Lampiran: Codebase Penelitian

6.1 Library Import

```
import mido
import math
from music21 import *
import pandas as pd
```

6.2 Koefisien Szymkiewicz-Simpson (Overlap Coefficient)

Fungsi: Menghitung derajat kemiripan dari dua data yang diberikan

```
def overlap_coefficient(a, b):
    overlap = 0
    similarityConstant = 6
    listA = list(enumerate(a))
    listB = list(enumerate(b))
    minValue = min(len(a), len(b))

    print(a, b)

    similarSet = []

    counter = 1

    if (len(listA) <= similarityConstant) | (len(listB) <= similarityConstant):
        similarityConstant = math.ceil(minValue / 2)
        #similarityConstant = minValue

    minValueNumber = min(len(a), len(b)) - similarityConstant + 1
    for valueA in listA:
        if (valueA[0] + similarityConstant) <= len(listA):
            for valueB in listB:
                if (valueB[0] + similarityConstant) <= len(listB):
                    counter = counter + 1
                    aSelected = a[valueA[0]:valueA[0]+similarityConstant]
                    bSelected = b[valueB[0]:valueB[0]+similarityConstant]
                    print(aSelected, bSelected)
                    if aSelected == bSelected:
                        overlapExist = False
                        for similars in similarSet:
                            if (similars[0] == aSelected) & (similars[1] == bSelected):
                                overlapExist = True
                                print('overlap exist. skipping...')
                        if overlapExist == False:
                            overlap = overlap + 1
                            similarSet.append([aSelected, bSelected])
                            print('overlap')
    print("Overlap Count: " + str(overlap) + ", Denominator Value: " + str(minValue))
    """if overlap > minValueNumber:
        overlap = minValueNumber"""
    if overlap > minValue:
        overlap = minValue
    #print(minValue, minValueNumber)
    #return overlap / minValueNumber
    return overlap / minValue
```

6.3 Operasi Internal MIDI

Nama Fungsi	Kegunaan Fungsi
timing(track, ticksPerBeat)	Fungsi untuk mendapatkan BPM serta Jumlah Ketukan (Time Signature)
determineNotationBasedOnNumber(pitchNumber)	Fungsi untuk mendapatkan kunci dari angka nada
determineNumberBasedOnNotation(notation)	Fungsi untuk mendapatkan angka nada dari kunci

rootChordDetection(chordList)	Fungsi untuk mendapatkan nada akar dari chord
splitMeasure(timeSig, tickRate, data)	Fungsi untuk memisahkan birama – birama pada data MIDI yang didapat
simplifyChord(data)	Fungsi untuk menjalankan fungsi rootChordDetection() terhadap data MIDI yang didapat

```
def timing(track, ticksPerBeat):
    tempo = []
    timeSig = []
    for msg in track:
        if msg.type == 'set_tempo':
            tempo.append([msg.time, math.floor(60 / (msg.tempo  / 1000000))])
        elif msg.type == 'time_signature':
            timeSig.append([msg.time, msg.numerator, msg.denominator])
    return [tempo, timeSig]

def determineNotationBasedOnNumber(pitchNumber):
    notation = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"]
    octaveNum = math.floor(pitchNumber / 12)
    notationModulo = pitchNumber % 12
    return notation[notationModulo] + str(octaveNum)

def determineNumberBasedOnNotation(notation):
    notationList = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"]
    octave = notation[-1]
    octaveNum = int(octave) * 12
    pitch = 0
    for note in list(enumerate(notationList)):
        if note[1] == notation[0:-1]:
            pitch = note[0]
    return octaveNum + pitch

def rootChordDetection(chordList):
    notations = []
    for pitch in chordList:
        pitchNotation = determineNotationBasedOnNumber(pitch)
        notations.append(pitchNotation)
    receivedChord = chord.Chord(notations)
    chordRoot = receivedChord.root()
    return determineNumberBasedOnNotation(str(chordRoot))
```

```python
def splitMeasure(timeSig, tickRate, data):
    curTime = 0
    curMeasure = 0

    # For Collecting Measures in Temp and add them as one in MeasureSplit Array
    measureSplit = []
    tempMeasure = []

    # For Time Signature Change Tracking
    timeSigChange = len(timeSig) - 1
    timeSigCounter = 0

    # For markers on Where Timing Changes

    for note in data:
        # Check if it's entering new measures
        # If it's, send the previous temp measures into the split if it's not empty
        if len(note) > 0:
            measure = math.floor(note[0][0] / tickRate / timeSig[timeSigCounter][1])
            if measure > curMeasure:
                measureSplit.append(tempMeasure)
                tempMeasure = []
                curMeasure = measure

            # Insert the note to the measure
            tempMeasure.append(note[0])

            # If the Time Signature Timestamp exceeds current time, add a counter to the next one
            if timeSigChange != timeSigCounter:
                if timeSig[timeSigCounter][0] >= curTime:
                    timeSigCounter = timeSigCounter + 1
    return measureSplit
```

```python
def simplifyChords(data):
    time = 0
    melodies = []
    chord = []

    for note in data:
        """
        If the time changes, insert the note
        If the time is the same (indicates a chord), collect as a chord
        """
        if time < note[0]: # If this is a new timestamp
            if len(chord) > 1: # Check if it is a chord or not (two note or more)
                chordPitches = []
                start = 0
                duration = 0

                # If it is a chord, get the chord pitches as notation and
                for pitch in chord:
                    start = pitch[0]
                    if (duration > pitch[2]) | (duration == 0):
                        duration = pitch[2]
                    chordPitches.append(pitch[1])

                chordRoot = rootChordDetection(chordPitches)
                melodies.append([[start, chordRoot, duration]])
            else: # If it isn't a chord
                melodies.append(chord)
            chord = [] # Reset the chord container if it's already inside the melodies array
            chord.append(note) # Add the new note
            time = note[0] # Set the current time marker with the new timestamp
        else: # If there are other note in the same timestamp
            chord.append(note)
    return melodies
```

