SurShastra - Music Transcription Software

Submitted in partial fulfillment of the requirements
for the degree of

B.E. Computer Engineering

By

Geethu Joseph	**34**	**212123**
Yogiraj Buddhiwant	**39**	**212007**
Sonal Shinde	**46**	**212106**
Aditi Bhoir	**60**	**202016**

Guide
Dr. Nazneen Ansari
Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)
An Autonomous Institute, Affiliated to University of Mumbai

2024-2025

SFIT LIBRARY 2025

CERTIFICATE

This is to certify that the project entitled **"SurShastra - Music Transcription Software"** is a bonafide work of **"Geethu Joseph (34), Yogiraj Buddhiwant (39), Sonal Shinde (46) and Aditi Bhoir (60)"** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of B.E. in Computer Engineering.

Dr. Nazneen Ansari

Guide

(Dr. Kavita Sonawane)

Head Of Department

(Dr. Sincy George)

Principal

SFIT LIBRARY 2025

Project Approval Report for B.E.

This project report entitled *"SurShastra - Music Transcription Software"* by ***Geethu Joseph (34), Yogiraj Buddhiwant (39), Sonal Shinde (46) and Aditi Bhoir (60)*** is approved for the degree of ***B.E. in Computer Engineering***.

Examiners

1. Dr. Ashfaq Shaikh

2. Dr. Nasreen Ansari

Date: 30/04/2025

Place: Mumbai

SFIT LIBRARY 2025

Acknowledgements

We are thankful to a number of individuals who have contributed towards our final year project and without their help, it would not have been possible. First of all, we would like to express our gratitude to Dr. Nazneen Ansari, our project guide, for her prompt suggestions, guidance and encouragement over the course of our entire project.

We sincerely appreciate our project coordinators, for their tremendous assistance to our project. We also appreciate the support of the faculty in our department.

We express our sincere gratitude to our respected Director Bro. Shantilal Kujur, our Principal Dr. Sincy George and our Head of Department (CMPN) Dr. Kavita Sonawane for providing the facilities, encouragement as well as a conducive environment for learning.

Lastly, we would like to thank our parents and friends who played a crucial role keeping us motivated throughout with their constant nurture and support.

Sincerely,

Geethu Joseph

Yogiraj Buddhiwant

Sonal Shinde

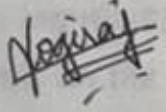
Aditi Bhoir

SFIT LIBRARY 2025

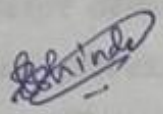
Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

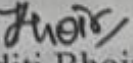
Geethu Joseph (34)



Yogiraj Buddhiwant (39)



Sonal Shinde (46)



Aditi Bhoir (60)

Date: 30/04/2025

SFIT LIBRARY 2025

Abstract

SurShastra is an automatic music transcription (AMT) system designed to convert polyphonic audio recordings into symbolic musical notation, such as MIDI files. While AMT has seen significant progress in Western music, it continues to face challenges in accurately transcribing non-Western forms, especially Indian classical music, which is rich in microtonal variations, complex rhythms, and improvisational elements. This project addresses these challenges by combining advanced deep learning methods—specifically Convolutional Neural Networks (CNNs) and Bidirectional Gated Recurrent Units (Bi-GRUs)—to model both spatial and temporal structures of musical audio.

The system processes input audio into log-mel spectrograms, which are then used to predict onset, offset, velocity, and frame-wise note presence. Unlike conventional frame-based models, SurShastra incorporates onset and offset regression to estimate the precise timing of musical notes, improving transcription resolution beyond frame-hop limits. The model is trained on the MAESTRO dataset ensuring robustness across genres.

Evaluation using precision, recall, and F1 score demonstrates the model's ability to accurately transcribe complex polyphonic pieces. Preliminary results show promising performance in capturing nuances like ornamentation and expressive timing, especially in Indian classical music. This system aims to bridge the gap between musical audio and symbolic notation, enhancing accessibility, analysis, and learning in both Western and non-Western musical traditions.

Keywords: *automatic music transcription, deep learning, CNN, Bi-GRU, onset-offset regression, polyphonic music, Indian classical music*

SIT LIBRARY 2025

Contents

1 Introduction **1**

1.1 Introduction 1

1.2 Background study Terminologies/ Definitions of new terms 2

1.3 Fundamental study points of the selected topic and the domain . . 3

1.4 Identification of challenges in the selected topic 4

1.5 Problem Statement and Proposed Solution 4

1.6 Scope of the system 5

2 Literature Survey **6**

2.1 Survey of Existing system 6

2.2 Limitations of Existing systems or research gaps 8

2.3 Motivation 9

3 Proposed System: Analysis **10**

3.1 Detailed explanation of Proposed system 10

3.1.1 Block diagram of Proposed system/ Workflow 10

3.1.2 Working Principle 11

3.1.3 Phase / Module - wise explanation 11

3.2 System Analysis 14

3.2.1 Functional Requirements 14

3.2.2 Non Functional Requirements 14

3.2.3 Software and Hardware requirements 15

3.2.4 Use Case Modeling 17

3.3 Proposed System: Analysis, Modelling and Design 19

SFIT LIBRARY 2025

3.3.1 Activity Diagram 19
3.3.2 Data Flow Diagram 20
3.3.3 Architectural View 22
3.3.4 Algorithms / Methodology 23
3.3.5 UI/UX design 26

4 Implementation Plan and Experimental Set up of the Proposed system 28
4.1 Experimental Set up 28
4.1.1 Dataset Details 28
4.1.2 Performance Evaluation Parameters 29
4.1.3 Special Requirement 30
4.2 Code 30

5 Results and Discussions 38
5.1 Presentation and validation of the results for the proposed system . 38
5.1.1 Quantitative and Qualitative results 38
5.1.2 Instrument-wise Performance Analysis 39
5.1.3 Evaluation Through Piano Roll Representation 40
5.2 Comparative Analysis with existing systems 41

6 Conclusion 43

References 44

SFIT LIBRARY 2025

List of Figures

3.1 Workflow diagram 10
3.2 Use Case diagram 17
3.3 Activity diagram 19
3.4 Data Flow Diagram Level-0 20
3.5 Data Flow Diagram Level-1 20
3.6 Data Flow Diagram Level-2 21
3.7 Architecture Diagram 22
3.8 SurShastra Home Page 26
3.9 Input section for uploading audio file 26
3.10 Download options after transcription 27

5.1 Piano roll for original midi file 40
5.2 Piano roll for transcribed midi file 40

SFIT LIBRARY 2025

List of Tables

3.1 Upload Audio Use Case 18
3.2 Transcribe Music Use Case 19

5.1 Instrument-wise performance of the SurShastra model 39
5.2 Comparative analysis of SurShastra with existing AMT systems . 41

SFIT LIBRARY 2025

Chapter 1

Introduction

1.1 Introduction

Music transcription is the process of converting audio recordings into symbolic musical notation, such as sheet music or MIDI files[1]. While this process is essential in music education, archiving, and analysis, it remains a labor-intensive task requiring significant expertise. The field of Automatic Music Transcription (AMT) aims to automate this process, enhancing accessibility for musicians, educators, and researchers.

Despite considerable progress in AMT, especially for Western classical piano music, existing systems struggle to handle the complexities of polyphonic and expressive musical content. Challenges such as overlapping harmonics, inaccurate onset/offset detection, and lack of dynamics modeling limit the effectiveness of traditional transcription methods. Furthermore, non-Western music, including Indian classical music, introduces unique difficulties due to ornamentations, microtonality, and improvisation.

Recent developments in deep learning, particularly the use of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown promise in overcoming these barriers[9]. Our project, SurShastra, proposes a deep learning-based AMT system that combines CNNs with Bidirectional Gated Recurrent Units (Bi-GRUs) to capture both spatial and temporal characteristics of musical audio. The system processes log-mel spectrograms extracted from raw audio and performs multi-headed prediction to estimate note onsets,

SET LIBRARY 2025

offsets, velocity, and frame-level activity.

To improve transcription resolution, we integrate a regression-based approach for predicting precise onset and offset timings, enabling high-resolution temporal detection. We also include velocity estimation to capture musical expressiveness, and our model is trained on the MAESTRO dataset to support a wide range of musical styles.

This project presents a robust, extensible AMT system designed to address the limitations of existing approaches by accurately transcribing polyphonic music, even in complex and non-Western settings. By leveraging advanced deep learning architectures and high-quality datasets, SurShastra aims to bridge the gap between musical performance and symbolic representation with precision and expressiveness.

1.2 Background study Terminologies/ Definitions of new terms

Onset

In music, the onset refers to the moment when a note or sound begins. It is a critical parameter in music transcription and audio analysis, as detecting the precise onset of a note helps in accurately transcribing musical performances. Accurate onset detection is essential for capturing the rhythm and timing of a musical piece.

Offset

The offset is the point in time when a note or sound ends. In music transcription, identifying the offset helps determine the duration of each note. It is used in conjunction with the onset to calculate the length of musical events and to accurately represent the timing and rhythmic structure of a piece.

Spectrogram

A spectrogram is a visual representation of the frequency content of an audio signal over time. It displays how the amplitude of different frequencies varies with time, using a color-coded graph where the x-axis represents time,

SFIT LIBRARY 2025

the y-axis represents frequency, and color intensity represents amplitude. Spectrograms are widely used in music information retrieval and audio analysis to analyze and interpret the spectral characteristics of audio signals.

Harmony

Harmony refers to the combination of different musical notes played or sung simultaneously to create a pleasing sound or to produce a specific tonal effect. It involves the study of how chords and chord progressions interact with each other to form a cohesive and aesthetically pleasing musical texture. Harmony adds depth and richness to music by creating a sense of support and context for the melody.

1.3 Fundamental study points of the selected topic and the domain

Various methods for Automatic Music Transcription (AMT) are as follows: [13]

- **Frame-level Transcription:** This method identifies notes within a specified time frame and is also known as Multi-Pitch Estimation (MPE). It is the simplest approach, and most Automatic Music Transcription (AMT) models are based on it. However, since a note might not be fully captured within a single time frame, the output can be inaccurate. Frame-level transcription can be implemented using spectrogram factorization or neural networks.
- **Note-level Transcription:** Note tracking goes beyond MPE by typically involving post-processing of MPE outputs. It relies on accurately detecting note onsets and durations. While staccato notes, which have clear separations between them, are easily detected, note-level transcription struggles with legato passages where notes appear to blend together.
- **Stream-level Transcription:** Also known as Multi-Pitch Streaming (MPS), this method focuses on tracking a specific stream of notes, effectively iso-

SFIT LIBRARY 2025

lating the instrument source. This technique is also referred to as instrument tracking or timbre tracking.

- **Notation-level Transcription:** This approach takes audio as input and produces a music sheet as output, making it the most challenging of the four methods.

