases, with valid loss becoming flat first, followed by training loss. The figure also shows that CER is decreasing aggressively at the start of the training and then becomes stable in the middle and maintains the trend until the end of the training.


Figure 4.5 Training vs validation loss

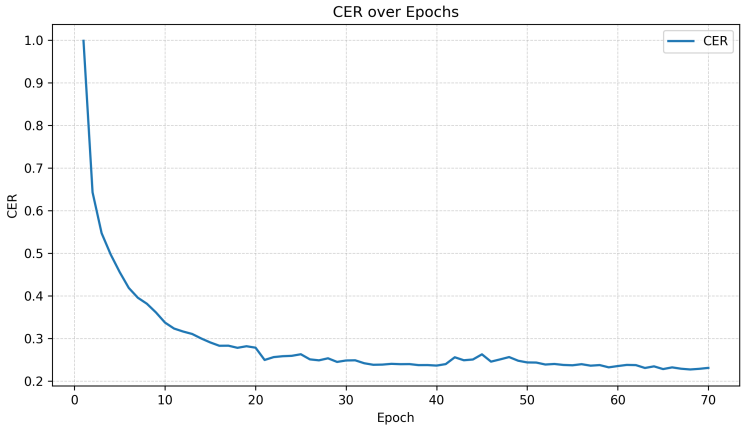


Figure 4.6 CER over Epochs

### 4.3.3 Whisper Fine-Tuning Outcomes

The Whisper model fine-tuning is using a pre-trained model provided by OpenAI, where the model is built upon a large-scale weak supervision, which has been shown to improve robustness under domain mismatch and reduce the reliance on task-specific feature engineering (**radford2023robust**). Previous studies from **bajo2024efficient<empty citation>** show that a notable improvement is it achieved when fine-tuning the base multi-language models by adapting to the Japanese language. The fine-tune stability can be seen from the loss vs epoch values below.

Table 4.7
Whisper fine-tuning loss across epochs (lower is better).

| Epoch | tiny | base | small | medium | turbo |
|---|---|---|---|---|---|
| 1 | 0.6626 | 0.3683 | 0.1988 | 0.0747 | 0.2137 |
| 2 | 0.3636 | 0.2106 | 0.0820 | 0.0415 | 0.1361 |
| 3 | 0.2862 | 0.1466 | 0.0413 | 0.0236 | 0.0911 |
| 4 | 0.2327 | 0.1034 | 0.0216 | 0.0130 | 0.0591 |
| 5 | 0.1930 | 0.0731 | 0.0115 | 0.0071 | 0.0362 |
| 6 | 0.1636 | 0.0520 | 0.0061 | 0.0035 | 0.0198 |
| 7 | 0.1426 | 0.0380 | 0.0033 | 0.0016 | 0.0095 |
| 8 | 0.1289 | 0.0296 | 0.0016 | 0.0012 | 0.0041 |
| 9 | 0.1205 | 0.0252 | 0.0008 | 0.0006 | 0.0016 |
| 10 | 0.1173 | 0.0234 | 0.0006 | 0.0003 | 0.0003 |

The larger model, like medium and large-turbo, becomes stable faster at only epoch 5, while the smaller model, like tiny, becomes stable become stable after epochs 9. Also worth noting that the loss value after epochs 10 is different eventhough the

training become stable showing that the larger model is more quick to adapt to the training data compared to the smaller model.

