PROJECT REPORT

MoodRaag

Submitted by :

Srishti Sharma	**(102395006)**
Harshita	**(102215073)**
Sia	**(102215140)**
Prerna Nagpal	**(102215243)**

Group: 4O3B

Submitted to

Dr. Gaganpreet Kaur

&

Dr. Deepak Rakesh

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Department of Computer Science and Engineering

Thapar Institute of Engineering & Technology, Patiala

Jan - May (2025)

INTRODUCTION

Music is a powerful form of expression that resonates with our emotions. Whether someone is feeling joyful, heartbroken, in love, or simply reflective, there's often a song that captures that exact feeling. With the rise of digital music platforms, users now have access to vast libraries of songs. However, despite these advancements, many users still struggle to find songs that genuinely match their current mood.

Most existing music recommendation systems focus on user history, popularity, or genre classification. While helpful, these methods often overlook the emotional depth found in song lyrics, especially in regional languages like Punjabi, where lyrics are rich in sentiment and storytelling.

This project presents a Punjabi Song Recommender System that uses Natural Language Processing (NLP) to analyze song lyrics and detect the underlying mood, such as happiness, sadness, romance, or motivation. Based on this analysis, the system recommends songs that align with the user's emotional state.

The goal is to create a more personalized music experience by understanding the content of the lyrics rather than relying solely on external metadata. The project also emphasizes the importance of NLP applications in underrepresented regional languages, encouraging technological growth beyond English and other major languages.

By combining emotion detection with Punjabi lyrics, this system offers a unique and meaningful way to discover music, helping users connect more deeply with songs that reflect how they truly feel.

LITERATURE REVIEW

Multimodal Music Mood Classification (Herrera et al., Music Technology Group)

Herrera and colleagues presented an early and influential study demonstrating that music mood classification benefits from combining audio signal features with lyric-derived textual features. They extracted conventional audio descriptors (timbre, rhythm, spectral features) alongside natural-language representations from lyrics, then evaluated the predictive power of each modality independently and jointly. Their analysis showed that while audio cues capture musical affect (e.g., tempo, mode) effectively, lyrics often provide complementary semantic and affective signals, particularly for emotions that are explicitly expressed through words (e.g., sadness, longing). The paper also explored feature fusion strategies and found that standard distance-based methods and latent semantic analysis can produce meaningful improvements over single-modality baselines, especially when lyric quality and alignment to the audio are high. Herrera et al. highlighted practical issues that still persist noisy or missing lyrics, differing annotation taxonomies, and the subjective nature of mood labels, making their work a useful foundation for later multimodal MER research. [1]

Mood Classification Using Lyrics and Audio (Brilis et al., AIAI 2012)

Brilis et al. investigated mood classification from songs using a pipeline that jointly exploited acoustic and linguistic information. The authors extracted a set of acoustic descriptors and a parallel set of lyric-based linguistic features (including lexical sentiment and lexicon-matching scores), then trained standard classifiers to assign mood labels. Their experiments illustrated concrete trade-offs: lyric-derived features were especially valuable for high-level semantic moods (e.g., romantic, nostalgic), whereas acoustic features better captured arousal-related properties (e.g., energetic vs. calm). The study also emphasized preprocessing steps that significantly affect results, such as lyric normalization, stopword removal, and mapping between different mood taxonomies and argued for careful dataset curation. One useful takeaway is that, for practical recommender systems, a lyric-only or lightweight fusion approach can often yield competitive results with much lower computational cost, which is appealing for language-focused projects or low-resource deployments. [2]

Joint Sentiment Analysis of Lyrics and Audio (Schaab & Kruspe, 2024)

Schaab and Kruspe provide a recent, systematic comparison of lyric-based and audio-based sentiment models and propose several fusion strategies to combine them. Their work evaluates deep and classical models on both modalities, analyzes mismatches (cases when lyrics and audio convey conflicting emotions), and benchmarks fusion mechanisms such as probability averaging, weighted combinations, and selection by maximum confidence. They report that multimodal fusion generally improves sentiment detection performance over single-modality systems, but also document failure cases attributable to subjectivity, genre-specific conventions, and annotation inconsistencies. Importantly, the paper discusses evaluation design for music sentiment tasks and promotes more standardized taxonomies and shared datasets to reduce result variability across studies. This article is valuable for designers of lyric-first recommenders who want to understand where a lyric-only pipeline will succeed or fall short compared to multimodal alternatives. [3]

Content-driven Music Recommendation: Evolution and Challenges (Deldjoo, Schedl & Knees, 2021)

Deldjoo et al. present a comprehensive survey of content-driven music recommender systems and introduce an “onion model” that categorizes content into signal-level (audio) features, embedded metadata, expert-generated content, user-generated content, and derivative sources. The survey systematically compares algorithmic choices across these layers and maps them to classical objectives such as diversity, novelty, context-awareness, and cold-start mitigation. Of particular relevance to lyric-based mood systems, the authors highlight how semantic and affective content, such as lyrics, tags, and user comments, can enhance personalization when combined with other signals. They also discuss persistent challenges: scalability for large catalogs, cultural and cross-lingual differences in affect interpretation, and the need for explainable recommendations. The paper provides a useful conceptual framing for integrating lyric-based emotion detection into a broader recommendation stack and for arguing the value of content-aware pipelines in research and production settings. [4]

Kāvi: An Annotated Corpus of Punjabi Poetry with Emotion Labels (Saini & Kaur, 2020)