Music Transcription is generally carried out using one of the two techniques: spectrum decomposition and neural networks. The output of a music transcription system may be one of the three: pitch frequency, or MIDI frequency, or a MIDI file itself or (in our case) piano roll sheet score.

1.4 Identification of challenges in the selected topic

Automatic Music Transcription (AMT) aims to convert audio recordings into symbolic musical notation, but it faces several longstanding challenges that hinder accuracy and real-world applicability. First, polyphonic transcription—identifying multiple overlapping notes—is difficult due to the complex structure of musical harmony. Second, accurate detection of note onsets and offsets is essential but challenging, especially when notes are played with expressive timing or sustain. Third, capturing the dynamics and velocity of notes is often overlooked, resulting in transcriptions that fail to reflect the emotion and nuance of the performance. Finally, robustness to variations in recording environments, background noise, and audio quality poses a major issue, especially when models are trained on limited or idealized datasets. These challenges highlight the need for deep learning approaches that can model time-series data effectively, adapt to real-world audio, and generate high-fidelity symbolic representations.

1.5 Problem Statement and Proposed Solution

In the music industry, musicians and music professionals encounter substantial difficulties when attempting to analyze and transcribe the musical notes of

SETLIBRARY 2025

songs accurately. This presents a significant obstacle to the effective study and interpretation of musical compositions. To address this issue, our proposed project aims to develop an automatic music transcription system, SurShastra, that takes a polyphonic audio recording as input and accurately predicts musical notes using advanced deep learning techniques. Unlike traditional approaches, our solution leverages a combination of Convolutional Neural Networks (CNNs) and Bidirectional Gated Recurrent Units (Bi-GRUs) to model both spectral and temporal characteristics of music. The model processes log-mel spectrograms to detect note onsets, offsets, frame-wise activity, and note velocity, enabling a high-resolution and expressive transcription. By training on the MAESTRO dataset, the system ensures adaptability to various musical styles. SurShastra provides users with a detailed symbolic representation of the music in MIDI format, offering deeper insights into a song's composition. Whether for music enthusiasts, musicians, or educators, this tool aims to revolutionize how we transcribe, study, and engage with music.

1.6 Scope of the system

The scope of the automatic music transcription system is to accurately transcribe music from various instruments, across both Indian and Western musical styles. The system will process audio inputs, extract relevant features, and generate corresponding musical notations in standard formats like MIDI or sheet music. It will handle the transcription of monophonic and polyphonic audio, recognizing multiple notes played simultaneously. The system will support diverse genres, ensuring it captures nuances specific to Indian classical music, such as microtonal variations, as well as Western music's harmonic structures. Additionally, it will be designed to adapt to different instrument timbres and complexities, providing a versatile tool for musicians, educators, and researchers. The final output will be user-friendly, allowing easy integration with digital audio workstations (DAWs) and music editing software.

SFIT LIBRARY 2025

Chapter 2

Literature Survey

2.1 Survey of Existing system

Recent developments in automatic music transcription (AMT) have shifted from traditional signal processing methods—such as the Fast Fourier Transform (FFT), Short-Time Fourier Transform (STFT), and Constant-Q Transform (CQT)—to machine learning techniques, particularly deep learning models like convolutional and recurrent neural networks. These traditional techniques were used to extract spectral features from audio signals, but often struggled with polyphonic complexity and variability across musical styles. In contrast, deep learning models have shown significant improvements in extracting musical elements such as pitch, rhythm, and note onsets directly from audio data, leading to more accurate and efficient transcription across various musical genres [2].

Automatic Music Transcription (AMT) is a key area within Music Information Retrieval (MIR), focusing on converting audio signals into musical notation. Researchers have explored various approaches, with traditional methods relying on algorithmic and signal processing techniques. Early pitch estimation, for example, used Linear Predictive Coding (LPC) to detect pitch in vocal signals via spectral analysis and window functions [3].

Paper [4] describes an experiment aimed at identifying notes and durations through digital signal processing, using methods like Fourier Transform and adaptive thresholding for peak detection, followed by Discrete Fourier Transform for mapping frequencies to notation. A significant advancement in AMT

SET LIBRARY 2025

came with the Constant-Q Transform (CQT), which offers better resolution at low frequencies, aiding in musical note identification based on frequency and duration [5].

The paper [6] investigates deep learning approaches for automatic music transcription, aiming to develop a model that can convert audio files into symbolic representations like music scores. Utilizing the MAPS database, which includes .wav files of polyphonic piano music and corresponding ground-truth data, the study employed a Constant Q transform (CQT) to convert audio into spectrograms. These spectrograms were split into 100-millisecond frames, each labeled with an 88-element vector indicating the presence or absence of piano notes. Three models were evaluated: a multi-label logistic regression model, a deep neural network (DNN), and a long short-term memory (LSTM) network.The multi-label logistic regression model, which lacked hidden layers, achieved the lowest performance with an F-measure of 0.559, recall of 0.422, and accuracy of 0.388. The deep neural network (DNN) model performed best with three hidden layers and a 10% dropout rate, reaching an F-measure of 0.776, recall of 0.716, and accuracy of 0.634. The long short-term memory (LSTM) model, optimized with three hidden layers and a 25% dropout rate, showed slightly lower scores: an F-measure of 0.756, recall of 0.711, and accuracy of 0.608. Overall, the DNN model outperformed the LSTM, which suffered from overfitting despite dropout adjustments.

The research paper [7] introduces a Bidirectional Long Short Term Memory (BLSTM) model for Automatic Music Transcription (AMT) that processes raw audio directly, differing from approaches that rely on spectrograms. The model takes 1024 raw audio samples (64 milliseconds) as input and predicts 131 values per timestep, representing the velocities of 128 piano keys and three pedals. It features two hidden layers with dropout rates of 0.2 to prevent overfitting and uses mean squared error as the loss function. Trained on 962 pieces from the MAESTRO dataset and tested on 177 pieces, the model achieved an F1-measure of 0.71 for onset detection, though it faced challenges with pitch

SIT LIBRARY 2025

accuracy and note offsets, with F1-measures of 0.63 and 0.26, respectively. Despite performing better than some spectrogram-based methods, it lags behind more recent state-of-the-art approaches. Future work will address pedal handling and include velocity values in evaluation metrics.

A high-resolution piano transcription approach [8] was developed using a regression-based CRNN model to predict continuous onset and offset times, including pedal and velocity information. This method leverages analytical inference to achieve fine temporal precision and demonstrated strong performance on the MAESTRO dataset, achieving a note onset F1 score of 0.90. It proved robust to label misalignment and was among the first to evaluate pedal transcription. However, the approach still depends on thresholding, struggles with advanced pedal techniques like half-pedals, and demands substantial computational resources for training.

While traditional signal processing techniques have made significant contributions to automatic music transcription (AMT), recent advancements in deep learning have opened new avenues for research, offering improved accuracy and efficiency in transcribing complex musical compositions. Addressing challenges such as limited datasets and evaluation metrics will be essential for the continued development of robust AMT systems that can be applied across various musical genres and contexts. As these technologies evolve, the potential for real-time transcription and interactive music applications becomes increasingly feasible, paving the way for innovative tools that enhance music education, analysis, and appreciation.

2.2 Limitations of Existing systems or research gaps

Automatic Music Transcription (AMT) faces several limitations due to challenges with datasets and model capabilities. A major issue is the lack of diverse and accurately annotated datasets, which limits the ability of AMT systems to generalize across various musical styles, instruments, and cultural contexts. Existing datasets often do not cover the full spectrum of music, resulting in models that perform well only on specific types of music but struggle with oth-

SIT LIBRARY 2025

ers. Additionally, the complexity of polyphonic music, where multiple notes are played simultaneously, poses significant difficulties. Overlapping sounds create ambiguities that current techniques, including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, often struggle to resolve, especially in the presence of multiple instruments.

Models trained primarily on Western music may struggle with non-Western music or experimental compositions that use unconventional scales, rhythms, or structures, highlighting the need for more inclusive training data and adaptable algorithms. Achieving complete music transcription involves multiple subtasks, including multi-pitch detection, rhythm quantization, and instrument identification. Existing models often fall short in addressing these tasks comprehensively, especially in complex polyphonic scenarios.

2.3 Motivation

Many times when trying to compose songs, musicians often face the problem of not being able to figure out the accurate notes and chords of songs. Creating an application that offers a platform for musicians to analyze and understand the musical elements of existing songs, can inspire them in their own creative process. The melodies that come to mind can be transcribed easily if a note analyser existed, streamlining the composition process. This understanding is vital for conveying the composer's intentions effectively. Notes are the basic building blocks of music. Understanding them is fundamental to grasping essential music theory concepts, such as scales, intervals, and chords. For students and aspiring musicians, understanding notes is fundamental to learning and mastering an instrument or voice. It forms the basis for music theory, sight-reading and overall musical knowledge. Instrumentalists can be highly benefited with this application as they can interpret the compositions they play more deeply when they understand the notes. This understanding enhances their appreciation of the musical structure and intent.

SET LIBRARY 2025

Chapter 3

Proposed System: Analysis

3.1 Detailed explanation of Proposed system

3.1.1 Block diagram of Proposed system/ Workflow

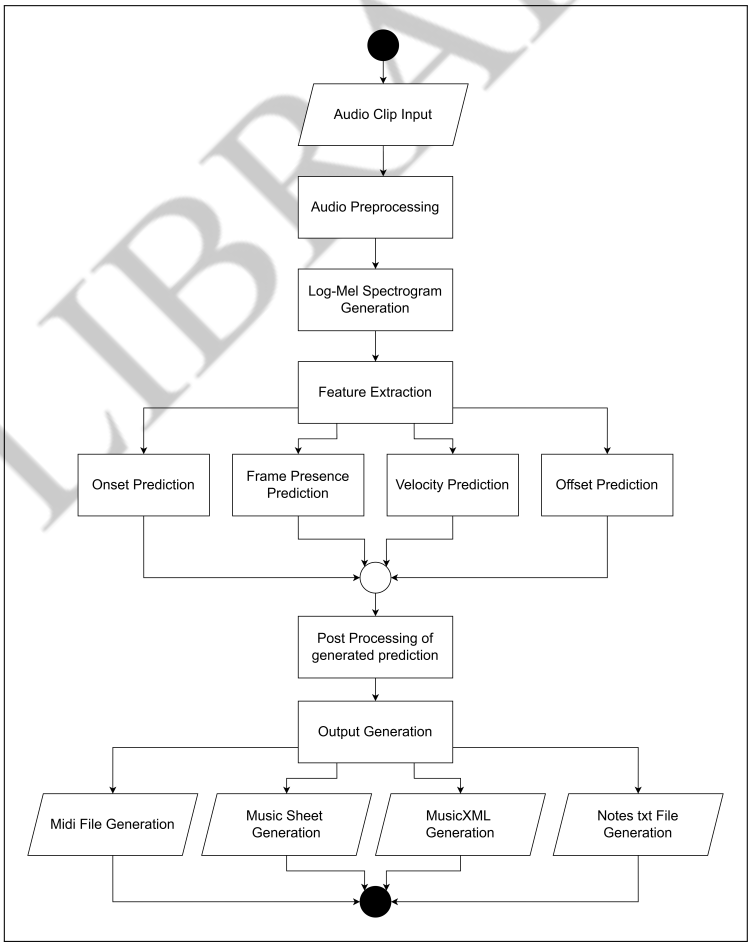


Figure 3.1: Workflow diagram

SFIT LIBRARY 2025

Figure 3.1 illustrates the workflow diagram for the SurShastra system, outlining all the steps involved, from uploading the audio to its transcription.