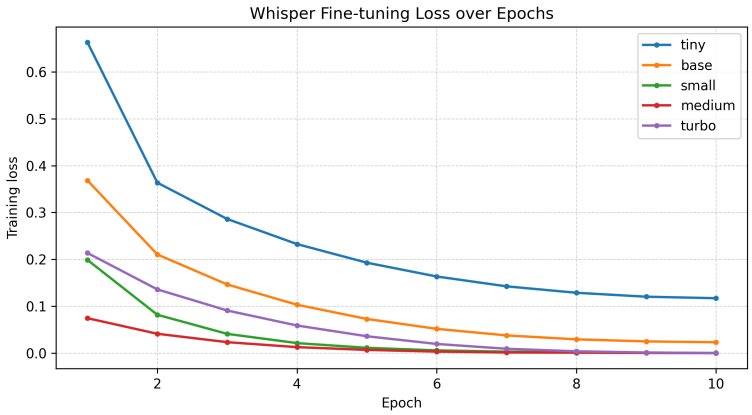


Figure 4.7 Whisper Fine-tuning loss over Epochs

## 4.4 Evaluation Results (CER, WER, and RTF)

### 4.4.1 Overall Performance Comparison

Table 4.8 above shows that Whisper achieved the highest accuracy with the lowest CER and WER overall, followed by CRDNN–CTC which is second in accuracy but has the highest decoding speed. And then, the traditional model Kaldi shows a good result and strong gains from staged training but remains less accurate compared to the neural approaches.

Based on the literature review in Chapter 2, the pattern is consistent with past studies where newer end-to-end models that build using transformers perform better compared to the older pipeline in terms of accuracy. However, architectural and decoding choices have a strong impact on latency (**xu2023recent**; **radford2023robust**).

Table 4.8
Overall comparison of ASR systems on the test split (lower is better for CER/WER/RTF).

| Model | CER (%) | WER (%) | RTF |
|---|---|---|---|
| GMMHMM-mono | 49.49 | 53.43 | 0.6500 |
| GMMHMM-tri1 | 24.95 | 29.31 | 0.3410 |
| GMMHMM-tri2 | 21.30 | 25.59 | 0.2480 |
| GMMHMM-tri3 | 19.48 | 23.57 | 0.2510 |
| CRDNNCTC-1 | 15.23 | 22.87 | **0.0129** |
| CRDNNCTC-2 (beam) | 15.12 | 22.70 | 0.0147 |
| Whisper-tiny | 10.72 | 11.61 | 0.0297 |
| Whisper-base | 5.67 | 5.89 | 0.0413 |
| Whisper-small | 4.34 | 4.30 | 0.0737 |
| Whisper-medium | 4.29 | **4.22** | 0.1605 |
| Whisper-turbo | **4.28** | 4.23 | 0.0959 |