6.4 Modul Deteksi Plagiarisme

Nama Fungsi	Kegunaan Fungsi
loadMidi(midifile)	Fungsi untuk loading file MIDI dan mendapatkan data kontennya
dataProcessing(data, tickRate)	Fungsi untuk membersihkan data MIDI yang dimuat

plagiarismDetection(midiA, midiB)	Fungsi untuk mengolah file MIDI A dan file MIDI B menghasilkan derajat kemiripan dari kedua file MIDI

```
def loadMidi(midifile):
    # Load the MIDI File
    mid = mido.MidiFile(midifile)
    ticksPerBeat = mid.ticks_per_beat
    timingList = timing(mid.tracks[0], ticksPerBeat)
    bpm = timingList[0]
    timeSig = timingList[1]

    curTime = 0
    listOfNotes = []
    tempPressedNote = []
    tempChordDuration = 0
    for msg in mid.tracks[1]:
        if not msg.is_meta:
            if msg.type == 'note_on':
                curTime = curTime + msg.time
                tempPressedNote.append([msg.note, curTime])
            elif msg.type == 'note_off':
                curTime = curTime + msg.time
                for pressedNote in tempPressedNote:
                    if pressedNote[0] == msg.note:
                        listOfNotes.append([pressedNote[1], msg.note, curTime - pressedNote[1]])
                        tempPressedNote.remove(pressedNote) #Empty the tempPressedNote because the note is released

    return listOfNotes, bpm, timeSig, ticksPerBeat
```

```
def dataProcessing(data, tickRate):
    cleanedIntervals = []
    cleanedRhythms = []
    cleanIntervals = []
    cleanRhythm = []
    prevNote = 0
    for measure in data:
        if len(measure) > 0:
            for pitch in measure:
                # Getting intervals of pitch
                interval = 0
                if prevNote != 0:
                    interval = pitch[1] - prevNote
                prevNote = pitch[1]

                cleanIntervals.append(interval)

                # Getting duration of note in fraction
                duration = pitch[2] / tickRate
                cleanRhythm.append(round(duration, 2))
            cleanedIntervals.append(cleanIntervals)
            cleanedRhythms.append(cleanRhythm)
            cleanRhythm = []
            cleanIntervals = []
    return [cleanedIntervals, cleanedRhythms]
```

```python
def plagiarismDetection(midiA, midiB):
    musicA, bpmA, timeSigA, tickRateA = loadMidi(midiA)
    musicB, bpmB, timeSigB, tickRateB = loadMidi(midiB)

    simplifiedA = simplifyChords(musicA)
    simplifiedB = simplifyChords(musicB)

    splitMeasureA = splitMeasure(timeSigA, tickRateA, simplifiedA)
    splitMeasureB = splitMeasure(timeSigB, tickRateB, simplifiedB)

    processedA = dataProcessing(splitMeasureA, tickRateA)
    processedB = dataProcessing(splitMeasureB, tickRateB)

    overlapsInterval = []

    for x in range(len(processedA[0])-1):
        selectedMeasureA = []
        for aNoteOne in processedA[0][x]:
            selectedMeasureA.append(aNoteOne)
        for aNoteTwo in processedA[0][x+1]:
            selectedMeasureA.append(aNoteTwo)
        for y in range(len(processedB[0])-1):
            selectedMeasureB = []
            for bNoteOne in processedB[0][y]:
                selectedMeasureB.append(bNoteOne)
            for bNoteTwo in processedB[0][y+1]:
                selectedMeasureB.append(bNoteTwo)
            detection = overlap_coefficient(selectedMeasureA, selectedMeasureB)
            overlapsInterval.append(detection)

    sumDetection = 0

    for i in overlapsInterval:
        sumDetection = sumDetection + i
```

```python
    if len(overlapsInterval) > 0:
        averagedDetection = sumDetection / len(overlapsInterval)
    else:
        averagedDetection = 0

    averagedDetection = round(averagedDetection, 2)
    print("Detection Result (Melody): " + str(averagedDetection))

    overlapsRhythm = []

    for rhythmMeasureA in processedA[1]:
        for rhythmMeasureB in processedB[1]:
            rhythmDetection = overlap_coefficient(rhythmMeasureA, rhythmMeasureB)
            overlapsRhythm.append(rhythmDetection)

    sumRhythmDetection = 0

    for j in overlapsRhythm:
        sumRhythmDetection = sumRhythmDetection + j

    if len(overlapsRhythm) > 0:
        averagedRhythmDetection = sumRhythmDetection / len(overlapsRhythm)
    else:
        averagedRhythmDetection = 0

    averagedRhythmDetection = round(averagedRhythmDetection, 2)
    print("Detection Result (Rhythm): " + str(averagedRhythmDetection))
    return [averagedDetection, averagedRhythmDetection]
```

