CE-903 Group Project

System Requirements Specification

Automatic Speech Recognition (ASR) using Deep Learning

Team No: 7

Prepared by:

Hrithik Majumdar Shibu (2501488)
Aniket Domadiya (2501463)
Uswa Mahmood (2501194)
Aaditya Nair (2500803)
Soham Dharne (2500908)
Edward Hoogewerf (2501806)
Latha Mahendra (2508157)
Sayed Hossain Bhuiyan (2500958)
Tarak Hossain (2500810)

Project ID: 83794

Supervisor: Dr Delaram Jarchi

Course: CE-903 (Group Project)

Date: 6 February 2026

Table of Contents

ABSTRACT

Automatic Speech Recognition (ASR) using Deep Learning has emerged as a foundational technology for natural, human-like interaction between people and machines, enabling robust speech-to-text conversion across diverse domains. This project, “Automatic Speech Recognition (ASR) using Deep Learning,” aims to design and implement a modular, end-to-end research-grade ASR pipeline that ingests public speech datasets, preprocesses audio and text, trains state-of-the-art neural architectures, and delivers reproducible transcription and evaluation workflows. By leveraging contemporary models such as wav2vec 2.0, XLS-R, Conformer, and Whisper-small, the system targets accurate, scalable, and experimentally transparent recognition performance measured through standard metrics like Word Error Rate and Character Error Rate. As a primary implementation of testing the pretrained Whisper-small model on the Mozilla Common Voice Dataset, we have gained a Word Error Rate of 0.09, which means it transcribed 91% of words correctly. Developed within an agile, academic setting, the project emphasizes repeatable experimentation, clear documentation, and rigorous testing, providing a robust platform for exploring deep learning approaches to speech recognition and supporting a good-level research.

1. Introduction

Automatic Speech Recognition (ASR) is a core technology in modern human–computer interaction systems, enabling machines to convert spoken language into textual representations. Recent works in deep learning have greatly advanced the quality, robustness, and scalability of ASR systems, especially by using neural architectures (at various levels of processing) such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Transformers, and self-supervised representation learning methods. These developments have positioned deep learning–based ASR systems as essential components in applications such as virtual assistants, transcription services, accessibility tools, and large-scale speech analytics.

This paper provides the System Requirements Specification (SRS) for Automatic Speech Recognition (ASR) based on Deep Learning methodologies. The system is developed as an academic research and experimentation tool; its development emphasizes reproducibility, quantitative evaluation, and modular design. The SRS that includes the functional and non-functional requirements, system definition, and a high-level view of operation serves to direct design, development, testing, and validation activities.

1.1 Purpose

The purpose of this document is to formally specify the requirements and constraints of a deep learning–based Automatic Speech Recognition system developed as part of an academic group project. It is also a benchmark proposition for students, supervisors, and assessors: it states explicitly what the system should perform, how it should be performed, and when that performance occurs. Specifically, this document aims to:

- Define a concise list of functional and non-functional requirements for the ASR system.
- Establish a common basis of understanding and agreement on system goals, assumptions, and constraints.
- Support traceability of impact chains between requirements, design decisions, and evaluation results.
- Support methodical development, testing, and validation coordinated with academic performance evaluation and dissertation norms.

1.2 Scope

The proposed system tries to develop a complete end-to-end ASR pipeline that involves modern deep learning. The system is dedicated to converting speech recordings into standard texts based on several publicly accessible datasets of transcribed speech and neural network–based acoustic models.

The system consists of:

- Speech audio data ingestion and pre-processing (resampling, normalisation, augmentation, etc.).
- Supervised or self-supervised learning–based training and fine-tuning of deep learning ASR models.
- Text conversion: transcription of decoded speech to text.
- The quantitative evaluation based on Word Error Rate (WER) and Character Error Rate (CER) is carried out.
- Benchmark and Ablation on model topology, preprocessing methodologies, and decoding style.

The scope of the problem specifically excludes commercial deployment, real-time delay, and high value of sensitive or private speech data. The system is for research, experimentation, and academic demonstration only.

1.3 Overview

We begin with documenting the complete ASR system below in this paper. Section 2 explains the requirements analysis and specification process along with data assumptions, system architecture, use cases, and both functional and non-functional requirements. In Section 3, we present the test strategy and test cases, as well as the evaluation schedule used to verify software correctness and model performance. Section 4 describes project planning for the development methodology used, tools, computational requirements, and timeline.

Together, these sections ensure that the proposed ASR system is well-defined, systematically developed, and rigorously evaluated, providing a strong foundation for both academic assessment and dissertation-level research in deep learning–based speech recognition.

The remainder of this document includes: (2) requirements analysis and specification, (3) testing schedule, and (4) project planning, followed by references.

2. Requirements Analysis and Specification

2.1 Methodology for Elicitation, Analysis, and Specification

This section defines the methodology used for the elicitation, analysis, specification, and validation of system requirements for the Automatic Speech Recognition using Deep Learning (ASR) system.

2.1.1 Requirements Elicitation

System requirements shall be elicited from multiple authoritative sources to ensure completeness, stakeholder alignment, and feasibility. Primary sources include:
(i) formal group project objectives and academic assessment criteria, which define expected system capabilities and evaluation standards;
(ii) structured user stories derived from key stakeholders, including the system developer/researcher, the academic supervisor, and potential demonstration users; and
(iii) environmental and operational constraints imposed by the target execution platforms, such as cloud-based environments (e.g., Kaggle GPU P100 and Google Colab).

Elicitation activities shall employ document analysis, stakeholder goal decomposition, and scenario-based reasoning to capture both functional and non-functional requirements relevant to the ASR pipeline.

2.1.2 Requirements Analysis

All elicited requirements shall be analysed to ensure correctness, clarity, completeness, consistency, and feasibility. Analysis shall verify that requirements collectively cover the entire ASR workflow, including data ingestion, preprocessing, feature extraction, model training, inference, and performance evaluation.