Saini and Kaur developed *Kāvi*, a pioneering corpus of Punjabi poetry annotated with emotion labels based on the classical Indian *Navrasa* taxonomy. The dataset contains nearly a thousand poetry samples, each mapped to one of nine rasa/emotion categories (e.g., śṛṅgāra, karuṇā, vīra), and the authors validated annotation consistency across multiple annotators. Their analysis demonstrates that regional poetic forms encode affective nuance that off-the-shelf multilingual models may not fully capture without fine-tuning or culturally-aware preprocessing. The *Kāvi* corpus thus provides an important resource for researchers working on Punjabi or Indic emotion analysis, offering not only labeled text but also insights into annotation schemes better aligned with South Asian affective traditions. For projects targeting Punjabi lyrics, *Kāvi* is a model for how to create language-specific training data and underscores the need for domain-specific tokenization and lexicons. [5]

IndicBERT and Indic NLP Resources (AI4Bharat)

AI4Bharat's IndicBERT family (and related IndicNLP resources) addresses the scarcity of robust pretrained encoders for Indian languages by training compact transformer models on large monolingual corpora covering many Indic languages. IndicBERT variants adopt ALBERT-style parameter sharing to remain computationally efficient while achieving performance competitive with larger multilingual models on a range of downstream tasks. The IndicNLP project also provides tokenizers, corpora, and evaluation suites that make fine-tuning for tasks like sentiment and emotion classification far more accessible for regional languages. For Punjabi lyric emotion detection, IndicBERT (and its successor models) represent a pragmatic starting point: they capture subword and morphosyntactic patterns in Indic scripts, can leverage cross-lingual transfer from related languages, and reduce the need to train large models from scratch on limited labeled data. Using IndicBERTv2 or similar models for fine-tuning yields consistent gains in low-resource scenarios and simplifies engineering for deployment. [6]

BERT-Based Approaches for Emotion Recognition (Huang et al., 2019)

Huang and collaborators investigated the emotion-recognition capability of large pretrained language models, specifically BERT, by fine-tuning them on conversational and sentence-level emotion datasets. Their experiments revealed that BERT-based models substantially outperform classical feature-engineering approaches and LSTM-based networks across standard benchmarks, thanks to richer contextualized token embeddings and attention mechanisms that capture subtle affective cues. The authors also studied transfer learning effects and showed that pretraining on large corpora followed by careful fine-tuning with task-relevant regularization (dropout, learning rate schedules) stabilizes training on smaller emotion datasets. For lyric-based mood classification, this work supports the decision to adopt transformer-based encoders (BERT/IndicBERT) rather than training shallow or RNN-based models from scratch, especially when labeled data are scarce. [7]

Contextualized Affect Representations for Emotion Recognition (CARER, Saravia et al., 2018)

Saravia et al. introduced CARER (Contextualized Affect Representations), a technique for injecting emotion-aware contextual embeddings into downstream classifiers. CARER leverages pretraining on emotion-labeled corpora to bias representations toward affective distinctions, improving performance on emotion recognition tasks where subtle cues matter. The paper demonstrates that contextualized affect embeddings, when combined with task-specific classifiers, yield state-of-the-art results on several benchmark datasets. CARER is particularly relevant for lyric analysis because song lyrics often express emotion through figurative language, metaphors, and culturally-specific references; representations pretrained to be sensitive to affective nuance help downstream classifiers disambiguate such patterns. Incorporating CARER-style objectives or using affect-enriched pretraining can be an effective augmentation when building Punjabi lyric emotion detectors where lexical sentiment lexicons are insufficient. [8]

Deep-Learning-Based Multimodal Emotion Classification for Music Videos (Pandeya et al., 2021)

Pandeya and colleagues developed a multimodal deep learning pipeline that utilizes audio, visual frames, and textual metadata to predict the emotional label of music videos. Their model combines convolutional and recurrent networks for temporal modeling of acoustic and visual features, while textual inputs (captions/lyrics) are processed with embeddings and attention layers. The study reports that integrating visual cues (artist expressions, scene context) with audio and lyrics improves classification accuracy relative to single-modality baselines, highlighting the complementary nature of modalities in conveying affect. Although music-video analysis uses additional visual information not available for many audio-only tasks, the architectural lessons, which are modality-specific encoders, cross-modal attention, and late fusion strategies, translate well to music recommendation systems that rely on lyrics and audio. For a lyric-focused Punjabi recommender, Pandeya et al.'s findings justify exploring attention-based fusion and robust per-modality encoders even if visual data are absent. [9]

Lyrics and Metadata for Genre and Emotion Analysis (Million Song / MusiXmatch studies)

Several works leveraging large-scale datasets (Million Song Dataset, MusiXmatch lyric alignments) have explored how lyric-derived emotional scores (from lexicons like ANEW) and metadata combine with audio features to predict genre, mood, and popularity. These studies typically extract valence-arousal estimates from lyrics, compute aggregated lexical affect features, and then fuse them with timbral and rhythmic descriptors for classification. Their large-scale experiments indicate that lyric-derived features improve generalization on semantic tasks (such as genre or mood grouping) and help mitigate cold-start issues where audio may be unavailable or noisy. For practical systems, these results support using TF–IDF or lexicon-based lyric encodings as efficient retrieval signals and justify a two-stage pipeline where lyric similarity retrieves candidates and a fine-tuned transformer verifies mood, an architecture that balances accuracy and computational cost for low-resource language deployments. [10]

DATASET

1. Web Scraping

To build the core of our song recommender system, we began by collecting Punjabi song data from the website https://lyricsmint.com/punjabi. Using web scraping techniques, we extracted details for approximately **4,200 Punjabi songs**. The scraped data was saved in a .json file and includes the following attributes for each song:

- Title
- Artist
- Music Director
- Lyrics
- Lyricist
- Url
- Language

2. Creating a labelled dataset

In order to train our NLP model for mood-based classification, we needed labeled data that indicated the emotional tone of each song. Since no such dataset was readily available, we generated mood labels programmatically using a custom function called mood_analyser(). This function analyzed song lyrics and matched them with predefined mood-related keywords. Each song was then classified into one of six mood categories:

- Angry
- Happy
- Sad
- Spiritual
- Energetic
- Nostalgic

The resulting labeled dataset was also saved in .json format and used for training and evaluation of the mood detection model.