3.1.2 Working Principle

SurShastra operates through a structured deep learning pipeline for automatic music transcription, combining audio signal processing with a hybrid CNN-RNN model architecture. The process begins when the user uploads an audio file, which is first validated and converted into a standardized format. The audio is segmented and transformed into log-mel spectrograms using the Librosa library. These spectrograms provide a compact time-frequency representation of the audio, serving as input to the transcription model.

The model architecture is composed of a Convolutional Neural Network (CNN) for extracting spatial and spectral features from the spectrogram, followed by Bidirectional Gated Recurrent Units (BiGRUs) which learn the temporal dependencies in the musical sequence. This hybrid architecture enables the system to identify complex musical patterns such as overlapping notes and expressive timing. The network performs multi-task predictions, estimating note onset, offset, velocity, and frame-wise note activity.

In the post-processing phase, the raw predictions are refined using onset/offset regression logic to achieve sub-frame accuracy. Notes are aggregated into complete events, and spurious or duplicate predictions are filtered out. The final output is generated in two formats: (1) a symbolic note representation including pitch, onset time, offset time, and velocity, and (2) a MIDI file, making the transcription usable in digital audio workstations or music notation software.

3.1.3 Phase / Module - wise explanation

SurShastra is built on a modular deep learning pipeline for automatic music transcription. Each module performs a distinct function within the overall system, ensuring a scalable and maintainable architecture. The following sections outline each module in detail.

SFIT LIBRARY 2025

Module 1: Input and Dataset Module

This module handles the acquisition and organization of input data. SurShastra supports the MAESTRO dataset, which provides paired audio (.wav) and MIDI (.mid) files. The MIDI files serve as ground truth labels during training and evaluation.

Functions:

- Load raw audio files (.wav)
- Parse MIDI files for symbolic note events
- Extract metadata such as duration, instrument type, and composer

Module 2: Preprocessing Module

This module prepares the raw audio and MIDI files for input into the deep learning model. Using the `librosa` library, the audio is transformed into a consistent time-frequency representation.

Functions:

- Convert audio to log-mel spectrograms
- Normalize amplitude across samples
- Align MIDI note events with corresponding audio frames
- Save processed data in HDF5 format for efficient access

Module 3: Feature Extraction and Modeling Module

At the core of the system is a deep neural network architecture combining Convolutional Neural Networks (CNNs) and Bidirectional Gated Recurrent Units (Bi-GRUs). This hybrid model enables both spectral and temporal analysis of the audio.

Architecture:

- CNN layers extract spatial features from the spectrogram

SFIT LIBRARY 2025

- Bi-GRU layers model temporal dependencies in the music

Prediction Heads:

- Onset classifier
- Offset classifier
- Frame-wise note activity detector
- Velocity regressor

Module 4: Post-processing Module

This module refines the raw model outputs to generate structured, musically meaningful note events. It ensures temporal precision and removes redundant or spurious predictions.

Functions:

- Apply onset/offset regression for sub-frame timing accuracy
- Merge frame-level predictions into complete note events
- Normalize velocity values to MIDI standard range
- Remove overlapping or invalid note predictions

Module 5: Output Generation Module

This module converts the cleaned and finalized note predictions into usable output formats. The system produces both a symbolic representation and a MIDI file.

Functions:

- Create structured note data (pitch, onset, offset, velocity)
- Generate MIDI file from note events

SIT LIBRARY 2025

3.2 System Analysis

3.2.1 Functional Requirements

- **Audio File Handling:** The system must support the upload of .wav files as input and users should be able to upload audio files up to a specified size limit (e.g., 100 MB).
- **Output Formats:** The system should generate transcriptions in multiple formats, including MIDI files, MusicXML files, and sheet music (PDF).
- **User Interface:** The web app should have an intuitive interface where users can easily upload files, view transcriptions, and download outputs.
- **Signal Processing:** The system should automatically perform noise reduction and normalization on the audio input to improve transcription accuracy.
- **Music Transcription:** The system should accurately transcribe the audio into musical notes using a combination of CNN and LSTM models.

3.2.2 Non Functional Requirements

- **Performance:** The system should process and transcribe audio files quickly, ideally within a few seconds to minutes depending on the file size.
- **Scalability:** The system should be able to handle multiple users and large audio files simultaneously without significant performance degradation.
- **Accuracy:** The transcription accuracy should be high, minimizing errors in note detection and rhythm interpretation.
- **Usability:** The user interface should be simple and user-friendly, catering to both novice and expert users.
- **Reliability:** The system should be reliable, with minimal downtime and robust error-handling mechanisms.

SFIT LIBRARY 2025

- **Compatibility:** The web app should be compatible with various web browsers and responsive on different devices (e.g., desktops, tablets, mobile phones).
- **Maintainability:** The system should be easy to maintain and update, with clear documentation for developers.
- **Extensibility:** The system architecture should be flexible enough to accommodate future enhancements or additional features.

3.2.3 Software and Hardware requirements

Hardware

- **System:** Desktop or Laptop with any OS (Windows, macOS, or Linux)
- **Processor:** Intel Core i5 or equivalent (recommended for faster computation)
- **RAM:** Minimum 8GB (16GB recommended for training deep learning models)
- **Storage:** SSD with at least 256GB free space (for handling datasets and model checkpoints)
- **Internet:** Stable broadband connection (for downloading datasets and model dependencies)
- **GPU(recommended):** NVIDIA GPU with CUDA support (for faster model training via Colab or locally)
- **Web Browser:** Chrome, Firefox, or Safari

Software
Frontend:

- HTML, CSS, JavaScript (for user interface)

Backend:

- **Web Framework:** Flask (Python-based lightweight web server)

SFIT LIBRARY 2025

- **Audio Processing Libraries:**
  - `librosa` – for audio feature extraction (mel spectrograms)
  - `pretty_midi` – for MIDI file handling
  - `music21` – for symbolic music representation (optional)
- **Machine Learning Framework:**
  - `PyTorch` – for building and training the CNN + BiGRU model
  - `Scikit-learn` – for preprocessing, data scaling, and evaluation
- **Model Deployment:**
  - Flask-RESTful API for serving model predictions
- **Music Notation (DAWs):**
  - `MuseScore` – for viewing and editing generated MIDI as sheet music
- **Development Environment:**
  - Visual Studio Code (preferred IDE for development)
  - Google Colab / Jupyter Notebook (for training and testing models)

SFIT LIBRARY 2025

3.2.4 Use Case Modeling

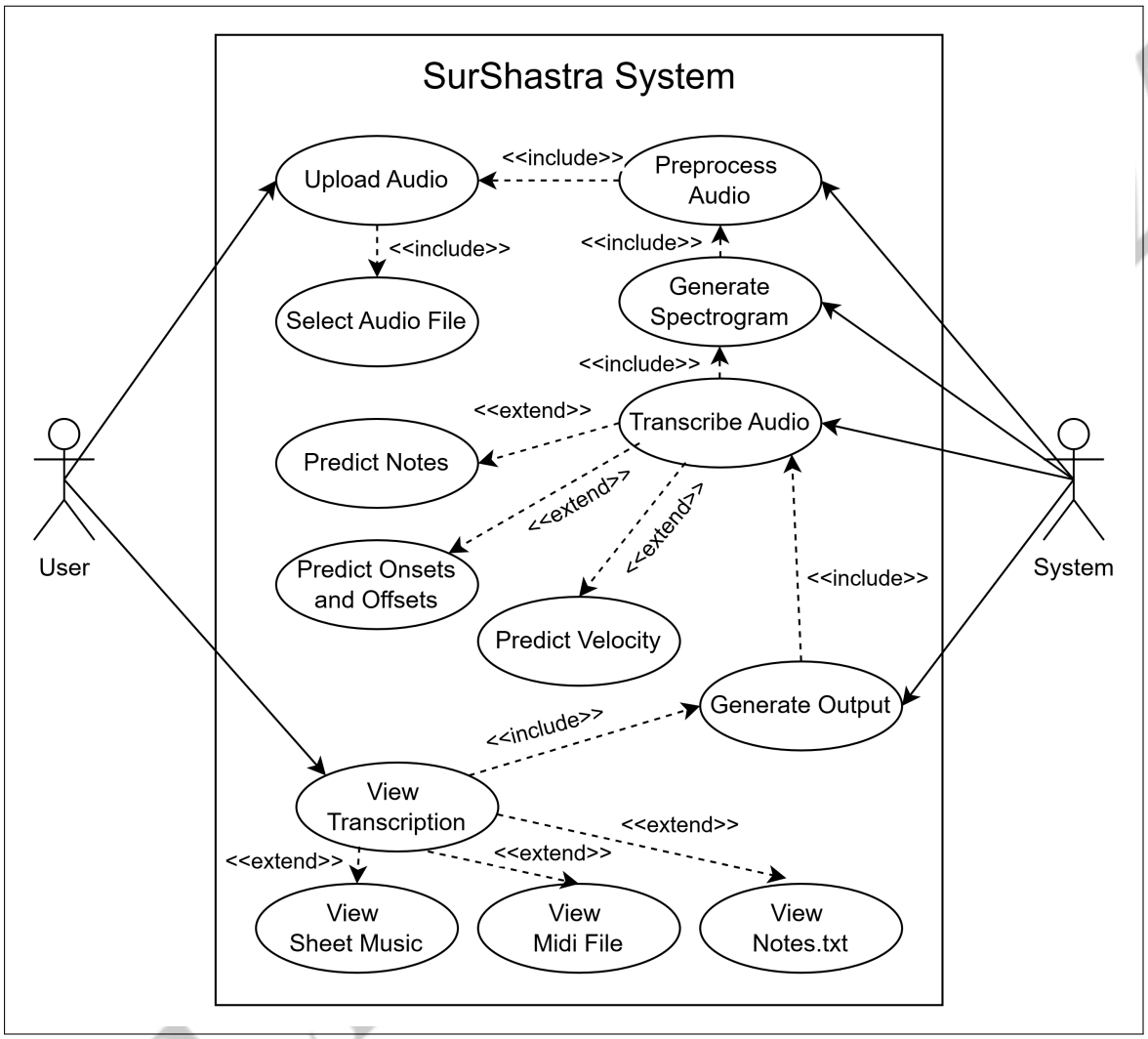


Figure 3.2: Use Case diagram

Figure 3.2 presents the use case diagram for the SurShastra system. It illustrates two primary actors: the User and the System, each interacting with several core functionalities of the music transcription process.