Each requirement shall be evaluated against project constraints such as dataset availability, computational resources, time limitations, and reproducibility requirements. Conflicting or ambiguous requirements shall be resolved through refinement and stakeholder validation. Requirements shall be prioritised using the MoSCoW method (Must-Have, Should-Have, Could-Have) to support incremental development and risk-aware decision-making.

2.1.3 Requirements Specification

Approved requirements will be documented using unambiguous, verifiable, and testable language in accordance with IEEE SRS standards. Each requirement will be uniquely identified and expressed using “shall” statements to indicate mandatory system behaviour.

The specification process shall be iterative and aligned with sprint-based development, enabling continuous refinement based on experimental results and stakeholder feedback. Traceability shall be maintained between requirements, design components, implementation artifacts, and validation procedures to ensure full lifecycle coverage.

2.1.4 Requirements Validation and Verification

Each specified requirement shall be associated with one or more verification methods, including unit testing, integration testing, and quantitative performance evaluation. Model-related requirements shall be validated using established ASR metrics such as Word Error Rate (WER), Character Error Rate (CER), and inference latency. System-level requirements shall be validated through end-to-end pipeline testing and reproducibility checks across supported execution environments.

2.1.6 Risk and Constraint Considerations

Potential risks include limited GPU availability, variability in audio quality across datasets, and dependency incompatibilities within cloud environments. These risks shall be mitigated through early environment validation, modular pipeline design, and fallback execution strategies (e.g., CPU-based inference for testing). All constraints and assumptions shall be explicitly documented to preserve transparency and reproducibility.

2.2 Dataset and Data Assumptions

The project will use one or more openly available, transcribed speech datasets from the Speech-Datasets-for-ASR collection, focusing on English and/or another target language depending on project scope (Dataset). Example datasets include Mozilla Common Voice (multi-language; large-scale), LibreSpeech(English speech; large-scale), and VoxForge (community-recorded). Audio will be resampled to a consistent format (e.g., 16 kHz mono), and transcripts will be normalised (e.g., punctuation/number handling) using a documented text-cleaning policy.

Key data fields: audio file path, transcription text, optional speaker/meta fields, and split labels (train/validation/test).

Dataset Name	Description	Source/Links	Strengths
VoxForge	Community-driven and multilingual, supporting over a dozen languages	VoxForge	Consists of nearly 40 languages spoken
LibriSpeech	English-only read speech, but great for training and validation	LibriSpeech	Focuses on the English language for more than 100 hours of training

Mozilla Common Voice	Massive multilingual dataset containing over 33,000 hours of validated audio in over 130 languages	[Mozilla Common Voice]()	Clean and less noisy audio speeches

2.3 System Architecture

This section presents a high-level system architecture for the proposed speech recognition system. The architecture is intended to describe the major functional components and their interactions, while remaining independent of specific implementation choices. The layered structure supports modular development, experimentation, and evaluation.

The system is organised into four main layers: (A) Data Layer, (B) Model Training Layer, (C) Evaluation Layer, and (D) Inference Layer.

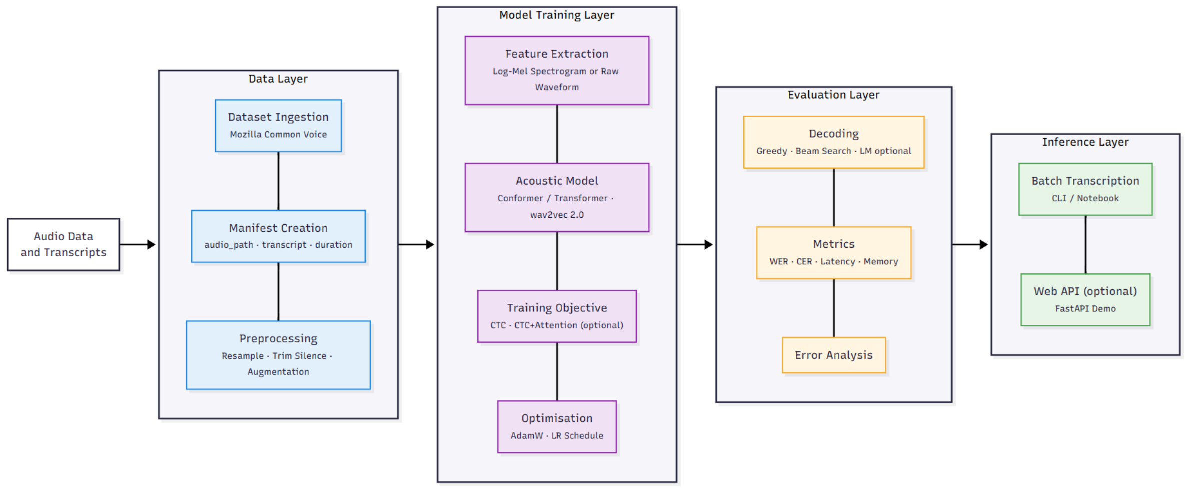

Figure 1: System Architecture

A. Data Layer
- Dataset download/ingestion (Common Voice/VoxForge)
- Manifest creation (audio_path, transcript, duration)
- Preprocessing (resample, trim silence, augmentation)

B. Model Training Layer
- Feature extraction (log-Mel spectrogram) OR raw waveform frontend
- Acoustic model (e.g., Conformer/Transformer, wav2vec 2.0 fine-tuning)
- Objective: CTC (and optionally CTC+Attention / Transducer)
- Optimisation (AdamW, LR schedule), checkpointing

C. Evaluation Layer
- Decoding (greedy / beam search + optional LM)
- Metrics: WER, CER, latency, memory usage
- Error analysis and reporting

D. Inference Layer
- Batch transcription CLI / notebook interface
- Optional simple web API (FastAPI) for demo

Data flows sequentially from the Data Layer to the Model Training Layer during training and from the Data Layer to the Inference Layer during deployment. Trained models produced by the Model Training Layer are consumed by both the Evaluation Layer and the Inference Layer. Evaluation outputs inform iterative refinement of data preprocessing, model configuration, and training strategies.