MODEL

We are utilizing the **IndicBERTv2-MLM-only** model developed by **AI4Bharat**, available on Hugging Face, for our project focused on Punjabi song recommendation based on user mood, functioning as a conversational song chatbot. IndicBERTv2 is a multilingual BERT-style language model trained on the IndicCorp v2 dataset and evaluated using the IndicXTREME benchmark. It consists of 278 million parameters and supports 23 Indic languages along with English, making it highly suitable for multilingual and regional language applications like ours.

Our system leverages IndicBERTv2's contextual understanding of Punjabi to interpret user inputs—often expressed in natural, code-mixed, or emotive language—and map them to appropriate mood-based song recommendations. This allows users to interact in Punjabi and receive personalized music suggestions through a dialogue-based interface.

CODE

scraper.py

```
import requests
from bs4 import BeautifulSoup
import json
import time
from urllib.parse import urljoin
class PunjabiFolkSongScraper:
    def __init__(self):
        self.headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
        }
        self.base_url = "https://www.lyricsmint.com/punjabi"
        self.songs_data = []
    def scrape_punjabi_songs(self, max_pages=2):
        for page in range(1, max_pages + 1):
            if page == 1:
                url = self.base_url
            else:
                url = f"{self.base_url}?page={page}"
            print(f"Fetching page {page}: {url}")
            try:
                response = requests.get(url, headers=self.headers, timeout=15)
                response.raise_for_status()
                soup = BeautifulSoup(response.content, 'html.parser')
                print(f"Page title: {soup.title.string if soup.title else 'No title'}")
                song_urls = []
                selectors_to_try = [
                    "div.container a[href*='/punjabi/']",
                    "a[href*='/punjabi/']",
                    ".block a",
                    "div.block a"
                ]
                for selector in selectors_to_try:
                    links = soup.select(selector)
                    if links:
                        print(f"Found {len(links)} links with selector: {selector}")
                        for link in links:
                            href = link.get('href')
                            if href:
                                print(f"  Link found: {href}")
                                if (href.startswith('/') and
                                    not any(x in href.lower() for x in ['#', 'javascript:', 'mailto:', 'tel:']) and
                                    len(href.split('/')) >= 3 and
                                    href not in song_urls):
                                    song_urls.append(href)
                        break
                print(f"Found {len(song_urls)} unique song URLs on page {page}")
                for i, song_url in enumerate(song_urls):
                    if not song_url.startswith('http'):
                        full_url = "https://www.lyricsmint.com" + song_url
                    else:
                        full_url = song_url
                    print(f"Processing song {i+1}/{len(song_urls)}: {full_url}")
                    self.scrape_individual_song(full_url)
                    time.sleep(2)
                print(f"Completed page {page}. Total songs collected: {len(self.songs_data)}")
                time.sleep(3)
            except requests.RequestException as e:
                print(f"Network error on page {page}: {e}")
            except Exception as e:
                print(f"Unexpected error on page {page}: {e}")
    def scrape_individual_song(self, song_url):
        try:
            response = requests.get(song_url, headers=self.headers, timeout=15)
            response.raise_for_status()
            soup = BeautifulSoup(response.content, 'html.parser')
            title = self.extract_title(soup)
            metadata = self.extract_metadata(soup)
            lyrics = self.extract_lyrics(soup)
            if title and lyrics:
                song_data = {
                    'title': title,
                    'artist': metadata.get('singer', 'Unknown'),
                    'music': metadata.get('music', 'Unknown'),
                    'lyricist': metadata.get('lyricist', 'Unknown'),
                    'lyrics': lyrics,
                    'url': song_url,
                    'language': 'punjabi',
                }
                self.songs_data.append(song_data)
                print(f"Successfully scraped: {title}")
            else:
                print(f"Missing title or lyrics for: {song_url}")

        except requests.RequestException as e:
            print(f"Network error scraping {song_url}: {e}")
        except Exception as e:
            print(f"Error scraping {song_url}: {e}")
    def extract_title(self, soup):
        title_selectors = [
            "div.pt-4.pb-2 h2",
            "h1",
            "h2",
            ".title",
            "[class*='title']"
        ]
        for selector in title_selectors:
            title_element = soup.select_one(selector)
            if title_element:
                title = title_element.get_text(strip=True)
                title = title.replace("Lyrics", "").replace("lyrics", "").strip()
                if title:
                    return title
        if soup.title:
            title = soup.title.string.replace("Lyrics", "").replace("lyrics", "").strip()
            if title:
                return title
        return None
    def extract_metadata(self, soup):
        metadata = {}
        table = soup.find("table")
        if table:
            rows = table.find_all("tr")
            for row in rows:
                cells = row.find_all(["td", "th"])
                if len(cells) >= 2:
                    key = cells[0].get_text(strip=True).lower()
                    value_cell = cells[1]
                anchor = value_cell.find("a")
                value = anchor.get_text(strip=True) if anchor else value_cell.get_text(strip=True)

                if "singer" in key or "artist" in key:
                    metadata["singer"] = value
                elif "music" in key or "composer" in key:
                    metadata["music"] = value
                elif "lyricist" in key or "writer" in key:
                    metadata["lyricist"] = value
        return metadata
    def extract_lyrics(self, soup):
        lyrics_selectors = [
            "div.text-base.lg\\:text-lg .pb-2",
            "div[class*='text-base'] p",
            ".lyrics p",
            "[class*='lyrics'] p",
            "div p"
        ]
        for selector in lyrics_selectors:
            content_div = soup.select_one(selector.split()[0])
            if content_div:
                paragraphs = content_div.find_all("p")
                lyrics_lines = []
                for p in paragraphs:
                    text = p.get_text(strip=True)
                    if len(text) > 10 and not any(word in text.lower() for word in ['advertisement', 'ads',
'click', 'visit']):
                        lyrics_lines.append(text)

                if lyrics_lines:
                    return "\n".join(lyrics_lines)
        return None
    def save_data(self, filename='punjabi_lyricsmint_songs.json'):
        if not self.songs_data:
            print("No songs data to save!")
            return None
        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(self.songs_data, f, ensure_ascii=False, indent=2)
        print(f"Saved {len(self.songs_data)} songs to {filename}")
        if self.songs_data:
            print("\n Summary:")
            print(f"Total songs: {len(self.songs_data)}")
            print("Sample titles:")
            for i, song in enumerate(self.songs_data):
                print(f"  {i+1}. {song['title']} - {song['artist']}")
        return filename
    def debug_page_structure(self, url):
        try:
            response = requests.get(url, headers=self.headers, timeout=15)
            soup = BeautifulSoup(response.content, 'html.parser')
            print(f"\nDebug info for: {url}")
            print(f"Page title: {soup.title.string if soup.title else 'None'}")
            print(f"Total links found: {len(soup.find_all('a'))}")
            print(f"Total divs found: {len(soup.find_all('div'))}")
            print("\nSample links found:")
            for i, link in enumerate(soup.find_all('a')[:15]):
                href = link.get('href', 'No href')
                text = link.get_text(strip=True)[:50]
                print(f"  {i+1}. {href} -> '{text}'")
            print("\nSample div classes:")
            for div in soup.find_all('div', class_=True)[:10]:
                classes = ' '.join(div.get('class', []))
                print(f"Div classes: {classes}")
        except Exception as e:
            print(f"Debug error: {e}")
if __name__ == "__main__":
    scraper = PunjabiFolkSongScraper()
    print("Debugging page structure...")
    scraper.debug_page_structure("https://www.lyricsmint.com/punjabi")
    print("\nStarting scraping...")
    scraper.scrape_punjabi_songs(max_pages=299)
    if scraper.songs_data:
        scraper.save_data()
    else:
        print(" No songs were scraped. Check the page structure and selectors.")
```