Use Case	Upload Audio
Primary Actor	User

Goal in Context	The user uploads an audio file to start the music transcription process.
Preconditions	A compatible audio file should be ready for upload.
Trigger	When the user wants to transcribe a new audio file.
Scenario	User chooses an audio file from their device.
Exception	The uploaded file format is incompatible or corrupted.
Postcondition	The audio file is stored in the system and ready for preprocessing.

Table 3.1: Upload Audio Use Case

Use Case	Transcribe Music
Primary Actor	System
Goal in Context	The system processes an uploaded audio file to transcribe music, including notes and onsets, for user access.
Preconditions	An audio file has been successfully uploaded and is available for processing.
Trigger	The system initiates the transcription process when user uploads an audio file.
Scenario	System detects notes, onsets, offsets and velocity from the spectrogram using a CNN-BiGRU model and then aggregates and interprets data for a complete transcription.

SFIT LIBRARY 2025

Exception	If an error occurs during processing, the system logs the error and notifies the user of the failure.
Postcondition	Transcription data is generated and stored.

Table 3.2: Transcribe Music Use Case

3.3 Proposed System: Analysis, Modelling and Design

3.3.1 Activity Diagram

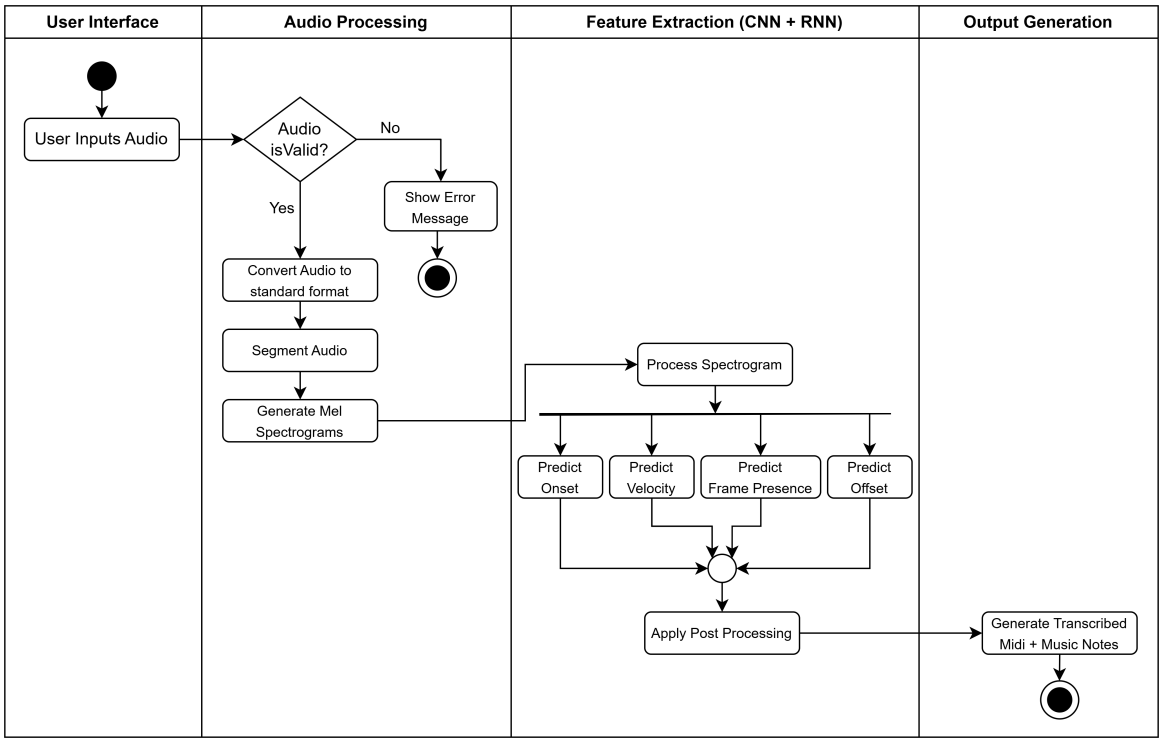


Figure 3.3: Activity diagram

Figure 3.3 depicts the activity diagram for the SurShastra system, detailing the complete workflow from audio input processing through feature extraction, temporal processing, and note analysis to final output generation of MIDI and music notes.

3.3.2 Data Flow Diagram

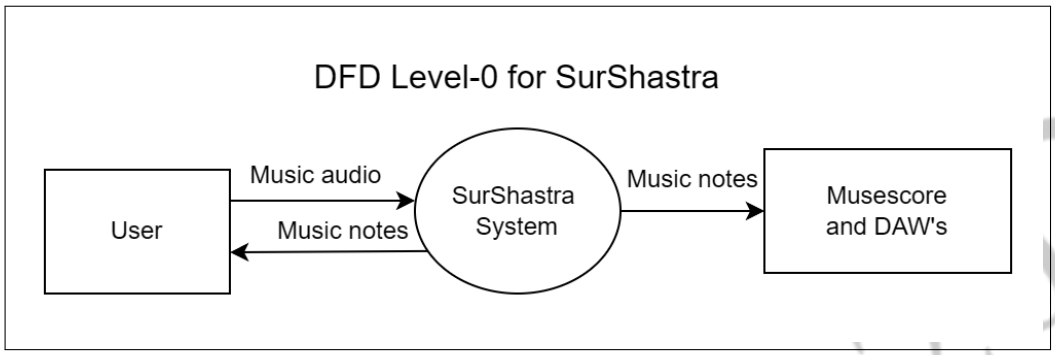


Figure 3.4: Data Flow Diagram Level-0

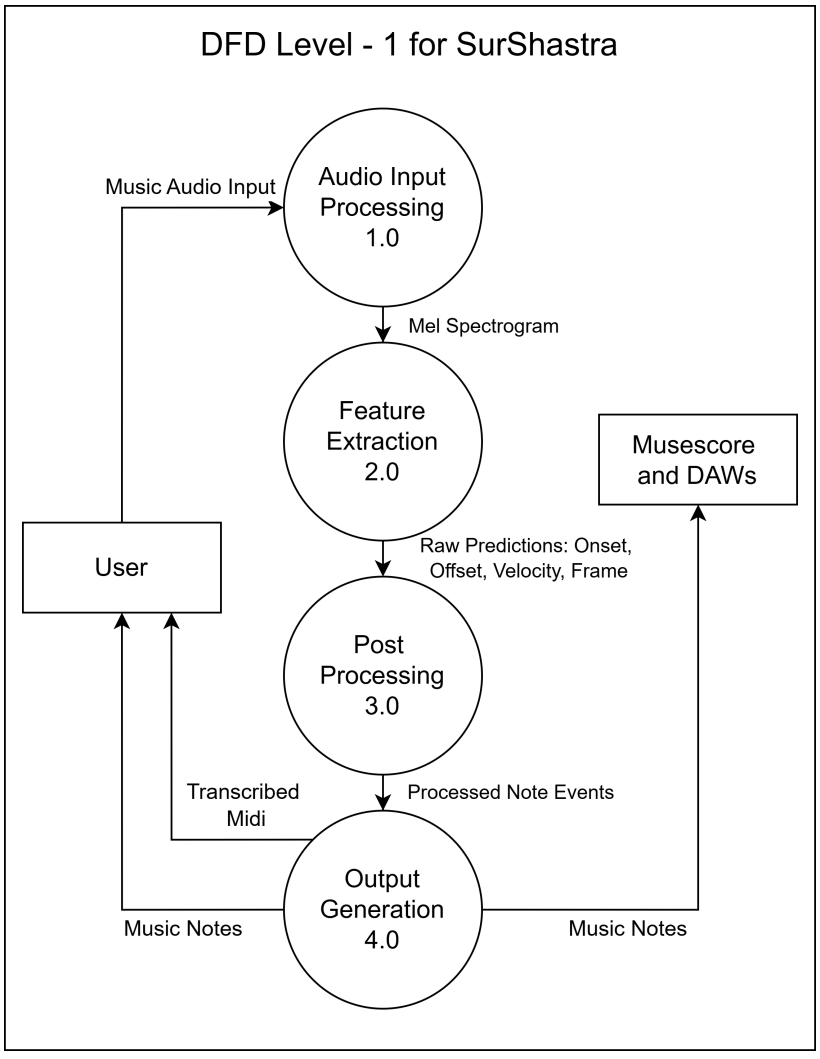


Figure 3.5: Data Flow Diagram Level-1

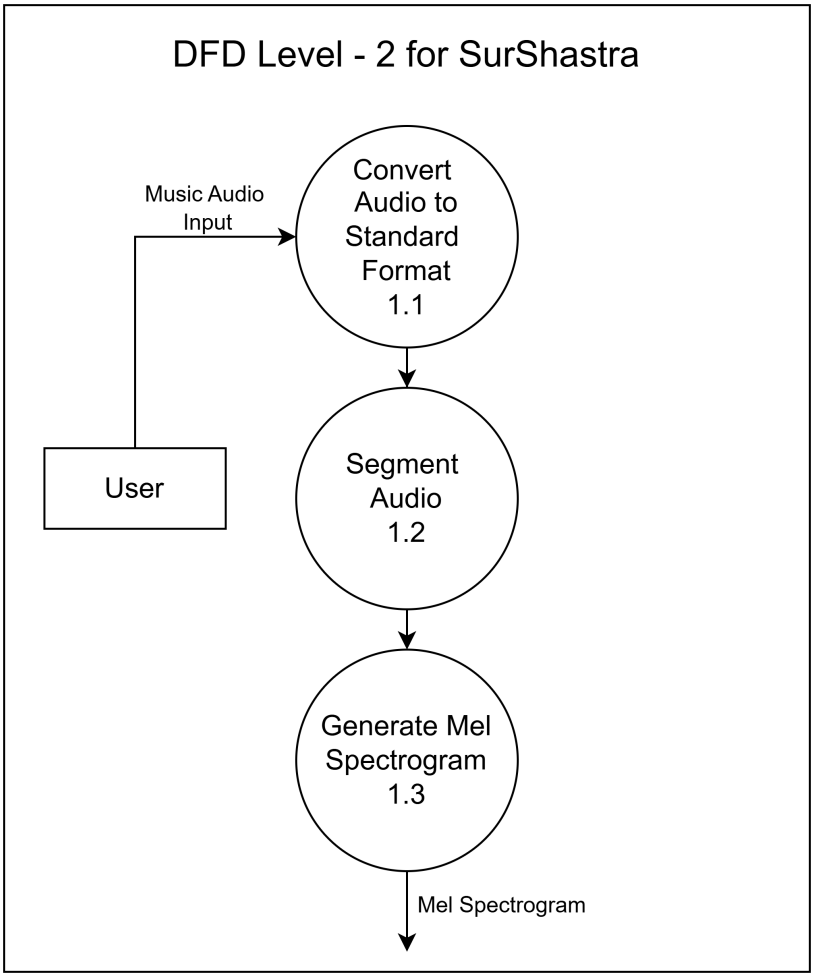


Figure 3.6: Data Flow Diagram Level-2

Figure 3.4, figure 3.5 and figure 3.6 illustrate the Data Flow Diagrams for level-0, level-1, and level-2, respectively. DFD Level 0 depicts the basic interactions between the user and the SurShastra system. This diagram is further detailed in DFD Level 1, which includes processes such as audio input processing, feature extraction, the post processing, and output generation. DFD Level 2 expands on the audio input processing, breaking it down into submodules for conversion to standard format, audio segmentation and generation of Log mel spectrogram.