2.4 Model Architecture

This project will explore various state-of-the-art deep learning architectures for automatic speech recognition (ASR). These pre-trained self-supervised and hybrid models will be fine-tuned on labelled speech data.

This section will assess the suitability of wav2vec2, XLS-R, whisper-small, and conformer for an end-to-end ASR and will do a comparative analysis of ASR models.

2.4.1 Architectural Paradigms in Modern ASR

There are three conceptual concepts in modern ASR systems

1. Acoustic Frontend: It transforms raw audio into latent representations
2. Sequence Modelling Network: It captures temporal and contextual dependencies
3. Decoding Objective: It maps representations to textual output

In recent breakthroughs, self-supervised pretraining on raw waveforms has been introduced in order to improve the performance of low-resource models and handle multilingual scenarios, which is better than hand-crafted acoustic features. Some of the models that will be discussed in this project will follow these paradigms, differing in learning strategy, etc.

2.4.2 wav2vec 2.0 Architecture

Wav2vec 2.0 is a self-supervised framework for learning speech representations, which was introduced by Facebook AI Research. It learns contextualized speech representations directly from raw audio waveforms rather than explicitly extracting features such as MFCCs and log-Mel spectrograms.

Architecture Design:

There are three main components:

Feature Encoder: It is a multi-layer convolutional neural network that compresses the audio signals into latent speech representations. These are smaller and more meaningful snippets of speech.

Context Network: It is a transformer-based encoder that maps the relationship between different latent representations. It helps the model to learn patterns and understand the context of the entire recording.

Quantization Module (pretraining only): For a model to learn the structure of language without labeled data, it categorizes latent representations in a fixed dictionary of sounds. The model can learn to guess the correct sound from the dictionary based on the surrounding context by hiding certain parts of the audio.

For the fine-tuning phase, the quantization module is removed, and a linear layer is added on top of the context network to map the speech features directly to words. This layer is trained using CTC loss to make sure that the model aligns audio features with the transcript.

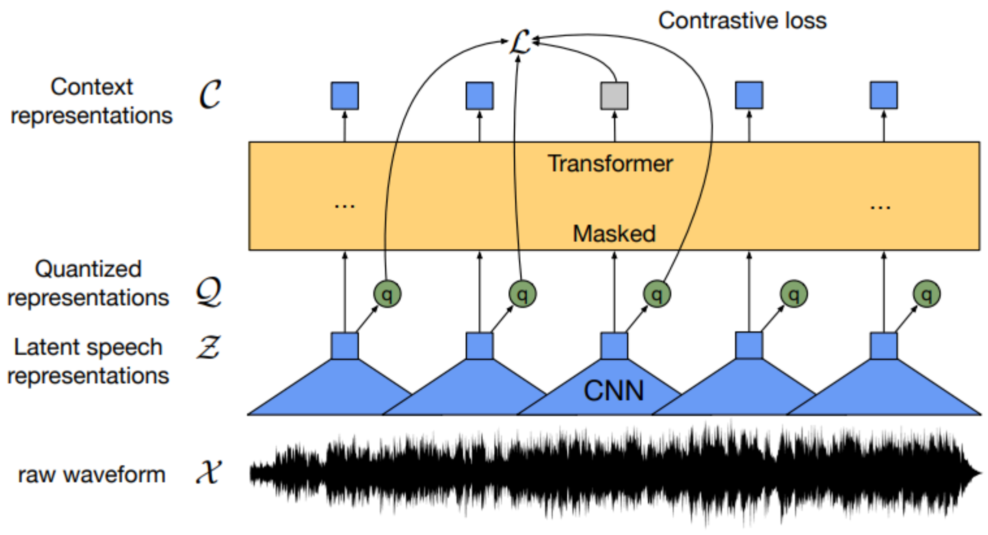

Figure 2: Architecture of the wav2vec 2.0 framework. ***Source:*** *Baevski et al. (2020).*

Strengths and Limitations

Wav2vec2 performs well in low-resource settings by learning from large amounts of unlabeled audio. But there is a major drawback, which is memory usage. The model consumes a lot of memory because it relies on transformer encoders during fine-tuning on longer audio files.

2.4.3 XLS-R (Cross-Lingual Speech Representations)

XLS-R builds on top of wav2vec2 and serves as a large-scale multilingual framework. It is pretrained on hundreds of thousands of hours of speech covering more than 100 languages.

Architecture design

It keeps the same architectural backbone as wav2vec2

- CNN-based feature encoder
- Transformer context network
- Contrastive self-supervised pretraining

The key difference here is the massive scale and linguistic diversity. It learns a universal phonetic representation by training on over 436,000 hours of speech across 128 languages.

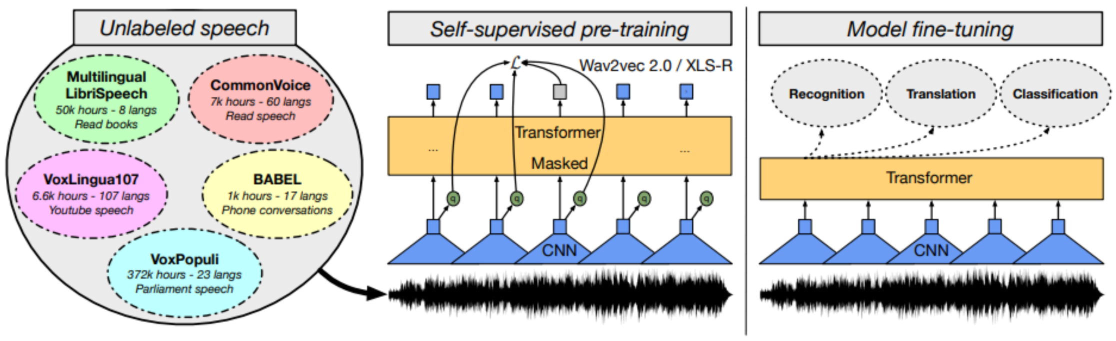

Figure 3: Self-supervised cross-lingual representation learning. ***Source:*** *Babu et al. (2022).*

Strengths and Limitations

XLS-R performs really well in multilingual and low-resource language scenarios, which makes it suitable for datasets like Common Voice. But larger variants of XLS-R(300M-2B parameters) introduce major computational overhead.