model.py

```
import torch
import json
import pandas as pd
import numpy as np
import re
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support,
classification_report, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
from datasets import Dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
    EarlyStoppingCallback
)
import torch.nn as nn
import os
def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    text = re.sub(r'\s+', ' ', text.strip())
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'[!?]{2,}', lambda x: x.group()[0], text)
    return text
def load_data(file_path="punjabi_songs_with_mood.json"):
    with open(file_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    df = pd.DataFrame(data)
    df = df.dropna(subset=['lyrics', 'mood'])
    df['lyrics'] = df['lyrics'].apply(preprocess_text)
    df = df[df['lyrics'].str.strip() != '']
    df = df[df['lyrics'].str.split().str.len() >= 4]
    print(f"Loaded {len(df)} samples")
    print("Class distribution:\n", df['mood'].value_counts())
    return df
def prepare_dataset(df):
    label_encoder = LabelEncoder()
    df['labels'] = label_encoder.fit_transform(df['mood'])
    train_df, val_df = train_test_split(df, test_size=0.2, stratify=df['labels'], random_state=42)
    train_dataset = Dataset.from_pandas(train_df[['lyrics', 'labels']].reset_index(drop=True))
    val_dataset = Dataset.from_pandas(val_df[['lyrics', 'labels']].reset_index(drop=True))
    return {'train': train_dataset, 'validation': val_dataset}, label_encoder
def tokenize_dataset(dataset, model_name):
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    def tokenize_fn(examples):
        return tokenizer(
            examples["lyrics"],
            truncation=True,
            padding="max_length",
            max_length=256,
            return_attention_mask=True
        )
    return {k: v.map(tokenize_fn, batched=True) for k, v in dataset.items()}, tokenizer
class WeightedTrainer(Trainer):
    def __init__(self, *args, class_weights=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.class_weights = class_weights
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        loss_fct = nn.CrossEntropyLoss(
            weight=self.class_weights.to(outputs.logits.device) if self.class_weights is not None else
None,
            label_smoothing=0.1
        )
        loss = loss_fct(outputs.logits.view(-1, model.config.num_labels), labels.view(-1))
        return (loss, outputs) if return_outputs else loss
def train_model(tokenized_dataset, tokenizer, label_encoder, model_name, output_dir):
    model = AutoModelForSequenceClassification.from_pretrained(
        model_name,
        num_labels=len(label_encoder.classes_),
        hidden_dropout_prob=0.2,
        attention_probs_dropout_prob=0.2,
        classifier_dropout=0.3
    )
    class_weights = torch.FloatTensor(
        compute_class_weight(
            class_weight='balanced',
            classes=np.unique(tokenized_dataset["train"]["labels"]),
            y=tokenized_dataset["train"]["labels"]
        )
    ).clamp(min=0.3, max=3.0)
    print("Class weights:", class_weights)
    training_args = TrainingArguments(
        output_dir=output_dir,
        eval_strategy="steps",
        eval_steps=100,
        save_strategy="steps",
        save_steps=100,
        learning_rate=1.5e-5,
        per_device_train_batch_size=8,
        per_device_eval_batch_size=16,
        num_train_epochs=15,
        weight_decay=0.05,
        warmup_ratio=0.1,
        load_best_model_at_end=True,
        metric_for_best_model="eval_f1",
        greater_is_better=True,
        logging_steps=50,
        fp16=torch.cuda.is_available(),
        lr_scheduler_type='cosine_with_restarts',
        optim="adamw_torch_fused",
        max_grad_norm=1.0,
        report_to="none",
        seed=42,
        dataloader_pin_memory=False,
        gradient_accumulation_steps=2,
        adam_epsilon=1e-6,
        group_by_length=True,
    )
    trainer = WeightedTrainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_dataset["train"],
        eval_dataset=tokenized_dataset["validation"],
        compute_metrics=lambda p: {
            "accuracy": accuracy_score(p.label_ids, np.argmax(p.predictions, axis=1)),
            "f1": precision_recall_fscore_support(
                p.label_ids, np.argmax(p.predictions, axis=1), average='weighted')[2]
        },
        callbacks=[
            EarlyStoppingCallback(
                early_stopping_patience=5,
                early_stopping_threshold=0.005
            )
        ],
        class_weights=class_weights
    )
    print("\nStarting training with loss convergence monitoring...")
    trainer.train()
    history = trainer.state.log_history
    train_losses = [x['loss'] for x in history if 'loss' in x]
    eval_losses = [x['eval_loss'] for x in history if 'eval_loss' in x]
    print("\nTraining Report:")
    if train_losses and eval_losses:
        print(f"Final Train Loss: {train_losses[-1]:.4f}")
        print(f"Final Eval Loss: {eval_losses[-1]:.4f}")
        print(f"Loss Difference: {abs(train_losses[-1] - eval_losses[-1]):.4f}")
    eval_results = trainer.evaluate()
    print(f"\nValidation Accuracy: {eval_results['eval_accuracy']:.2%}")
    print(f"Validation F1: {eval_results['eval_f1']:.2%}")
    if eval_results['eval_accuracy'] >= 0.80:
        print("Target accuracy of 80% achieved!")
    elif eval_results['eval_accuracy'] < 0.75:
        print("Try: Reduce learning rate to 1e-5 and increase epochs to 20")
    return trainer
def evaluate_model(trainer, tokenized_dataset, label_encoder):
    print("\n" + "="*60)
    print("COMPREHENSIVE MODEL EVALUATION")
    print("="*60)
    predictions = trainer.predict(tokenized_dataset["validation"])
    y_pred = np.argmax(predictions.predictions, axis=1)
    y_true = predictions.label_ids
    accuracy = accuracy_score(y_true, y_pred)
    precision, recall, f1, support = precision_recall_fscore_support(y_true, y_pred,
average='weighted')
```