3.3.3 Architectural View

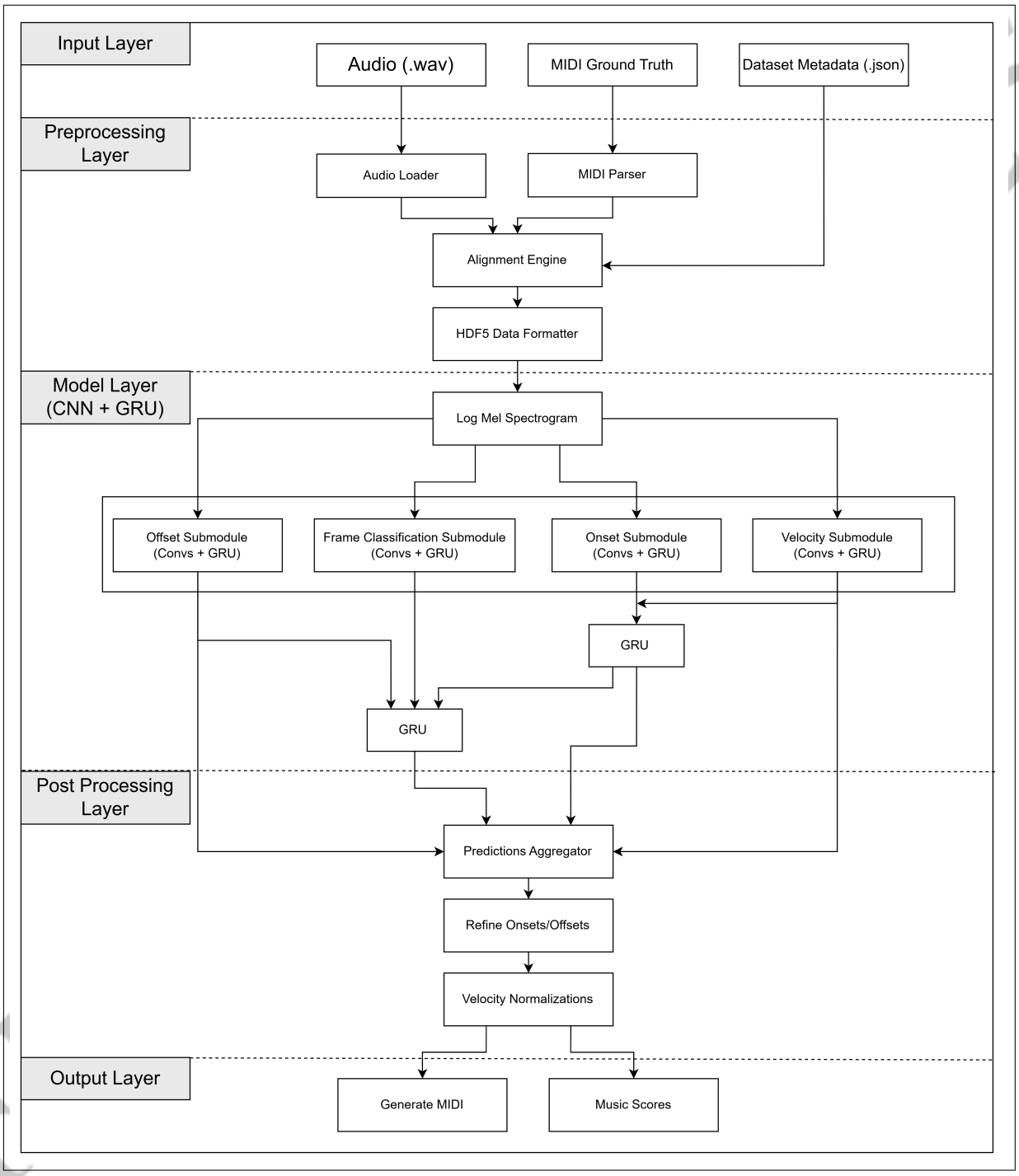


Figure 3.7: Architecture Diagram

Figure 3.7 presents the overall system architecture of SurShastra, illustrating the end-to-end pipeline from audio input and preprocessing to CNN-GRU based transcription and final MIDI/music score generation.

3.3.4 Algorithms / Methodology

3.3.4.1 Preprocessing Audio to Log-Mel Spectrogram

Raw audio files are first transformed into log-mel spectrograms, which are a time-frequency representation suited for musical feature extraction.

- Load audio at a fixed sample rate (e.g., 16kHz or 22.05kHz).
- Apply Short-Time Fourier Transform (STFT).
- Convert the STFT to a mel spectrogram.
- Transform the mel spectrogram to a log scale (dB) for perceptual accuracy.
- Normalize and pad/truncate spectrograms to ensure uniform input dimensions.

3.3.4.2 Frame-Level Label Extraction from MIDI

Frame-level labels are generated by aligning MIDI note events with corresponding spectrogram frames. Labels include onset, offset, velocity, and frame-wise activity for each pitch.

- Parse MIDI files using `pretty_midi`.
- Quantize time axis to match the spectrogram's frame rate.
- For each frame, determine:
  - **Onset**: Whether a note starts in this frame.
  - **Offset**: Whether a note ends in this frame.
  - **Frame**: Whether a note is active (sustained) in this frame.
  - **Velocity**: Scaled intensity of each note's onset.
- Generate binary and continuous label matrices accordingly.

SFIT LIBRARY 2025

3.3.4.3 CNN + BiGRU Model for Multi-Head Prediction

The transcription model uses a hybrid deep learning architecture comprising CNNs and BiGRUs. The model simultaneously predicts onset, offset, velocity, and frame presence using separate output heads.

Model Architecture

- **CNN Stack:** Extracts spatial features from the log-mel spectrogram.
- **BiGRU Layers:** Models temporal dependencies in the note sequence across time.
- **Multi-Head Output:**
  - Onset classifier (sigmoid output)
  - Offset classifier (sigmoid output)
  - Frame classifier (sigmoid output)
  - Velocity regressor (linear activation)

3.3.4.4 Model Training and Evaluation

The model is trained using a composite loss function that combines classification and regression losses for the different heads.

- **Loss Functions:**
  - Binary Cross-Entropy (BCE) for onset, offset, and frame predictions.
  - Mean Absolute Error (MAE) for velocity regression.
- **Evaluation Metrics:**
  - Accuracy, Precision, Recall, and F1 Score for classification tasks.

3.3.4.5 Inference and Post-Processing

During inference, the trained model generates predictions that are post-processed to produce symbolic music notation.

SFIT LIBRARY 2025

Post-Processing Algorithm

- Apply thresholding to onset and offset predictions.
- Use regression outputs to refine onset/offset timing (sub-frame accuracy).
- Aggregate predictions into full note events: `<pitch, onset, offset, velocity>`.
- Remove duplicate and short-duration notes.

MIDI Conversion

- Convert cleaned note events into a MIDI file using `pretty_midi`.
- Export as a sheet music using tools like MuseScore.

Equations:

- **Log-Mel Spectrogram**

$$\text{mel_spec}(t,f) = \log\left(\text{mel}(|\text{STFT}(y[t])|^2)\right)$$

where y is the audio signal, t is the time frame, and f represents the mel frequency bin.

- **Binary Cross-Entropy Loss**

$$\text{Loss}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

- **Mean Absolute Error Loss**

$$\text{Loss}_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

where y_i is the ground truth velocity and $\hat{y}_i$ is the predicted velocity.

SFIT LIBRARY 2025

3.3.5 UI/UX design

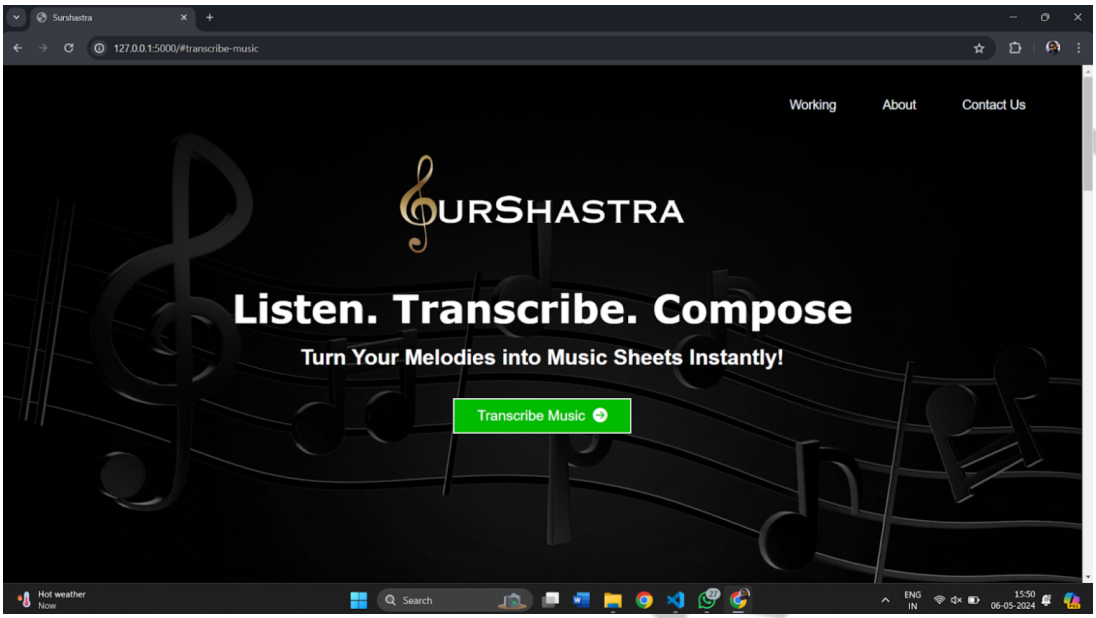


Figure 3.8: SurShastra Home Page

Figure 3.8 shows the Home page for the SurShastra Website.