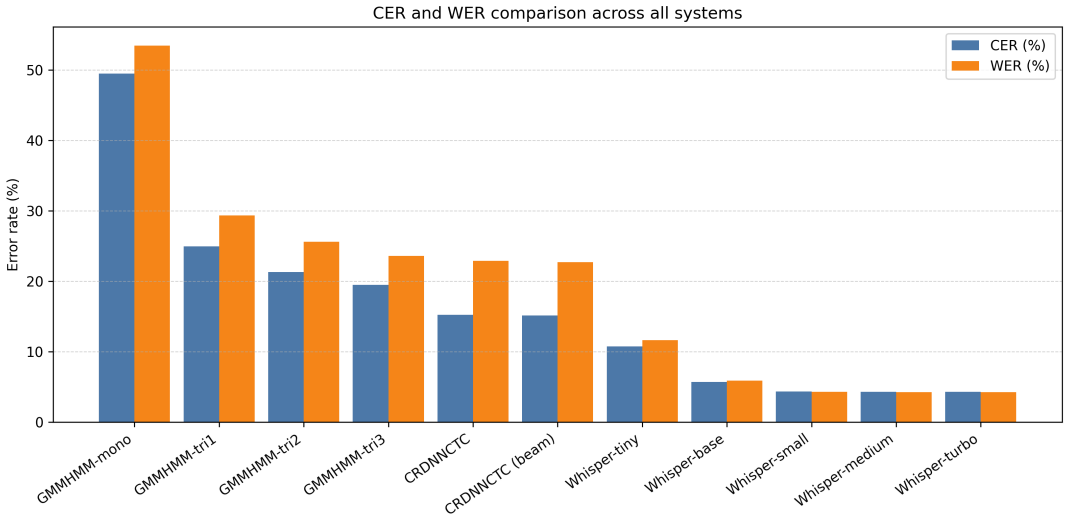


Figure 4.8 CER and WER comparison across all system

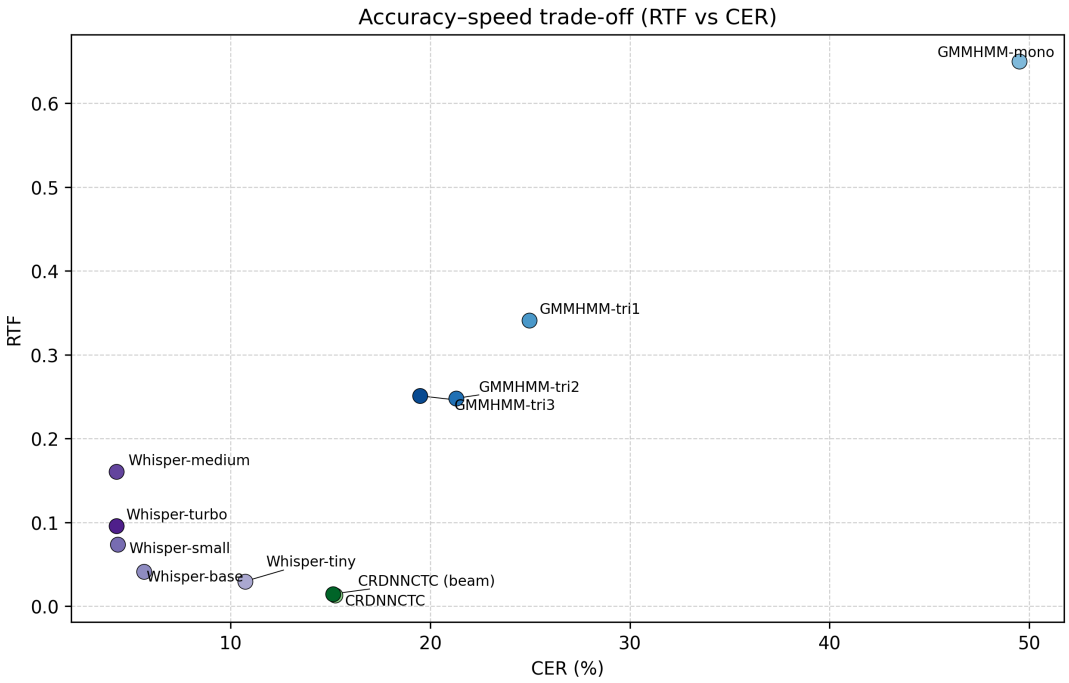


Figure 4.9 Accuracy speed trade-off

The ranking for accuracy is consistent across both CER and WER, where Whisper variants are the best in terms of precision while placing CRDNN–CTC a little bit behind in terms of accuracy, which sits in the middle, and Kaldi is the weakest, even at tri3. In contrast, from the RTF performance, CRDNN–CTC runs the fastest, with Whisper slowing down as model size increases.

### 4.4.2 Kaldi GMM–HMM Results (mono to tri3)

Starting at mono and moving through tri3, the performance of GMM-HMM steadily increases. As the model advances, CER drops and falls from nearly 49.49% to 19.48%, a drop of more than 60%. WER follows a similar trend, first decreasing from over 53.43% to 23.57%, cutting errors by roughly half. Most of the progress happens early, particularly between mono and tri1. After that point, each step only lowers the error rate by a small amount but continuously.

Table 4.9
Kaldi GMM–HMM results across training stages on the test split.

| Stage | CER (%) | WER (%) | RTF |
|---|---|---|---|
| mono | 49.49 | 53.43 | 0.6500 |
| tri1 | 24.95 | 29.31 | 0.3410 |
| tri2 | 21.30 | 25.59 | 0.2480 |
| tri3 | 19.48 | 23.57 | 0.2510 |

Smaller improvements from tri1 to tri3 show an expected behavior of conventional pipelines, where shifting to context-sensitive triphones results in notable progress. Additionally, subsequent enhancements via feature adjustments and SAT add modest improvements (**kaldi2011**; **KaldiCSJResults2015**). In terms of real-time usability, GMM-HMM shows that the model's performance in terms of RTF becomes better as efficiency holds despite deeper layers in GMM-HMM models.