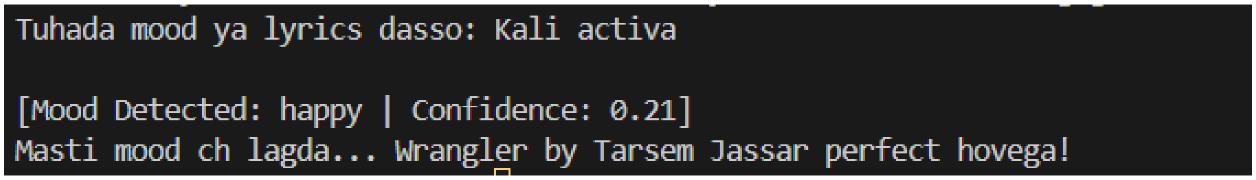
```
print(f"\nOVERALL PERFORMANCE:")
print(f"Accuracy:  {accuracy:.4f} ({accuracy:.2%})")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-Score:  {f1:.4f}")
print(f"\nPER-CLASS PERFORMANCE:")
class_report = classification_report(y_true, y_pred,
                                     target_names=label_encoder.classes_,
                                     digits=4)
print(class_report)
print(f"\nCONFUSION MATRIX:")
cm = confusion_matrix(y_true, y_pred)
print("Predicted ->")
print("Actual |")
for i, actual_class in enumerate(label_encoder.classes_):
    print(f"{actual_class:>10} | ", end="")
    for j in range(len(label_encoder.classes_)):
        print(f"{cm[i][j]:>6}", end=" ")
    print()
print(f"\n" + "="*60)
if accuracy >= 0.80:
    print(f"SUCCESS: Model achieved {accuracy:.2%} accuracy (Target: 80%)")
else:
    print(f"BELOW TARGET: Model achieved {accuracy:.2%} accuracy (Target: 80%)")
    print("Suggestions:")
    if accuracy < 0.75:
        print("- Reduce learning_rate to 1e-5")
        print("- Increase num_train_epochs to 20")
    else:
        print("- Fine-tune learning_rate to 1.2e-5")
        print("- Increase num_train_epochs to 18")
print("="*60)

return {
    'accuracy': accuracy,
    'precision': precision,
    'recall': recall,
    'f1': f1,
    'predictions': y_pred,
    'true_labels': y_true
    }
def save_model(trainer, tokenizer, label_encoder, output_dir, evaluation_results=None):
    os.makedirs(output_dir, exist_ok=True)
    print(f"Saving model to {output_dir}...")
    trainer.save_model(output_dir)
    print("Saving tokenizer...")
    tokenizer.save_pretrained(output_dir)
    print("Saving label mappings...")
    with open(f"{output_dir}/label_mapping.json", "w") as f:
        label_mapping = {
            "id2label": {i: label for i, label in enumerate(label_encoder.classes_)},
            "label2id": {label: i for i, label in enumerate(label_encoder.classes_)}
        }
        json.dump(label_mapping, f, indent=2)
        print(f"Label mapping: {label_mapping['id2label']}")

    if evaluation_results:
        print("Saving evaluation results...")
        with open(f"{output_dir}/evaluation_results.json", "w") as f:
            results_to_save = {
                'accuracy': float(evaluation_results['accuracy']),
                'precision': float(evaluation_results['precision']),
                'recall': float(evaluation_results['recall']),
                'f1': float(evaluation_results['f1']),
                'class_names': label_encoder.classes_.tolist(),
                'target_accuracy': 0.80,
                'achieved_target': evaluation_results['accuracy'] >= 0.80
            }
            json.dump(results_to_save, f, indent=2)
    print(f"\nAll files saved successfully to: {output_dir}")
    print("Files created:")
    for file in os.listdir(output_dir):
        print(f"  - {file}")
def main():
    DATA_FILE = "punjabi_songs_with_mood.json"
    MODEL_NAME = "ai4bharat/IndicBERTv2-MLM-only"
    OUTPUT_DIR = "./punjabi_mood_model"
    print("Starting Punjabi Mood Classification Training Pipeline")
    print("="*60)
    print("Step 1: Loading data...")
    df = load_data(DATA_FILE)
    print("\nStep 2: Preparing dataset...")
    dataset, label_encoder = prepare_dataset(df)
    print("\nStep 3: Tokenizing data...")
    tokenized_dataset, tokenizer = tokenize_dataset(dataset, MODEL_NAME)
    print("\nStep 4: Training model...")
    trainer = train_model(
        tokenized_dataset,
        tokenizer,
        label_encoder,
        MODEL_NAME,
        OUTPUT_DIR
    )
    print("\nStep 5: Evaluating model...")
    evaluation_results = evaluate_model(trainer, tokenized_dataset, label_encoder)
    print("\nStep 6: Saving model...")
    save_model(trainer, tokenizer, label_encoder, OUTPUT_DIR, evaluation_results)
    print(f"\n" + "="*60)
    print("TRAINING PIPELINE COMPLETED")
    print(f"Final Model Accuracy: {evaluation_results['accuracy']:.2%}")
    print(f"Model saved to: {OUTPUT_DIR}")
    print("="*60)
if __name__ == "__main__":
    main()
```