2.4.4 Conformer Architecture

Conformer architecture was proposed to address the limitations of pure transformer-based ASR models. A pure transformer struggles to capture local patterns that define individual speech sounds. This is a task where Convolutional layers excel.

Architecture Design

A Conformer encoder block consists of four sequential modules:

1. Feed-Forward Network (FFN)
2. Multi-Head Self-Attention (MHSA)
3. Convolution Module (Depthwise separable convolution)
4. Second Feed-Forward Network

This structure allows the model to capture both:

- Local acoustic patterns (phoneme-level features)

- Long-range temporal dependencies (word and sentence level context)

Conformer models generally utilize one of two decoding approaches:

- **CTC (Connectionist Temporal Classification):** It is preferred because of its simplicity and fast inference speeds.
- **Hybrid CTC + Attention:** Used when higher accuracy is the priority because it uses an attention mechanism to refine the CTC predictions.

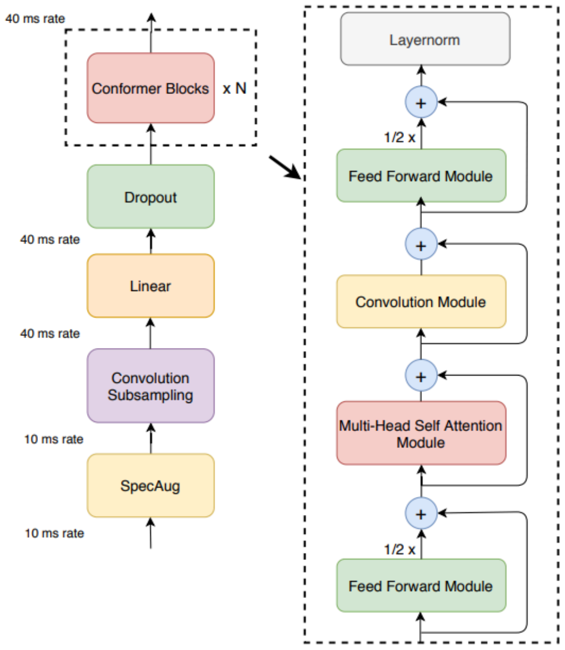

*Figure 4: Conformer encoder model architecture. Conformer consists of two feed-forward layers with half-step residual connections sandwiching the multi-headed self-attention and convolution modules, followed by post layernorm at the end. **Source:** Gulati et al. (2020).*

Strengths and Limitations

Conformers perform really well in noisy and conversational speech scenarios. It has achieved state-of-the-art accuracy across many ASR benchmarks. But one of the limitations is that it requires a lot of labelled data unless it is used with some pre-supervised model like wav2vec2.

2.4.5 Whisper-Small Architecture

Whisper is an encoder-decoder transformer-based ASR model that is trained in a supervised manner on large-scale multilingual speech text pairs. It is different from self-supervised models because it integrates acoustic and language modeling within a single framework. It removes the need for separate language modeling components. Whisper-small contains approximately 244 million parameters, and it offers a good trade-off between accuracy and computational cost.

Input Representation and Acoustic Frontend

Whispers does not work on raw audio waveforms; instead, it uses a log-Mel spectrogram frontend, which is created as follows:

- Audio is resampled to 16 kHz
- A log-Mel spectrogram with:
    - 80 Mel frequency bins
    - Fixed time window (30 seconds maximum)
- Spectrograms are normalized and padded/truncated to a fixed length

This design choice for whispers makes it less flexible than other waveform models, but it simplifies the training and convergence process.

Encoder Architecture

The encoder in whisper-small consists of 12 transformer layers. Its main goal is to process the input and transform it into high-level latent representations. The encoder captures complex relationships between parts of audio and provides them to the decoder.

Key characteristics of Whisper-small encoder:

- 12 Transformer encoder layers
- 384-dimensional hidden size
- 6 self-attention heads
- Standard Transformer components:
    - Multi-head self-attention
    - Position-wise feed-forward networks
    - Residual connections and layer normalization

The encoder identifies relationships between different words and produces deep acoustic embeddings that provide the decider with a detailed map of the speech.

Decoder Architecture

Whisper uses an autoregressive decoder that predicts the next text token based on the sequence of previous tokens and the encoder's hidden states. It uses cross-attention to focus on the relevant parts of the acoustic signal.

Decoder characteristics:

- 12 Transformer decoder layers
- Causal (masked) self-attention
- Cross-attention over encoder outputs
- Subword tokenization using a multilingual byte-pair encoding (BPE) vocabulary

At each decoding step, the model checks:

- Previously generated text tokens
- Full acoustic context from the encoder

CTC models assume that each audio frame is independent, whereas the whispers framework captures the joint probability of the entire text. Decoder predicts the next token based on the previous tokens and the audio, which makes the model function as its own internal language model.

Training Objective and Decoding Strategy

Whisper is trained using a cross-entropy sequence-to-sequence loss, rather than CTC. During inference Beam search decoding is used.