### 4.4.3 CRDNN–CTC Results (Greedy vs Beam)

In the CRDNN-CTC results, the error rate decreased slightly when using beam decoding compared to greedy decoding. The decrease from 22.87% to 22.70% resulted in a 0.17% difference between both decoders. The decoding speed increased significantly by about 13.95% from greedy decoding to beam decoding. However, both versions of CRDNN-CTC achieved the fastest decoding time compared to other models.

Table 4.10
CRDNN–CTC results on the test split (greedy vs beam).

| Decoder | CER (%) | WER (%) | RTF |
|---|---|---|---|
| Greedy (CRDNNCTC-1) | 15.23 | 22.87 | **0.0129** |
| Beam (CRDNNCTC-2) | 15.12 | 22.70 | 0.0147 |

A slight difference in accuracy between greedy and beam decoding shows that the acoustic model has a greater influence, while beam search adds little gain. In this case, the performance leans more on the encoder's strength rather than advanced decoding methods. Prior work supports this pattern, which shows that greedy approaches often match beam results when language modeling plays a minor role (**graves2013**; **chen2020streaming**). What stands out is how fast both setups oper-

ate, delivering very low RTF values. Their speed makes them practical choices for real-time applications where delay matters.

### 4.4.4 Whisper Fine-Tuning Results (Model Size Effects)

Transcription accuracy reached its highest with Whisper fine-tuning. Moving up from tiny to base brought a clear improvement, where the WER dropped from 11.61% down to 5.89%. A further step to small model reduced the WER again, landing at 4.30%. Progress beyond that slowed, where medium only edged ahead by 1.86%, lowering WER to 4.22%. At the same time, processing lagged behind, and RTF climbed past 0.16. Yet, Whisper-turbo matched medium closely, hitting 4.23% WER while responding quicker than medium, with an RTF under 0.1.

Table 4.11
Whisper fine-tuning results by model size on the test split.

| Variant | CER (%) | WER (%) | RTF |
|---|---|---|---|
| tiny | 10.72 | 11.61 | 0.0297 |
| base | 5.67 | 5.89 | 0.0413 |
| small | 4.34 | 4.30 | 0.0737 |
| medium | 4.29 | **4.22** | 0.1605 |
| turbo | **4.28** | 4.23 | 0.0959 |

Results reveal how speed ties to accuracy where bigger models gain precision only until gains slow, demanding far more computing power. Earlier work supports this because pre-trained Transformers handle Japanese speech well, though size must suit real-world limits (**radford2023robust**; **bajo2024efficient**). Here, Whisper-turbo strikes a useful middle ground, holding near top-tier word error rates while cutting response time compared to Whisper-medium.

## 4.5 Discussion

### 4.5.1 Main Findings and Trade-offs

Overall finding that can be concluded is that Whisper models handled Japanese TEDx talks better than others. This can be seen from the CER and WER in speech recognition tasks. However, the real-time performance aspect was something else that was unexpected, as the CRDNN–CTC system processed audio faster

than amaking it ll, suitable for real-time usage. Training helped GMM-HMM a lot at each stage. However, when measured against deep learning systems, its performance stayed behind.

The overall outcomes align with the objective explained in Chapter 1 and with past studies in reviewed in Chapter 2. ASR systems for Japanese speech recognition perform better when they handle symbol patterns and draw on broad pre-trained knowledge. Meanwhile, older processing methods still offer clearer logic paths and organized learning setups. However, their precision tends to drop when used across changing environments (**Koenecke2020**; **ando2021**; **xu2023recent**).