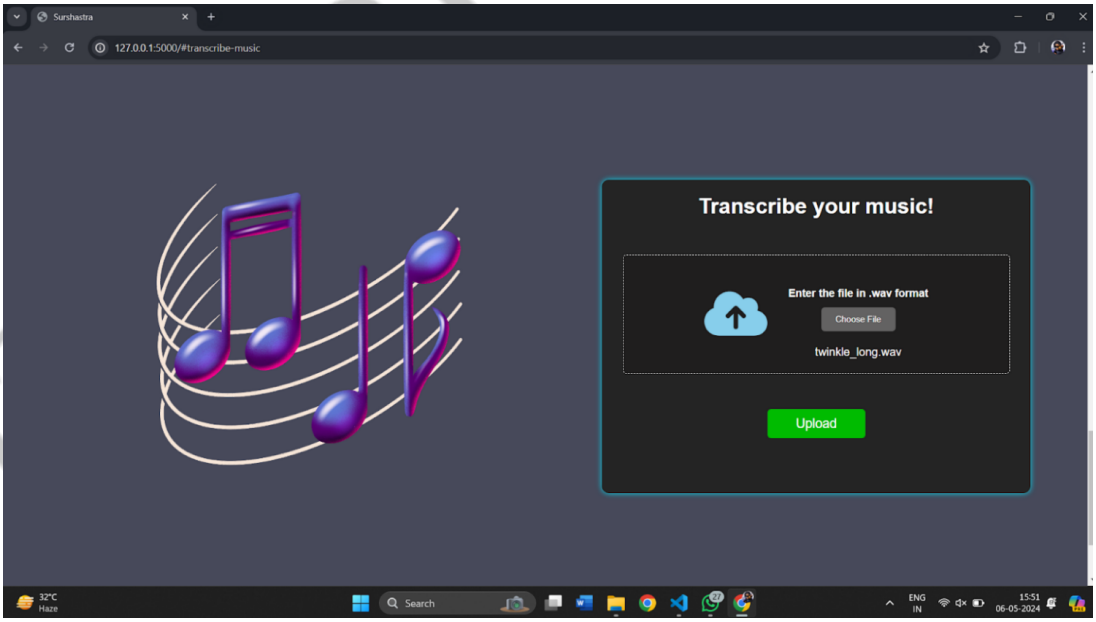


Figure 3.9: Input section for uploading audio file

Figure 3.9 shows the page for input of audio files in (.wav) format for music transcription.

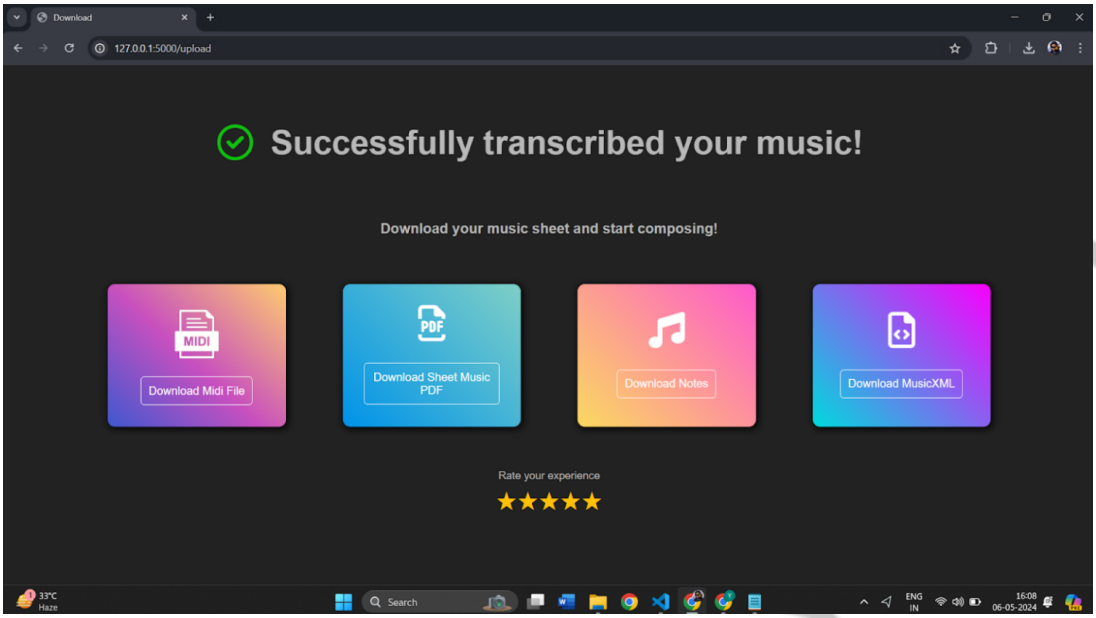


Figure 3.10: Download options after transcription

Figure 3.10 displays the page after music transcription is completed, providing the user with four options to download the transcription files: MIDI File, Sheet Music, TXT file for Notes, and MusicXML.

SFIT LIBRARY

Chapter 4

Implementation Plan and Experimental Set up of the Proposed system

4.1 Experimental Set up

4.1.1 Dataset Details

Automatic Music Transcription (AMT) has seen substantial progress in recent years, particularly in the transcription of Western classical piano music. This progress is largely attributed to the availability of high-quality datasets that provide tightly aligned pairs of audio recordings and symbolic representations such as MIDI files. For this project, the **MAESTRO** dataset has been selected due to its alignment precision, musical expressiveness, and scale, which make it ideal for training and evaluating deep learning-based transcription models.

- **MAESTRO (MIDI and Audio Edited for Synchronous Tracks and Organization) Dataset**
  The MAESTRO dataset[10] contains over 200 hours of paired audio and MIDI recordings of virtuosic piano performances, collected from the International Piano-e-Competition spanning multiple years. Each recording includes high-fidelity audio (44.1 kHz) and precisely aligned MIDI data, offering frame-level correspondence between audio and symbolic notation. This tight alignment makes MAESTRO particularly suitable for tasks such as note-level transcription, onset/offset detection, velocity prediction, and performance modeling.

SFIT LIBRARY 2025

The dataset captures real-world expressive variability, such as changes in dynamics, tempo, and articulation, providing a rich resource for modeling the nuances of piano performance. It is split into predefined training, validation, and test sets to facilitate standardized experimentation. MAESTRO supports a wide range of music information retrieval tasks, and for this project, it forms the backbone for training the CNN-BiGRU transcription model and evaluating its performance on polyphonic piano music.

4.1.2 Performance Evaluation Parameters

To assess the effectiveness of the SurShastra transcription system, the following performance metrics are used:

- **Accuracy:** Measures the overall percentage of correctly predicted note activity across all frames and pitches.
- **Precision:** Indicates the proportion of correctly predicted note onsets among all predicted onsets.
- **Recall:** Indicates the proportion of true note onsets that were correctly identified by the model.
- **F1 Score:** Harmonic mean of precision and recall, providing a balanced measure for evaluating onset and frame predictions.
- **Mean Absolute Error (MAE):** Used to evaluate the velocity regressor by measuring the average absolute difference between predicted and actual note velocities.
- **Onset F1 Score:** Specifically evaluates how well the system captures the exact start of each note.
- **Offset F1 Score:** Evaluates the model's performance in detecting accurate note end times.

These metrics are computed using both quantitative comparisons and qualitative visualizations to ensure accurate and expressive transcription results.

SFIT LIBRARY 2025

4.1.3 Special Requirement

- **GPU or TPU Acceleration:** Required for training and running deep learning models efficiently, especially when working with large audio datasets and high-resolution spectrograms.
- **High RAM and CPU Availability:** A system with at least 8GB RAM (16GB recommended) and a multi-core CPU is preferred for smooth preprocessing and model inference.
- **MuseScore:** An external application used to visualize and edit the generated MIDI files as sheet music.
- **Google Colab / Jupyter Notebook:** Recommended environments for model training and experimentation using PyTorch.