[lyrics-recommender.py]

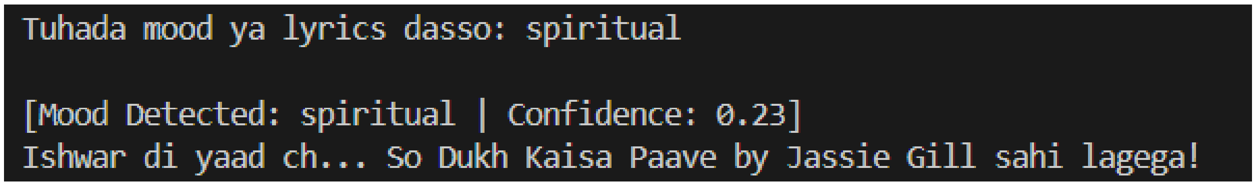
```
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import json
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import random
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
class EmotionClassifier:
    def __init__(self, model_path):
        self.model =
AutoModelForSequenceClassification.from_pretrained(model_path).to(device)
        self.tokenizer = AutoTokenizer.from_pretrained(model_path)
        self.label_map = {i: label for i, label in enumerate([
          'angry', 'energetic', 'happy', 'nostalgic', 'romantic', 'sad', 'spiritual'
        ])}
        self.model.eval()
    def predict_emotion(self, text):
        inputs = self.tokenizer(text, return_tensors="pt", padding=True, truncation=True).to(device)
        with torch.no_grad():
           outputs = self.model(**inputs)
        probs = torch.nn.functional.softmax(outputs.logits, dim=1)
        prediction = torch.argmax(probs, dim=1).item()
        return self.label_map[prediction], probs[0][prediction].item()
class SongDatabase:
    def __init__(self, dataset_path):
        with open(dataset_path, "r", encoding='utf-8') as f:
           self.songs = json.load(f)
    def get_songs_by_mood(self, mood):
        return [song for song in self.songs if song.get('mood') and song['mood'].lower() ==
mood.lower()]
    def get_all_lyrics(self):
        return [song.get('lyrics', '') for song in self.songs]
    def get_song_by_lyrics(self, lyrics):
        for song in self.songs:
           if song.get('lyrics', '') == lyrics:
              return song
        return None
class EnhancedLyricMatcher:
    def __init__(self, song_database):
        self.song_database = song_database
        self.vectorizer = TfidfVectorizer()
        self._fit_vectorizer()
    def _fit_vectorizer(self):
        lyrics_corpus = self.song_database.get_all_lyrics()
        self.tfidf_matrix = self.vectorizer.fit_transform(lyrics_corpus)
    def find_similar_lyrics(self, input_lyrics, mood, top_n=3):
        input_vec = self.vectorizer.transform([input_lyrics])
        cosine_similarities = cosine_similarity(input_vec, self.tfidf_matrix).flatten()
        matching_songs = self.song_database.get_songs_by_mood(mood)
        matching_lyrics = [song.get('lyrics', '') for song in matching_songs]
        matching_indices = [i for i, lyrics in enumerate(self.song_database.get_all_lyrics()) if lyrics
in matching_lyrics]
```

```
        ranked_indices = sorted(matching_indices, key=lambda i: cosine_similarities[i],
reverse=True)
        if not ranked_indices:
            emotion_groups = {
                'sad': ['sad', 'nostalgic'],
                'happy': ['happy', 'energetic'],
                'romantic': ['romantic'],
                'angry': ['angry'],
                'spiritual': ['spiritual'],
                'nostalgic': ['nostalgic', 'sad'],
                'energetic': ['energetic', 'happy']
            }
            fallback_labels = emotion_groups.get(mood, [mood])
            fallback_songs = []
            for label in fallback_labels:
                fallback_songs.extend(self.song_database.get_songs_by_mood(label))
            return random.sample(fallback_songs, min(top_n, len(fallback_songs)))
        top_indices = ranked_indices[:top_n]
        return [self.song_database.get_song_by_lyrics(self.song_database.get_all_lyrics()[i]) for i
in top_indices]
class ResponseGenerator:
    def __init__(self):
        self.response_templates = {
            'sad': [
                "Tera dil dukhi lagda ae... {} eh gaana sunn ke shayad changa lage.",