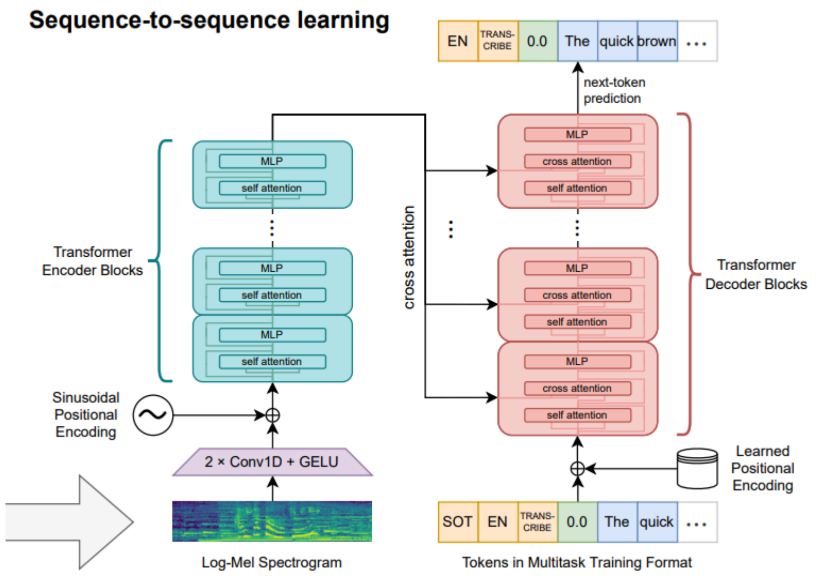
Sequence-to-sequence learning
EN
TRANS-CRIBE
0.0
The
quick
brown
...
next-token prediction
MLP
self attention
cross attention
Transformer Encoder Blocks
Transformer Decoder Blocks
Sinusoidal Positional Encoding
2 × Conv1D + GELU
Learned Positional Encoding
SOT
Log-Mel Spectrogram
Tokens in Multitask Training Format

Figure 5: Sequence-to-sequence learning approach. ***Source:*** *Radford et al. (2022).*

Strengths and Limitations

Whisper performs really well in noisy conditions and accents and has built-in multilingual capability. But its architecture is less flexible than wav2vec-style approaches because it relies on fixed-length 30-second segments, and it is more difficult to pair with external language models.

2.4.6 Comparative Analysis of ASR Architectures

Model	Pretraining Type	Input	Strengths	Limitations
wav2vec 2.0	Self-supervised	Raw waveform	Strong low-resource performance, flexible fine-tuning	High memory usage

Conformer	Supervised / Hybrid	Features or SSL embeddings	Excellent accuracy, robust to noise	Needs labelled data
XLS-R	Multilingual SSL	Raw waveform	Cross-lingual transfer, scalable	Large models are resource-heavy
Whisper	Supervised	Log-Mel	Strong zero-shot performance	Limited architectural control

2.4.7 Model Selection

We have tested Whisper-small on the Common Voice dataset (55,000 samples) to set a baseline, and it achieved a Word Error Rate (WER) of 0.09. which means it transcribed 91% of words correctly. This result makes it perfect to make a comparison against other models.

2.5 System Models (Use Cases)

Primary actors: Researcher/Developer (main), Supervisor (review), Demo User (optional).

Use Case	Actor	Outcome
UC-01 Upload/Select Audio	Researcher/Demo User	Audio available for transcription
UC-02 Run Batch Transcription	Researcher	Transcripts generated for a folder/list
UC-03 Train Model	Researcher	Model trained and checkpoints saved
UC-04 Evaluate Model	Researcher	WER/CER and reports produced
UC-05 Compare Experiments	Researcher	Ablation results logged and reproducible
UC-06 Export Results	Researcher/Supervisor	Tables/plots exported for projects

2.6 Functional Requirements (Enumerated)

1. **FR-01:** The system shall ingest a public ASR dataset (download or user-provided path) and build a manifest file (JSON/CSV) containing audio paths and transcripts.
2. **FR-02:** The system shall preprocess audio into a consistent format (sampling rate, channel count) and apply optional augmentation (noise, speed perturbation) configurable by the user.
3. **FR-03:** The system shall tokenise/normalise transcripts using a documented text-normalisation policy.
4. **FR-04:** The system shall support training at least one deep-learning ASR model (e.g., CTC-based Transformer/Conformer, or wav2vec 2.0 fine-tuning).
5. **FR-05:** The system shall save model checkpoints, training logs, and configuration files to enable reproducibility.

6. **FR-06:** The system shall decode model outputs to text using greedy decoding and optionally beam search with an external language model.
7. **FR-07:** The system shall evaluate the model on a held-out test set and compute WER and CER.
8. **FR-08:** The system shall provide error analysis outputs (examples of insertions/deletions/substitutions) and summary tables.
9. **FR-09:** The system shall provide an inference interface for single-audio and batch transcription (CLI or notebook).
10. **FR-10:** The system shall export results (metrics tables, plots) in standard formats suitable for dissertation writing.

2.7 Non-Functional Requirements (Enumerated)

- **NFR-01 Performance:** For the chosen dataset/model, the system shall achieve measurable improvements over a simple baseline; target WER to be defined per dataset split and documented.
- **NFR-02 Efficiency:** Training and inference shall be feasible on Kaggle GPU P100 (16 GB VRAM) using mixed precision where appropriate.
- **NFR-03 Reliability:** Training runs shall be resumable from checkpoints; failures should be detectable via logs.
- **NFR-04 Reproducibility:** Experiments shall be reproducible using fixed seeds, versioned configurations, and environment specifications.
- **NFR-05 Usability:** A clear runbook (README) shall enable a new user to reproduce training and evaluation.
- **NFR-06 Security & Privacy:** Only public datasets will be used; if any user audio is tested, it should not be stored beyond the experiment session.
- **NFR-07 Maintainability:** The code shall be modular (data, model, training, and evaluation) with docstrings and consistent formatting.