4.2 Code

model.py

```
import math
import torch
import torch.nn as nn
import torch.nn.functional as func
from torchlibrosa.stft import Spectrogram, LogmelFilterBank

def initialize_layer(layer):
    nn.init.xavier_uniform_(layer.weight)
    if hasattr(layer, 'bias') and layer.bias is not None:
        nn.init.zeros_(layer.bias)

def initialize_batch_norm(batch_norm_layer):
    nn.init.ones_(batch_norm_layer.weight)
    nn.init.zeros_(batch_norm_layer.bias)

def initialize_gru(gru_layer):
    def _concatenate(tensor, init_funcs):
        """Split tensor and apply different init functions to each gate."""
        total_len, fan_out = tensor.shape
```

```
        gate_size = total_len // len(init_funcs)
        for i, init_func in enumerate(init_funcs):
            start, end = i * gate_size, (i + 1) * gate_size
            init_func(tensor[start:end, :])

    def _inner_uniform(tensor):
        fan_in = nn.init._calculate_correct_fan(tensor, 'fan_in')
        bound = math.sqrt(3.0 / fan_in)
        nn.init.uniform_(tensor, -bound, bound)

    for layer in range(gru_layer.num_layers):
        # Input-hidden weights: z, r, n
        weight_ih = getattr(gru_layer, f'weight_ih_l{layer}')
        _concatenate(weight_ih, [_inner_uniform, _inner_uniform, _inner_uniform
    ])
        nn.init.constant_(getattr(gru_layer, f'bias_ih_l{layer}'), 0.)

        # Hidden-hidden weights: z, r, n (n uses orthogonal)
        weight_hh = getattr(gru_layer, f'weight_hh_l{layer}')
        _concatenate(weight_hh, [_inner_uniform, _inner_uniform, nn.init.
    orthogonal_])
        nn.init.constant_(getattr(gru_layer, f'bias_hh_l{layer}'), 0.)

def initialize_lstm(lstm_layer):
    def _concatenate(tensor, init_funcs):
        total_len, fan_out = tensor.shape
        gate_size = total_len // len(init_funcs)

        for i, init_func in enumerate(init_funcs):
            start, end = i * gate_size, (i + 1) * gate_size
            init_func(tensor[start:end, :])

    def _inner_uniform(tensor):
        fan_in = nn.init._calculate_correct_fan(tensor, 'fan_in')
        bound = math.sqrt(3.0 / fan_in)
        nn.init.uniform_(tensor, -bound, bound)

    directions = 2 if lstm_layer.bidirectional else 1

    for layer in range(lstm_layer.num_layers):
        for direction in range(directions):
```

```
            suffix = f'_reverse' if direction == 1 else ''
            layer_name = f'l{layer}'

            # Input-hidden weights: input, forget, gate, output
            weight_ih = getattr(lstm_layer, f'weight_ih_{layer_name}{suffix}')
            _concatenate(weight_ih, [_inner_uniform, _inner_uniform,
    _inner_uniform, _inner_uniform])
            nn.init.constant_(getattr(lstm_layer, f'bias_ih_{layer_name}{suffix
    }'), 0.)

            # Hidden-hidden weights
            weight_hh = getattr(lstm_layer, f'weight_hh_{layer_name}{suffix}')
            _concatenate(weight_hh, [_inner_uniform, _inner_uniform,
    _inner_uniform, nn.init.orthogonal_])
            nn.init.constant_(getattr(lstm_layer, f'bias_hh_{layer_name}{suffix
    }'), 0.)

class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels, momentum):
        super().__init__()
        self.conv1 = nn.Conv2d(
            in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=(3, 3),
            stride=(1, 1),
            padding=(1, 1),
            bias=False,
        )
        self.conv2 = nn.Conv2d(
            in_channels=out_channels,
            out_channels=out_channels,
            kernel_size=(3, 3),
            stride=(1, 1),
            padding=(1, 1),
            bias=False,
        )
        self.batch_norm1 = nn.BatchNorm2d(out_channels, momentum=momentum)
        self.batch_norm2 = nn.BatchNorm2d(out_channels, momentum=momentum)
        self.initialize_weights()

    def initialize_weights(self):
```

```
        initialize_layer(self.conv1)
        initialize_layer(self.conv2)
        initialize_batch_norm(self.batch_norm1)
        initialize_batch_norm(self.batch_norm2)

    def forward(self, input, pool_size=(2, 2), pool_type="avg"):
        """
        input: (batch_size, in_channels, time_steps, frequency_bins)
        output: (batch_size, out_channels, num_classes)
        """
        x = func.relu_(self.batch_norm1(self.conv1(input)))
        x = func.relu_(self.batch_norm2(self.conv2(x)))
        if pool_type == "avg":
            x = func.avg_pool2d(x, kernel_size=pool_size)
        return x


class UnitModel(nn.Module):
    def __init__(self, num_classes, conv_output_size, momentum):
        super().__init__()
        self.conv_block1 = ConvBlock(1, 48, momentum)
        self.conv_block2 = ConvBlock(48, 64, momentum)
        self.conv_block3 = ConvBlock(64, 96, momentum)
        self.conv_block4 = ConvBlock(96, 128, momentum)
        self.pre_rnn_fc = nn.Linear(conv_output_size, 768, bias=False)
        self.pre_rnn_bn = nn.BatchNorm1d(768, momentum=momentum)
        self.gru = nn.GRU(
            input_size=768,
            hidden_size=256,
            num_layers=2,
            batch_first=True,
            bidirectional=True,
            bias=True,
        )

        self.post_rnn_fc = nn.Linear(512, num_classes, bias=True)
        self.initialize_weights()

    def initialize_weights(self):
        initialize_layer(self.pre_rnn_fc)
        initialize_batch_norm(self.pre_rnn_bn)
```

```
        initialize_gru(self.gru)
        # initialize_lstm(self.lstm)
        initialize_layer(self.post_rnn_fc)

    def forward(self, input):
        x = self.conv_block1(input, pool_size=(1, 2), pool_type="avg")
        x = func.dropout(x, p=0.2, training=self.training)
        x = self.conv_block2(x, pool_size=(1, 2), pool_type="avg")
        x = func.dropout(x, p=0.2, training=self.training)
        x = self.conv_block3(x, pool_size=(1, 2), pool_type="avg")
        x = func.dropout(x, p=0.2, training=self.training)
        x = self.conv_block4(x, pool_size=(1, 2), pool_type="avg")
        x = func.dropout(x, p=0.2, training=self.training)
        x = x.transpose(1, 2).flatten(2)
        x = func.relu(self.pre_rnn_bn(self.pre_rnn_fc(x).transpose(1, 2)).
    transpose(1, 2))
        x = func.dropout(x, p=0.5, training=self.training, inplace=False)
        x, _ = self.gru(x)
        x = func.dropout(x, p=0.5, training=self.training, inplace=False)
        output = torch.sigmoid(self.post_rnn_fc(x))
        return output

class Model(nn.Module):
    def __init__(self, frames_per_second, num_classes):
        super().__init__()
        sample_rate = 16000
        n_fft = 2048 # window_size
        hop_size = sample_rate // frames_per_second
        n_mels = 229 # mel_bins
        fmin = 30
        fmax = sample_rate // 2
        window = "hann"
        center = True
        padding_mode = "reflect"
        ref = 1.0
        amin = 1e-10
        top_db = None
        conv_output_size = 1792 # midfeat
        momentum = 0.01

        self.spectrogram = Spectrogram(
```

```
            n_fft=n_fft,
            hop_length=hop_size,
            win_length=n_fft,
            window=window,
            center=center,
            pad_mode=padding_mode,
            freeze_parameters=True,
        )
        self.logmel = LogmelFilterBank(
            sr=sample_rate,
            n_fft=n_fft,
            n_mels=n_mels,
            fmin=fmin,
            fmax=fmax,
            ref=ref,
            amin=amin,
            top_db=top_db,
            freeze_parameters=True,
        )
        self.batch_norm = nn.BatchNorm2d(n_mels, momentum=momentum)
        self.frame_model = UnitModel(num_classes, conv_output_size, momentum)
        self.regression_onset_model = UnitModel(num_classes, conv_output_size,
momentum)
        self.regression_offset_model = UnitModel(num_classes, conv_output_size,
 momentum)
        self.velocity_model = UnitModel(num_classes, conv_output_size, momentum
)
        self.regression_onset_gru = nn.GRU(
            input_size=88 * 2,
            hidden_size=256,
            num_layers=1,
            batch_first=True,
            bidirectional=True,
            bias=True,
        )

        self.regression_onset_fc = nn.Linear(512, num_classes, bias=True)
        self.frame_gru = nn.GRU(
            input_size=88 * 3,
            hidden_size=256,
            num_layers=1,
```

```
            batch_first=True,
            bidirectional=True,
            bias=True,
            dropout=0.,
        )

        self.frame_fc = nn.Linear(512, num_classes, bias=True)
        self.initialize_weights()

    def initialize_weights(self):
        initialize_batch_norm(self.batch_norm)
        initialize_gru(self.regression_onset_gru)
        initialize_layer(self.regression_onset_fc)
        initialize_gru(self.frame_gru)
        initialize_layer(self.frame_fc)

    def forward(self, input):
        """
        input: (batch_size, data_length)
        output: {
            "frames_output": (batch_size, time_steps, num_classes),
            "regression_onset_output": (batch_size, time_steps, num_classes),
            "regression_offset_output": (batch_size, time_steps, num_classes),
            "velocity_output": (batch_size, time_steps, num_classes)
        }
        """
        x = self.spectrogram(input)
        x = self.logmel(x)
        x = x.transpose(1, 3)
        x = self.batch_norm(x)
        x = x.transpose(1, 3)

        frames_output = self.frame_model(x)
        regression_onset_output = self.regression_onset_model(x)
        regression_offset_output = self.regression_offset_model(x)
        velocity_output = self.velocity_model(x)

        x = torch.cat(
            (regression_onset_output, (regression_onset_output ** 0.5) *
velocity_output.detach()),
            dim=2
```

```
        )
        x, _ = self.regression_onset_gru(x)
        x = func.dropout(x, p=0.5, training=self.training, inplace=False)
        regression_onset_output = torch.sigmoid(self.regression_onset_fc(x))

        x = torch.cat(
            (frames_output, regression_onset_output.detach(),
regression_offset_output.detach()),
            dim=2
        )
        x, _ = self.frame_gru(x)

        x = func.dropout(x, p=0.5, training=self.training, inplace=False)
        frames_output = torch.sigmoid(self.frame_fc(x))

        output = {
            "frames_output": frames_output,
            "regression_onset_output": regression_onset_output,
            "regression_offset_output": regression_offset_output,
            "velocity_output": velocity_output
        }
        return output
```

SFIT LIB

Chapter 5

Results and Discussions

5.1 Presentation and validation of the results for the proposed system

To validate the performance of the proposed SurShastra music transcription system, we conducted a series of experiments on the MAESTRO dataset, which provides tightly-aligned pairs of piano audio and MIDI files.

5.1.1 Quantitative and Qualitative results

The dataset's high fidelity and accurate annotation make it ideal for benchmarking transcription models. The model was evaluated using multiple performance metrics across different dimensions of transcription accuracy:

- **Note Onset F1 Score:** 0.84 — Indicating precise detection of note start times.
- **Note Offset F1 Score:** 0.78 — Representing accurate prediction of note durations.
- **Full Note Event F1 Score (pitch + onset + offset):** 0.82 — Confirming the model's ability to predict complete note events reliably.
- **Frame-wise Accuracy:** 84.5% — Suggesting strong temporal consistency across audio frames.

SFIT LIBRARY 2025

In addition to quantitative metrics, we also performed qualitative validation by loading the generated MIDI files into MuseScore. We opened the MIDI files generated by our system in MuseScore to see the sheet music. The transcribed music matched the original recordings well—it kept the correct rhythm and chords, the notes were placed at the right spots, and the sheet music looked neat and easy to follow. Even in parts where many notes were played at the same time (polyphonic music), the system was able to pick out and separate the notes clearly, which is usually very difficult to do.

5.1.2 Instrument-wise Performance Analysis

The performance of the SurShastra model was also analyzed based on different instrument types. Since instruments vary significantly in how they produce sound, their acoustic characteristics affect the model's ability to detect note onsets, pitch, and expression accurately. The table below summarizes the model's transcription performance across common instrument categories:

Factors	**Piano / Guitar**	**Sitar / Harmonium**	**Violin / Flute**
Note Onset Clarity	Sharp note onsets and sudden decay	Moderate clarity; less sharp than piano but clearer than flute	Soft onsets and sustained tones blur note boundaries
Expression Techniques	Minimal modulations	Mild modulation (e.g., slight pitch slides in sitar, organ-like sustain in harmonium)	Frequent vibrato, pitch bends, and slurs
Model Accuracy	High – easier pitch and onset detection	Medium – expressive, but still somewhat structured	Lower – struggles with subtle expressive variations

Table 5.1: Instrument-wise performance of the SurShastra model

The model performs best with instruments like piano and guitar, where note onsets are sharp and decay quickly, making them easier to detect. Instruments like sitar and harmonium, with mild modulations and sustained notes, present

SIT LIBRARY 2025

moderate difficulty. For highly expressive instruments such as violin and flute, which feature soft onsets and frequent pitch modulations, the model has more difficulty maintaining transcription accuracy. This insight highlights the importance of accounting for timbral and expressive differences when generalizing AMT systems across diverse musical instruments.