### 4.5.2 Interpretation by Model Family

#### 4.5.2.1 Kaldi: staged training gains and diminishing returns

The increase in accuracy from mono to tri1 model is because the triphone model uses context-sensitive units, compared to mono models that use fixed units.The next gain is from training the models with tri2 and tri3, which results in a slightly increased performance. The increased gain in tri2 is due to better signal processing via LDA and MLLT. Speaker-specific adjustments during training also help to improve the models by implementing SAT in tri3. However, each subsequent step added less gain than the one before it, as limits were neared within the setup and model capabilities.

One key point from these findings is that the way to increase the accuracy of traditional models is by using step-by-step training. To further improve this kind of model, better lexicons, refined language modeling, or discriminative methods are required. The results shown from these models align with earlier studies where traditional Japanese ASR systems gain significantly from careful tuning of stages. However, the final accuracy score is still behind compared to advanced end-to-end models. Especially under difficult conditions, where newer Transformer-based models tend to perform better, a similar pattern also appears in studies conducted by (**ando2021**; **KaldiCSJResults2015**).

#### 4.5.2.2 CRDNN–CTC: decoding strategy impact

In CRDNN-CTC models, the differences in decoding between greedy and beam search were only slight. This is because most performance was gained from the acoustic model, and the chosen beam settings provided limited additional benefit relative to the added decoding cost. Across the tests, each CRDNN version ran faster than Whisper and Kaldi when measured using RTF. However, while beam tuning helped slightly, its extra computation did not justify the effort needed.

This also shows that CRDNN–CTC performs effectively when quick response times are crucial, like in live subtitles or spoken dialogue systems. Earlier studies on CTC-driven Japanese speech recognition support this outcome, demonstrating strong character error rates despite minimal context or continuous input modes (**chen2020streaming**).

#### 4.5.2.3 Whisper: strong accuracy with model-size cost

From the results produced by the Whisper models, a conclusion can be made that bigger models boosted Whisper's performance, yet slowed things down noticeably. The medium models version achieved the lowest WER, and the lowest CER is achieved by turbo models. However, when comparing these two models in terms of RTF, turbo models have an advantage compared to medium. The RTF becomes higher as the model size is larger because larger models need more resources to process the tokens and output the results when decoding.

The result is in line with the past study by **radford2023robust<empty citation>** where using a structured environment of a TEDx event, Whisper was able to achieve good results because of broad pre-trained systems able to manage sound variation efficiently, especially once adapted through targeted training data. However, one thing needs to be considered when choosing Whisper for real applications is that the resource usage is higher compared to other models.

### 4.5.3 Error Analysis

To complement aggregate metrics, this section can summarize recurring error patterns across systems (e.g., deletions under fast speech, named entities, subtitle mismatch, or segmentation boundary effects). Detailed examples can be included using **??**.

Across the qualitative examples (Tables 4.12–4.15), three recurring error patterns were observed. First, **function-word deletions and minor particles** were common in Kaldi and CRDNN outputs (e.g., missing small grammatical markers), which can occur under fast speech or reduced articulation and contribute disproportionately to CER when characters are missing. Second, **named entity and word-choice substitutions** appeared in the lower-performing systems (e.g., shorthand forms or near-homophones), reflecting limitations in lexical modeling and context resolution that are known challenges in Japanese due to ambiguous readings and context sensitivity (**curtin2020japanese**; **Ito2016End-to-end**; **ito2017**). Third, **formatting and normalization inconsistencies** (e.g., number rendering or punctuation/spacing variation) were visible across systems, which motivates the post-processing and text formalization emphasis discussed in Chapter 1 (**ando2021**).

Although Whisper produced the fewest errors overall, the examples show it still made small mistakes such as minor polite-form variation or subtle substitutions. This supports the view that even strong pretrained models may require additional normalization rules or domain-specific constraints to achieve fully standardized outputs for archival or reporting use cases (**radford2023robust**).