2.8 Detailed Requirements (Selected)

FR-04 – Train ASR model

Priority	Must
Inputs	Manifest (train/val), model config, hyperparameters
Process/Rules	Train a CTC-based ASR model (e.g., Conformer/Transformer) or fine-tune wav2vec 2.0 on the selected dataset; log loss and validation WER per epoch; save checkpoints.
Outputs	Saved model checkpoints; training logs; best model selected by validation WER/CER
Dependencies	PyTorch/Transformers/torchaudio; dataset storage; GPU if training
Acceptance Criteria	A training run completes end-to-end and produces checkpoints; validation metrics are

	reported and reproducible within a small variance across runs.

FR-07 – Evaluate with WER/CER

Priority	Must
Inputs	Test manifest, trained model, decoding configuration
Process/Rules	Decode test audio to text using greedy decoding; compute WER and CER; store per-utterance errors and summary statistics.
Outputs	Evaluation report (WER/CER), per-sample predictions, error examples
Dependencies	PyTorch/Transformers/torchaudio; dataset storage; GPU if training
Acceptance Criteria	The evaluation script produces WER/CER on the test split and saves a report file; the metrics match manual spot-checks on a small sample.

3. Testing Schedule

To ensure that the Automatic Speech Recognition (ASR) system using Deep Learning operates reliably and satisfies both functional and non-functional requirements, a structured testing and validation schedule is adopted. ASR systems combine conventional software components with probabilistic machine learning models, making rigorous testing essential throughout the development lifecycle. Testing activities are therefore carried out continuously, with more extensive validation performed during the later stages of the project.

3.1 Testing Approach

To ensure the system performs at its peak, testing and validation will be conducted during the project's final phase. All team members will work together to carry out an excessive number of verification and validation processes throughout the product development process. By completing these checks, the stakeholder and consumer expectations will be met. The project supervisors and principal collaborators, Dr. Delaram Jarchi (Project Supervisor) and Dr. Muge Sayit (Course Supervisor and Materials Acquisition), who will serve as the project's head collaborators, will oversee these tests. Meetings and agendas will be discussed with these key individuals.

The testing approach integrates unit testing, integration testing, and system-level evaluation. This multi-layered strategy ensures early detection of software faults while enabling objective assessment of model performance.

At the unit level, testing focuses on individual components of the ASR pipeline, including dataset loading, audio decoding, resampling, feature extraction, text normalisation, and tokenizer behaviour. These tests verify that raw speech data is consistently converted into the required standard format before being processed by the deep learning model. This is particularly important in ASR systems, as preprocessing inconsistencies can significantly degrade recognition accuracy without causing explicit runtime failures.

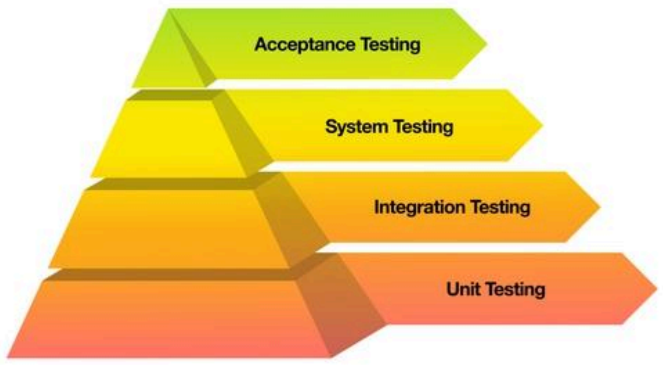

Figure 6: Software Testing Pyramid showing the hierarchy of different testing levels
Source: *Tetteh et al. (2024).*

Integration testing is conducted after major modifications to the data pipeline or model architecture. During this phase, short end-to-end training runs are executed on reduced subsets of the dataset to validate the interaction between data preprocessing, neural network training, optimisation, and checkpointing mechanisms. Successful integration testing confirms that the system can complete training cycles without errors.

Model acceptance is based on quantitative evaluation metrics widely used in ASR research, namely Word Error Rate (WER) and Character Error Rate (CER). These metrics are computed on held-out validation and test datasets to provide unbiased estimates of recognition accuracy. In addition to accuracy, inference latency and GPU memory utilisation are monitored to ensure that the system remains computationally efficient and stable.

3.2 ASR System Test Cases (Examples)

Test ID	Requirement	Input	Expected Output
TC-01	FR-01	Dataset path or dataset download flag	Dataset manifest generated with valid audio paths and aligned transcripts
TC-02	FR-02	Single WAV file (44.1 kHz, stereo)	Audio resampled to 16 kHz mono; duration preserved; no runtime errors
TC-03	FR-04	Reduced dataset (≈ 1 hour) with training configuration	Training completes one epoch; checkpoint saved; training loss decreases
TC-04	FR-07	Trained checkpoint with test dataset	WER and CER computed; evaluation report saved; per-utterance predictions generated
TC-05	NFR-02	Batch inference on 100 utterances	Inference latency recorded; GPU memory usage within limits; no OOM errors

3.3 Testing Timeline

Stage	When	Notes
Unit testing	Weekly (each sprint)	Validation of preprocessing utilities and helper functions

Integration testing	After major pipeline changes	Short end-to-end training and evaluation runs
System testing	Mid-project and final weeks	Full training, inference, evaluation, and reporting
Acceptance testing	Final two weeks	Final metric verification and requirement confirmation

3.4 Agile Sprint Timeline (12-Week Project Example)

Testing Type	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
Unit Testing	✔	✔	✔	✔	✔	✔
Integration Testing		✔		✔		✔
System Testing			✔			✔
Acceptance Testing					✔	✔

4. Project Planning

4.1 Project Management Structure and Methodology

A project management methodology provides a structured set of guidelines and processes aimed at achieving maximum project efficiency and effectiveness. Selecting the most suitable approach for a team requires careful consideration of the project's objectives and constraints. Accordingly, various development methodologies were evaluated to identify the one that best aligns with the proposed project plan. Key factors influencing this selection include cost and budget limitations, team size, ability to manage risk, flexibility in adapting to change, and the need for timely delivery. The project is managed using an iterative, research-driven development methodology, inspired by Agile practices and adapted for experimental deep learning research. Work is organised into short research sprints lasting one to two weeks, with each sprint producing a tangible outcome such as a preprocessing pipeline, baseline ASR model, improved architecture, or evaluation report.