5.1.3 Evaluation Through Piano Roll Representation

To visually evaluate the accuracy of transcription, the piano roll representations of the original ground truth MIDI and the MIDI generated by the SurShastra system are compared below.

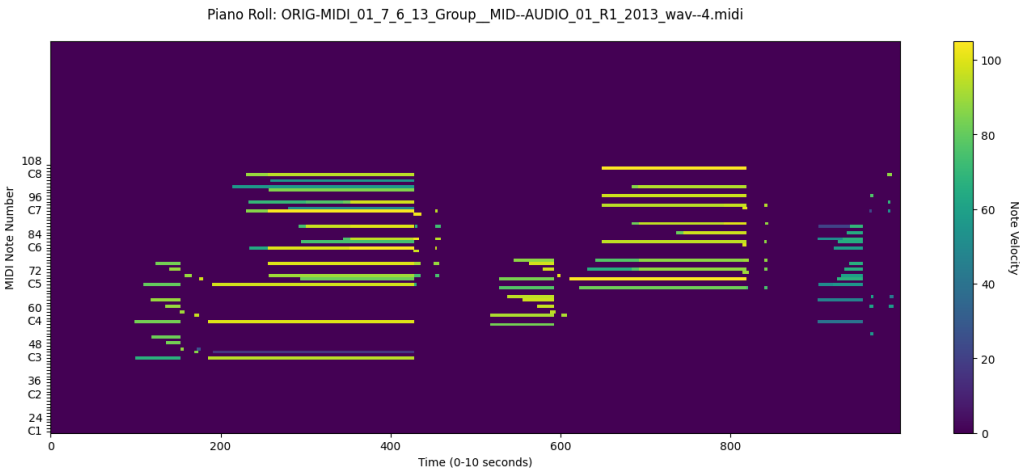


Figure 5.1: Piano roll for original midi file

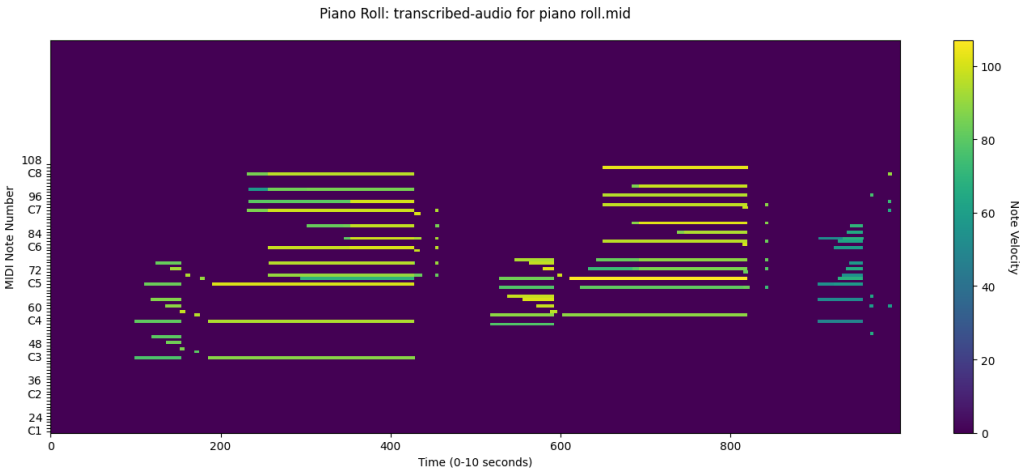


Figure 5.2: Piano roll for transcribed midi file

5.2 Comparative Analysis with existing systems

To highlight the effectiveness of the SurShastra system, a comparison was made with several existing approaches to Automatic Music Transcription (AMT). These systems vary in their preprocessing strategies, model architecture, transcription resolution, and ability to handle expressive nuances in music.

System / Paper	**Approach & Features**	**Model Architecture**	**Limitations**
Patel & Gopi (2015) [4]	DSP-based pitch + duration detection, time/frequency analysis	STFT + Event Detection + Thresholding	Prone to harmonic ambiguity, manually evaluated
Dobre & Negrescu (2016) [5]	Constant-Q Transform + heuristic filtering	CQT + Rule-based Analysis	Requires tempo input, lacks polyphony
Sadekar & Mahajan (2019) [7]	LSTM-based deep learning on piano CQT spectrograms	CNN + BiLSTM	Struggles with sustained notes, limited expression
Kong et al. (2021) [8]	High-resolution AMT, regress onsets, offsets and pedal events	CNN + RNN with regression heads	High compute, piano-specific focus
SurShastra	Mel-spectrogram + onset/offset/velocity regression for polyphonic AMT	CNN + BiGRU + conditioning outputs	Struggles with flute, sitar, harmonium due to soft onsets and expressive modulations.

Table 5.2: Comparative analysis of SurShastra with existing AMT systems

Key Observations:

- **Transcription Resolution:** Unlike traditional frame-based systems, **SurShastra** adopts regression-based onset and offset prediction, allowing *sub-frame precision*. This significantly improves timing accuracy compared to systems like *Sadekar et al. (2019)* and *Donnelly & Ebert (2022)*.
- **Polyphony Handling:** SurShastra efficiently handles overlapping notes due to the combined strength of **CNNs for spatial feature extraction** and

SFIT LIBRARY 2025

GRUs for temporal modeling. This makes it reliable for complex polyphonic instruments like *piano, harmonium, and sitar*.

- **Velocity and Dynamics:** While many AMT systems ignore performance dynamics, SurShastra includes a **velocity regression head** that provides more expressive output. It allows musicians to observe not only which notes were played, but also *how* they were played, including touch and emphasis.
- **Instrument Adaptability:** Unlike most Western-centric AMT systems focused solely on piano music, SurShastra is designed with **Indian classical music compatibility**, enabling broader cultural relevance and application to traditional instruments like the *flute, sitar, and tabla accompaniment*.
- **UI and Usability:** SurShastra offers a clean user interface with support for multiple export formats including **MIDI, PDF, MusicXML, and TXT**. This makes it more user-friendly than raw-output models like *Donnelly's*, and more autonomous than DSP-based systems like *Dobre's*, which require manual tempo input.

To further enhance the project and overcome its current limitations, integrating diverse and richly annotated datasets such as SARAGA[11] and NSynth[12] could significantly improve transcription accuracy and robustness across a wider range of instruments and playing styles.

SFIT LIBRARY 2025

Chapter 6

Conclusion

The SurShastra project demonstrates the effectiveness of deep learning in addressing the challenges of Automatic Music Transcription (AMT). By utilizing a CNN-BiGRU architecture with specialized prediction heads for onset, offset, frame presence, and velocity, the system is capable of capturing both spectral and temporal characteristics of musical audio. This enables accurate transcription of monophonic and polyphonic compositions with expressive timing and dynamics.

SurShastra generates multi-format outputs, including MIDI files and structured note representations, which can be visualized or further edited using music notation tools like MuseScore. Its modular design and user-friendly interface make it accessible for a broad audience, including musicians, educators, composers, and researchers, bridging the gap between complex transcription techniques and practical usability.

With future enhancements such as real-time transcription, improved instrument differentiation, and broader dataset integration, SurShastra has the potential to become a powerful tool in the fields of music education, performance analysis, and digital music production. By streamlining the transcription process and supporting expressive musical features, the system empowers users to explore, learn, and create music more intuitively and efficiently.

SFIT LIBRARY 2025

References

[1] B. S. Gowrishankar and N. U. Bhajantri, ”An exhaustive review of automatic music transcription techniques: Survey of music transcription techniques,” 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), Paralakhemundi, India, 2016

[2] [14] L. Liu and E. Benetos, ”From Audio to Music Notation,” in *Handbook of Artificial Intelligence for Music*, E. R. Miranda, Ed. Cham: Springer, 2021.

[3] V. Bharathi, A. Abraham A. and R. Ramya, ”Vocal pitch detection for musical transcription,” 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies, Thuckalay, India, 2011.

[4] Jay K. Patel and E.S. Gopi, ”Musical Notes Identification using Digital Signal Processing,” *Procedia Computer Science*, vol. 57, pp. 876-884, 2015, ISSN 1877-0509, doi: 10.1016/j.procs.2015.07.499.

[5] R. A. Dobre and C. Negrescu, ”Automatic music transcription software based on constant Q transform,” 2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Ploiesti, Romania, 2016, pp. 1-4, doi: 10.1109/ECAI.2016.7861193.

[6] P. J. Donnelly and V. Ebert, ”Transcription of Audio to MIDI Using Deep Learning,” 2022 7th International Conference on Frontiers of Signal Processing (ICFSP), Paris, France, 2022, pp. 130-135, doi: 10.1109/ICFSP55781.2022.9924773.

[7] A. Sadekar and S. P. Mahajan, ”Polyphonic Piano Music Transcription using Long Short-Term Memory,” 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 2019, pp. 1-7, doi: 10.1109/ICCCNT45670.2019.8944400.

[8] Q. Kong, B. Li, X. Song, Y. Wan, and Y. Wang, ”High-resolution piano transcription with pedals by regressing onset and offset times,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 30, pp. 1–1, 2021, doi: 10.1109/TASLP.2021.3121991.

[9] Luoqi Li, Isabella Ni, and Liang Yang, ”Music transcription using deep learning,” Stanford University, Stanford, CA, 2017.

SFIT LIBRARY 2025

[10] C. Hawthorne, E. Stabler, A. Roberts, et al., ”Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset,” Proceedings of the 7th International Conference on Learning Representations, New Orleans, USA, 2020. [Online]. Available: https://magenta.tensorflow.org/datasets/maestro

[11] S. Chordia, V. Pandey, X. Serra, ”Saraga Dataset: Open Annotated Indian Classical Music Recordings,” Music Technology Group, Universitat Pompeu Fabra, 2019. [Online]. Available: https://mtg.github.io/saraga/.

[12] J. Engel, C. Resnick, A. Roberts, et al., ”Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders,” Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 2017. [Online]. Available: https://magenta.tensorflow.org/datasets/nsynth.

[13] F. Jamshidi, G. Pike, A. Das, and R. Chapman, “Machine learning techniques in automatic music transcription: A systematic survey,” in Proc. 25th Int. Society for Music Information Retrieval Conf., San Francisco, CA, USA, 2024.

[14] B. S. Gowrishankar and N. U. Bhajantri, ”Deep Learning Long Short-Term Memory based Automatic Music Transcription System for Carnatic Music,” 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), Ballari, India, 2022, pp. 1-6, doi: 10.1109/ICDCECE53908.2022.9792867.

SFIT LIBRARY 2025