                "Thoda emotional mood lagda... {} sun ke halka feel karega.",
                "Eh wali vibe aa rahi ae... {} sahi rahega tenu."
            ],
            'happy': [
                "Vibe badi happy lagdi... {} try kar!",
                "Masti mood ch lagda... {} perfect hovega!",
                "Laggda ae tu hass reha si... {} sun!"
            ],
            'romantic': [
                "Ishq wala love lagda... {} sun!",
                "Dil romantic ho gaya? {} perfect match ae!",
                "Love vibes mil rahi ne... {} enjoy kar!"
            ],
            'nostalgic': [
                "Purani yaadan aa gayian? {} sun ke aur feel aayegi.",
                "Thoda throwback mood lagda... {} sahi choice ae!",
                "Nostalgia wali vibe aa gayi... {} perfect ae!"
            ],
            'angry': [
                "Thoda gussa lagda... {} naal energy mildi ae!",
                "Aggression feel ho rahi ae... {} perfect release ae!",
                "Angry mode ch? {} changi vibe dinda ae!"
            ],
            'spiritual': [
                "Ruhaniyat wali feeling aa rahi... {} sun ke mann shant ho jaega!",
                "Bhakti da mood lagda... {} perfect match hai!",
                "Spiritual zone ch hai tu... {} sun le!",
                "Ishwar di yaad ch... {} sahi lagega!"
            ],
            'energetic': [
                "Energy full on hai! {} da gaana naach layi perfect hai!",
                "Josh wale mood ch hai tu... {} sun ke aur pumped up ho ja!",
                "Energetic vibes mil rahe... {} try kar!",
                "Dance floor bula raha... {} sahi gaana hai!"
            ]
        }
    def generate_response(self, mood, song_name):
        templates = self.response_templates.get(mood, ["Here's a song you might like: {}"])
        response = random.choice(templates).format(song_name)
        return response
class PunjabiLyricsChatbot:
    def __init__(self, model_path, dataset_path):
        self.classifier = EmotionClassifier(model_path)
        self.song_db = SongDatabase(dataset_path)
        self.matcher = EnhancedLyricMatcher(self.song_db)
        self.responder = ResponseGenerator()
    def handle_input(self, user_input):
        mood, confidence = self.classifier.predict_emotion(user_input)
        print(f"\n[Mood Detected: {mood} | Confidence: {confidence:.2f}]")
        recommended_songs = self.matcher.find_similar_lyrics(user_input, mood, top_n=1)
        if recommended_songs:
            song = recommended_songs[0]
            song_title = song.get('title', 'Unknown Title')
            artist = song.get('artist', 'Unknown Artist')
            response = self.responder.generate_response(mood, f"{song_title} by {artist}")
            return response
        else:
            return "Koi gaana nahi mil reha. Thoda hor lyrics dasso."
if __name__ == "__main__":
    MODEL_PATH = "./punjabi_mood_model1"
    DATASET_PATH = "./punjabi_songs_with_mood.json"
    chatbot = PunjabiLyricsChatbot(MODEL_PATH, DATASET_PATH)
    print(">> Punjabi Mood Lyrics Bot\nType 'exit' to quit.\n")
    while True:
        user_input = input("Tuhada mood ya lyrics dasso: ")
        if user_input.lower() == "exit":
            print("Alvida! Fer milange!")
            break
        response = chatbot.handle_input(user_input)
        print(response)
```

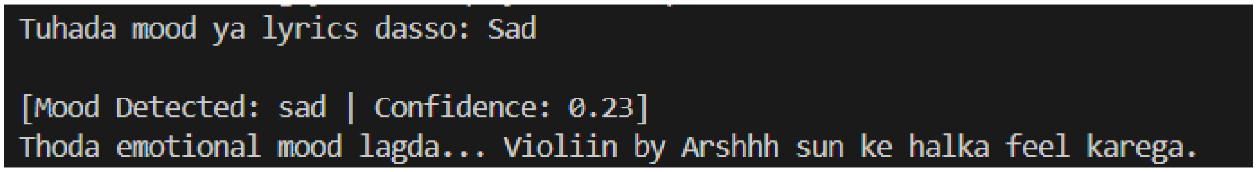
RESULTS



```
Tuhada mood ya lyrics dasso: Kali activa

[Mood Detected: happy | Confidence: 0.21]
Masti mood ch lagda... Wrangler by Tarsem Jassar perfect hovega!
```