Regular supervisory meetings are conducted to review progress, assess experimental findings, and refine project objectives. This flexible management structure supports rapid experimentation while maintaining the methodological rigor expected.

4.1.1 Agile Methodology

Agile project management is a flexible and collaborative approach that enables teams to self-organise and adapt throughout the project lifecycle. Planning and task execution are iterative and evolutionary in nature, with a strong emphasis on early delivery and continuous improvement. Changes are welcomed whenever they contribute to better outcomes or enhanced processes. Unlike the traditional waterfall model, which follows a rigid and sequential structure, agile methodologies are designed to be fast, responsive, and adaptable. This approach emerged in response to growing dissatisfaction with linear project management methods, particularly in environments where requirements frequently evolve.

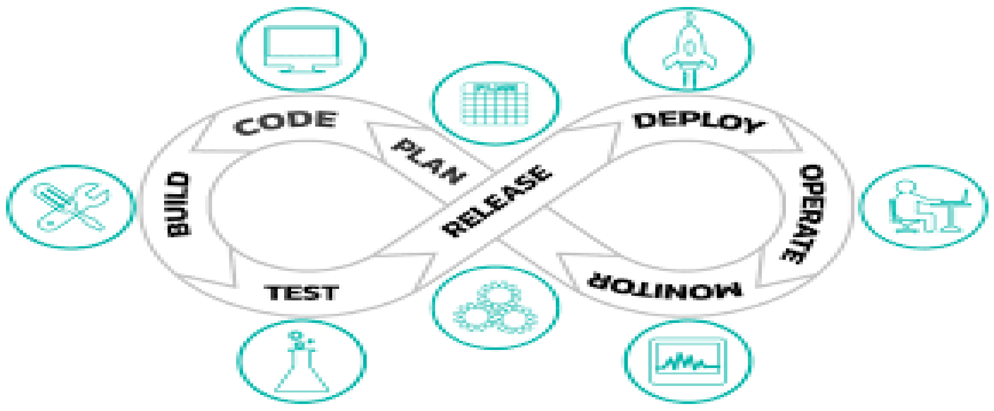

Figure 7: Continuous software development lifecycle as applied in agile methodology.
***Source:** Abrahamsson et. al. (2017)*

The core principles of agile project management include collaboration, rapid development cycles, and openness to data-driven decision-making. Agile is especially suitable for projects where the solution is not fully defined at the outset and may change as new information becomes available. Throughout the project, teams repeatedly engage in planning, execution, and evaluation phases, allowing continuous demonstration of progress and refinement of outcomes.

4.1.2 Scrum

Scrum is an agile software development framework that follows an incremental and iterative approach to project delivery. It is designed to be lightweight, flexible, and efficient, enabling teams to deliver value to stakeholders consistently during the development process. In Scrum, work is organised into short, time-boxed iterations known as *sprints*, which typically last between one and two weeks. During each sprint, tasks are selected from a prioritised backlog and completed collaboratively by the team.

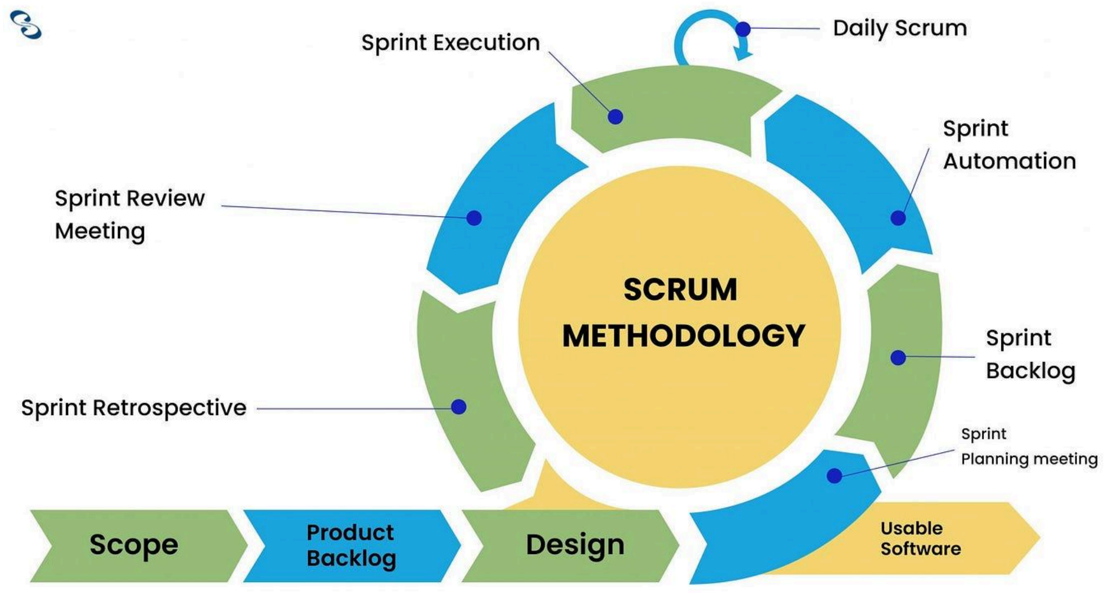

Figure 8: Scrum methodology cycle. ***Source:*** *Abrahamsson et. al. (2017)*

The primary goal of Scrum is to meet customer requirements by promoting transparent communication, shared responsibility, and ongoing improvement. Regular feedback and review cycles ensure that progress remains aligned with stakeholder expectations. However, Scrum is most effective when there is full commitment and active participation from all team members; without this level of engagement, the methodology may not be suitable for successful project execution.

