Overall Architecture & Techniques Used

1. Feature Extraction Technique
- **Mel-Frequency Cepstral Coefficients (MFCCs)** - Primary features
- **Chromagram** - Pitch class profiles
- **Mel Spectrogram** - Frequency representation
- **Spectral Contrast** - Spectral characteristics
- **Tonnetz** - Tonal relationships
- **Zero Crossing Rate** - Time-domain feature

2. Model Architecture
- **CNN + BiLSTM + Temporal Attention Hybrid Model**
  - **Conv1D Layers**: Feature extraction from spectral patterns
  - **BiLSTM Layers**: Sequence modeling (forward + backward context)
  - **Temporal Attention**: Focus on emotionally relevant time segments
  - **Dense Layers**: Final classification

3. Training Techniques
- **Class Weighting**: Handles imbalanced datasets
- **Learning Rate Scheduling**: Adaptive learning rate reduction
- **Early Stopping**: Prevents overfitting
- **Strong Regularization**: Dropout + L2 regularization
- **Data Augmentation**: Time/Frequency masking, pitch shifting, noise addition

---

📊 **Complete Workflow from Start to Finish**

PHASE 1: DATA PREPROCESSING
```python
```

Workflow:

Audio Files → Feature Extraction → Normalization → Padding → Train/Test Split

```

**Steps:**

1. **Load Audio Files** from dataset directory

2. **Extract Features** using MFCCs and other spectral features

3. **Normalize Features** using StandardScaler (mean=0, std=1)

4. **Pad Sequences** to equal length for batch processing

5. **Split Data** into training (85%) and validation (15%) sets

### **PHASE 2: MODEL TRAINING**

```python

# Workflow:

Build Model → Compile → Train with Callbacks → Save Best Model

```

Steps:

1. **Build CNN-BiLSTM-Attention** architecture

2. **Compile** with Adam optimizer and sparse categorical crossentropy

3. **Train** with validation monitoring

4. **Apply Callbacks**:

  - Early Stopping (patience=12)

  - Learning Rate Reduction (patience=5)

  - Model Checkpointing

5. **Save** best model, scaler, and class labels

PHASE 3: PREDICTION PIPELINE

```python

# Workflow:

Load Audio → Extract Features → Preprocess → Model Prediction → Postprocess → Results

```

Steps:

1. **Load trained model** and preprocessing artifacts

2. **Extract features** from new audio file

3. **Apply same preprocessing** as training (scaling, padding)

4. **Predict** emotion probabilities

5. **Postprocess** results into human-readable format

🏗️ **Detailed Technical Workflow**

Training Phase Detailed Steps:

1. **Data Loading**

  ```python
  files = glob.glob("dataset/*/*.wav")  # Find all audio files
  data = process_files_parallel(files)  # Parallel processing
  ```

2. **Feature Extraction**

  ```python
  # For each audio file:
  mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=40)
  chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
  mel = librosa.feature.melspectrogram(y=audio, sr=sr)
  # Combine all features into single matrix
  ```

3. **Data Preparation**
  ```

```python
# Pad all sequences to same length
features_padded = pad_sequences(features, max_length)

# Normalize features
scaler = StandardScaler()
features_normalized = scaler.fit_transform(features_padded)

# Encode labels
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
```

4. **Model Construction**

```python
# Architecture:
Input → Conv1D → BN → ReLU → Pooling → Dropout
     → Conv1D → BN → ReLU → Pooling → Dropout
     → BiLSTM → Dropout
     → BiLSTM → Dropout
     → Temporal Attention
     → Dense → Dropout
     → Output (Softmax)
```

5. **Training Execution**

```python
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
```

```
    batch_size=32,
    class_weight=class_weights,
    callbacks=[early_stopping, lr_reduction, checkpoint]
)
```

Prediction Phase Detailed Steps:

1. **Model Loading**

```python
model = load_model("models/ser_model.keras")
scaler = load_pickle("models/scaler.pkl")
classes = load_pickle("models/classes.pkl")
```

2. **Feature Processing**

```python
# Extract features from new audio
features = extract_feature(audio_path)

# Apply same preprocessing as training
features_scaled = scaler.transform(features)
features_padded = pad_to_model_shape(features_scaled)
```

3. **Prediction**

```python
# Get model predictions
predictions = model.predict(features_padded)

# Get top emotion and confidence
```

  emotion_idx = np.argmax(predictions)

  emotion = classes[emotion_idx]

  confidence = predictions[0][emotion_idx]

  ```

4. **Result Formatting**

  ```python

  results = {

     "emotion": emotion,

     "confidence": float(confidence),

     "all_scores": {classes[i]: float(score) for i, score in enumerate(predictions[0])},

     "timestamp": (0.0, audio_duration)

  }

  ```

⚙️ **Hyperparameters & Configuration**

Model Hyperparameters:

- **Learning Rate**: 1e-3 with reduction to 1e-6

- **Batch Size**: 32

- **Epochs**: 100 (with early stopping)

- **Dropout Rates**: 0.5-0.6

- **L2 Regularization**: 0.001

- **Optimizer**: Adam

Feature Extraction Parameters:

- **Sample Rate**: 22050 Hz

- **MFCCs**: 40 coefficients

- **FFT Size**: 2048
  ```

- **Hop Length**: 512

- **Mel Bands**: 128

📈 **Expected Performance Metrics**

Good Training Should Show:

- **Training Accuracy**: 92-95%

- **Validation Accuracy**: 90-93%

- **Accuracy Gap**: <3%

- **Validation Loss**: Decreasing then stabilizing

Warning Signs:

- **Gap >5%**: Overfitting

- **Validation loss increasing**: Overfitting

- **Both accuracies low**: Underfitting

---

pp**Deployment Ready Features**

1. **Model Persistence**: Saved in .keras format

2. **Preprocessing Artifacts**: Scaler and classes saved

3. **Error Handling**: Robust exception handling

4. **Memory Efficiency**: Batch processing for large files

5. **Real-time Ready**: Can be adapted for streaming audio

.