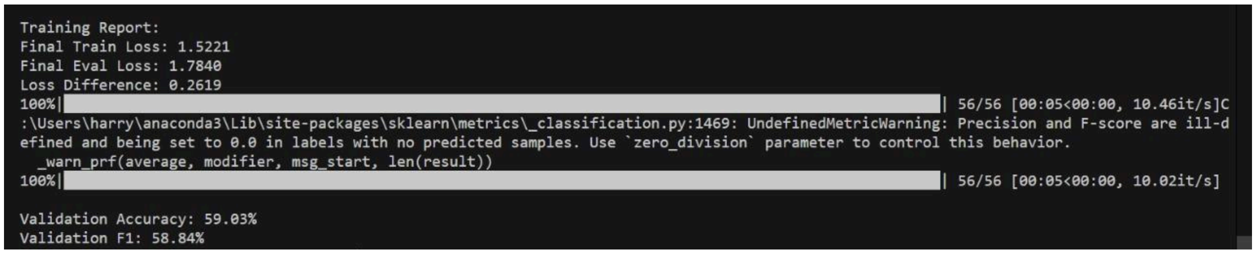


```
Tuhada mood ya lyrics dasso: spiritual

[Mood Detected: spiritual | Confidence: 0.23]
Ishwar di yaad ch... So Dukh Kaisa Paave by Jassie Gill sahi lagega!
```

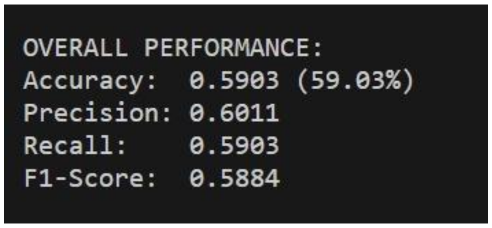


```
Tuhada mood ya lyrics dasso: Sad

[Mood Detected: sad | Confidence: 0.23]
Thoda emotional mood lagda... Violiin by Arshhh sun ke halka feel karega.
```

Model Report:



```
Training Report:
Final Train Loss: 1.5221
Final Eval Loss: 1.7840
Loss Difference: 0.2619
100%|████████████████████████████████████████| 56/56 [00:05<00:00, 10.46it/s]C
:\Users\harry\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
100%|████████████████████████████████████████| 56/56 [00:05<00:00, 10.02it/s]

Validation Accuracy: 59.03%
Validation F1: 58.84%
```



```
OVERALL PERFORMANCE:
Accuracy:  0.5903 (59.03%)
Precision: 0.6011
Recall:    0.5903
F1-Score:  0.5884
```

CONCLUSION

This project presents a Punjabi Song Recommender System that bridges the gap between user emotions and music discovery using Natural Language Processing. Unlike traditional recommenders relying on popularity or genre, our system focuses on the emotional depth within Punjabi song lyrics, a language rich in cultural and poetic expression.

We gathered and prepared a dataset of over **4,200 Punjabi songs** through web scraping, including metadata and lyrics. Using a mood analyzer function, songs were labeled into six emotional categories: happy, sad, nostalgic, angry, energetic, and spiritual—enabling effective training and evaluation.

By applying modern NLP techniques and a multilingual model (fine-tuned on mood-labelled data), the system captures both the semantic and emotional context of lyrics, allowing more accurate mood classification than keyword-based methods. This work highlights the importance of developing NLP tools for low-resource regional languages and demonstrates the potential for creating culturally aware, emotion-based music recommendation systems.

Overall, the project combines data collection, emotion detection, and language understanding to provide a meaningful, mood-based way to discover Punjabi music, paving the way for future advances in regional language AI applications.

REFERNECES

[1]. Herrera, P., Oliviera, J. L., & Gouyon, F. (2011). *Affective music classification: Analysis of features from audio and lyrics.* Music Technology Group, Universitat Pompeu Fabra.

[2]. Brilis, G., Lyras, N., & Kermanidis, K. L. (2012). *Mood classification using lyrics and audio: A decision engine for Affective Music Recommender Systems (AMRS).* In Proceedings of the 8th International Conference on Artificial Intelligence Applications & Innovations (AIAI).

[3]. Schaab, L., & Kruspe, A. (2024). *Joint sentiment analysis of lyrics and audio in music.* arXiv preprint arXiv:2401.09876.

[4]. Deldjoo, Y., Schedl, M., & Knees, P. (2021). *Content-driven music recommendation: Evolution, state of the art, and challenges.* Information Processing & Management, 58(5), 1–28.

[5]. Saini, J. R., & Kaur, N. (2020). *Kāvi: An annotated corpus of Punjabi poetry with emotion labels.* In Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020).

[6]. Kakwani, D., Kunchukuttan, A., Golla, S., et al. (2020). *IndicBERT: A pretrained language model for Indic languages.* AI4Bharat.

[7]. Huang, C. H., Lee, H. Y., & Chen, L. S. (2019). *Emotion recognition using BERT-based models.* In Proceedings of the IEEE International Conference on Affective Computing and Intelligent Interaction (ACII).

[8]. Saravia, E., Liu, H., Huang, Y., Moturu, S., & Cambria, E. (2018). *CARER: Contextualized affect representations for emotion recognition.* In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[9]. Pandeya, Y. R., Shi, Y., & Lee, J. (2021). *Deep-learning-based multimodal emotion classification for music videos.* IEEE Access, 9, 70907–70921.

[10]. Fell, M., & Sporleder, C. (2014). *Lyrics-based analysis for genre and mood classification using the Million Song Dataset.* In Proceedings of the International Society for Music Information Retrieval (ISMIR).