4.2 Tools and Platforms

Category	Tools/Choices
Programming	Python 3.10+, Jupyter Notebooks, Git
Deep Learning	PyTorch, torchaudio, Hugging Face Transformers (optional)
ASR Modeling	CTC-based Transformer/Conformer; wav2vec 2.0 fine-tuning (optional)
Experiment Tracking	Weights & Biases or TensorBoard; CSV logs
Compute	Kaggle P100 GPU (16GB) and/or local GPU
Packaging	Conda environment or requirements.txt
Tools for collaboration and teamwork	Gitlab, Jira, Tuleap, Github

4.3 System Requirements (Research Environment)

Component	Minimum / Recommended
GPU	Minimum: NVIDIA GPU with >= 8GB VRAM; Recommended: Kaggle P100 (16GB) or similar for training
CPU	4 cores minimum; 8+ cores recommended for data preprocessing
RAM	16GB minimum; 32GB recommended for large dataset preprocessing
Storage	50GB+ free for small experiments; 200GB+ for large-scale Common Voice subsets and checkpoints
OS	Linux (Kaggle/Ubuntu) recommended; Windows/macOS supported with WSL/conda
Software	PyTorch, torchaudio, transformers (optional), librosa, jiwer (WER), numpy/pandas; ffmpeg for audio conversion

4.4 Project Plan (Gantt-style Table)

The project's layout is depicted in the Gantt chart below. The tasks that still need to be done are in ash color, while the green sections are used to indicate the completion state of each task. The Gantt chart will be expanded as the project progresses. The Gantt Chart can be seen well here as an indicative 10-week plan:

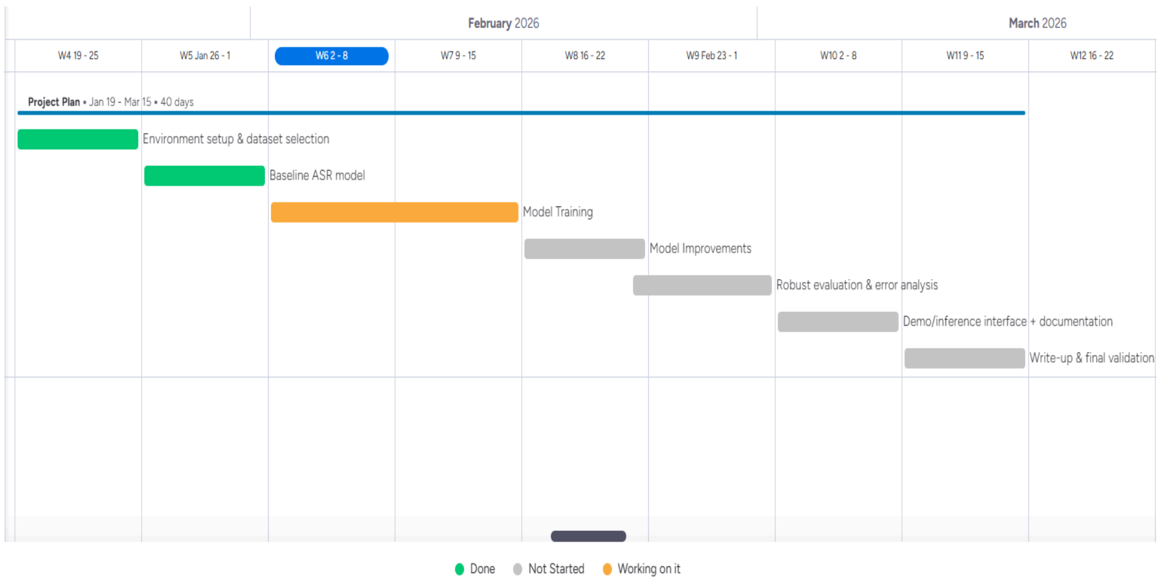

Figure 9: Gantt chart representing the project plan

The progress we managed to achieve so far is mentioned in the green-marked section in the above Gantt chart:

- We met our supervisor and chose the project topic.
- Researched the project and existing work.
- The product's conceptual prototypes are complete.
- A requirement specification has been created and approved.

5. References

- Rumeysa Keskin. Speech-Datasets-for-ASR (GitHub repository). https://github.com/Rumeysakeskin/Speech-Datasets-for-ASR (accessed 2026-01-26).
- A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. 'Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks.' ICML 2006.
- A. Baevski, H. Zhou, A. Mohamed, and M. Auli. 'wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations.' NeurIPS 2020.
- D. Amodei et al. 'Deep Speech 2: End-to-End Speech Recognition in English and Mandarin.' ICML 2016 / arXiv:1512.02595.
- A. Radford et al. 'Robust Speech Recognition via Large-Scale Weak Supervision' (Whisper). arXiv:2212.04356.
- Mozilla Common Voice. https://commonvoice.mozilla.org/
- VoxForge Speech Corpus. http://www.repository.voxforge1.org/
- V. Panayotov, G. Chen, D. Povey and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 2015, pp. 5206-5210, doi: 10.1109/ICASSP.2015.7178964. keywords: {Resource description framework; Genomics; Bioinformatics; Blogs; Information services; Electronic publishing; Speech Recognition; Corpus;LibriVox}.
- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile Software Development Methods: Review and Analysis. *ArXiv*. https://arxiv.org/abs/1709.08439
- Baevski et al., *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*, NeurIPS 2020 https://arxiv.org/pdf/2006.11477
- Gulati et al., Conformer: Convolution-augmented Transformer for Speech Recognition, Interspeech 2020 https://arxiv.org/pdf/2005.08100
- Babu et al., *XLS-R: Self-supervised Cross-lingual Speech Representation Learning*, ACL 2022 https://arxiv.org/pdf/2111.09296
- Radford et al., *Robust Speech Recognition via Large-Scale Weak Supervision*, 2022 https://arxiv.org/pdf/2212.04356
- Tetteh, Samuel Gbli. (2024). Software Testing Techniques and Levels in Software Development. International Journal of Advancements in Computing Technology. 2. 10-19. 10.56472/25838628/IJACT-V2I1P102.