Table 4.12
Qualitative transcription example 1: Wording errors.

| System | Text / diff vs reference |
|---|---|
| Reference | 来週、東京大学でAIの安全性について講演します。 |
| Kaldi (tri3) | 来週[、→␣]東京大[-学-]でAIの安全[-性-]について[講→公]演します |
| CRDNN–CTC (best) | 来週、東京大学でAIの安全[-性-]について講演します |
| Whisper (best) | 来週、東京大学でAIの安全性について講演します |

Table 4.13
Qualitative transcription example 2: Polite-form variation.

| System | Text / diff vs reference |
|---|---|
| Reference | その結果をすぐに共有して、次の手順を決めましょう。 |
| Kaldi (tri3) | その結果[を→␣]すぐ[-に-]共有して[、→␣]次の手順[を→␣]決めましょう |
| CRDNN–CTC (best) | その結果をすぐ[-に-]共有して、次の手順を決め[ま→よ][-し-][-ょ-]う |
| Whisper (best) | その結果をすぐに共有して、次の手順を決め[ま→よ][-し-][-ょ-]う |

Table 4.14
Qualitative transcription example 3: Paraphrase.

| System | Text / diff vs reference |
|---|---|
| Reference | 要するに、私たちは失敗から学ぶ必要があります。 |
| Kaldi (tri3) | 要する[、→␣]に私たちは失敗から学ぶ[必→␣][要→ひ][+つ+][+よ+][+う+]があります |
| CRDNN–CTC (best) | [要→つ][す→ま][る→り][-に-]、私たちは失敗から学ぶ必要があります |
| Whisper (best) | 要するに、私たちは失敗から学ぶ必要があ[り→る][-ま-][-す-] |

Table 4.15
Qualitative transcription example 4: Numbering format.

| System | Text / diff vs reference |
|---|---|
| Reference | 2024年にはクラウドへの移行が加速しました。 |
| Kaldi (tri3) | [2→二][0→千][2→二][4→十][+四+]年にはクラ[ウ→ン]ドへの移行が加速しました |
| CRDNN–CTC (best) | 2024年にはクラウドへの移行が加速しました |
| Whisper (best) | 2024年にはクラウドへの移行が加速しました |

### 4.5.4 Impact of Preprocessing Choices on Results

Because all systems used the same cleaned and standardized dataset, differences in accuracy and speed mainly reflected model architecture and decoding strategy rather than evaluation handling. Nevertheless, the preprocessing steps in Chapter 3 (manual subtitle selection, cleaning, VAD chunking, and filtering) were designed to reduce transcript noise and non-speech audio, which supported stable training and fair comparison across model families (**latif2020**; **ando2021**).

A issue thet need preprocesing is that when processing for wer, it resulting in very high WER because japanese is consider as one word for each sentence

since it dows not have spaces. So the strategy is to use tokenization to tokenize the sentence into words first using python library first before calculate the WER (**TOKENIZATION_PLACEHOLDER**).

In addition, consistent handling of punctuation and normalization rules is important when comparing CER/WER across systems, because different model families may produce different conventions (e.g., punctuation insertion or numeric formatting). This observation reinforces the Chapter 1 motivation that post-processing and text formalization are part of the practical ASR pipeline, not only model decoding (**ando2021**).

### 4.5.5 Limitations

Key limitations include TEDx-only speaking style, possible subtitle-to-speech mismatch, sensitivity of Japanese WER to tokenization, and limited ablation studies (e.g., VAD thresholds, filtering thresholds, or language model variants).

The TEDx domain emphasizes relatively clear monologue speech, and therefore may not represent conversational Japanese or noisy real-world environments. In addition, subtitle references can still contain minor mismatches even when manually authored, which can place a ceiling on achievable CER/WER. Finally, the tokenization choice for Japanese WER can influence reported values, so consistent reporting and tool documentation are required for fair interpretation across studies (**TOKENIZATION_PLACEHOLDER**).

## 4.6 Summary

This chapter presented results for each stage of the Chapter 3 pipeline and compared ASR performance across classical and neural approaches. Kaldi improved substantially across staged training, CRDNN–CTC provided the fastest decoding, and fine-tuned Whisper variants achieved the best transcription accuracy with clear speed trade-offs by model size. The next chapter concludes the study and proposes future improvements.