6.5 Confusion Matrix

Fungsi: Untuk mendapatkan nilai Akurasi, Presisi, Recall, dan F-Measure.

```python
def confusionMatrix(truePos, falsePos, trueNeg, falseNeg):
    if (truePos == 0) | (falsePos == 0) | (trueNeg == 0) | (falseNeg == 0):
        accuracy = 0
        precision = 0
        recall = 0
        fMeasure = 0
    else:
        accuracy = (truePos + trueNeg) / (truePos + falsePos + trueNeg + falseNeg)
        precision = truePos / (truePos + falsePos)
        recall = trueNeg / (trueNeg + falseNeg)
        fMeasure = (2 * precision * recall) / (precision + recall)

    return [accuracy, precision, recall, fMeasure]
```

6.6 Main Process

Proses utama dalam codebase deteksi plagiarisme

```
directory = "mid-file"

dataframe1 = pd.read_excel('python-data-music.xlsx')

detectionResults = []
counterCase = 0

threshold = 0.25
correct = 0
correctRhythm = 0

#Melodies
truePosMelody = 0
falsePosMelody = 0
trueNegMelody = 0
falseNegMelody = 0

#Rhythms
truePosRhythm = 0
falsePosRhythm = 0
trueNegRhythm = 0
falseNegRhythm = 0
```

```
for case in dataframe1.values:
    #if (counterCase == 5):
    year = case[0]
    midiA = directory + "/" + case[1] + ".mid"
    plaintiffTitle = case[2]
    plaintiffArtist = case[3]
    midiB = directory + "/" + case[4] + ".mid"
    suspectTitle = case[5]
    suspectArtist = case[6]
    result = case[7]

    print(plaintiffTitle + " vs. " + suspectTitle)
    print("Court Result: " + str(result))
    detector = plagiarismDetection(midiA, midiB)
    detectionResults.append([case[0], case[2], case[3], case[5], case[6], case[7], detector[0], detector[1]])
    print(str(detector[0]) + " - " + str(threshold))
    if (detector[0] > threshold) == result:
        correct = correct + 1
        if result == 1:
            truePosMelody = truePosMelody + 1
        else:
            trueNegMelody = trueNegMelody + 1
        print("Correct Melody Guess")
    else:
        if result == 1:
            falsePosMelody = falsePosMelody + 1
        else:
            falseNegMelody = falseNegMelody + 1
        print("Incorrect Melody Guess")
```

```
    print(str(detector[1]) + " - " + str(threshold))
    if (detector[1] > threshold) == result:
        correctRhythm = correctRhythm + 1
        if result == 1:
            truePosRhythm = truePosRhythm + 1
        else:
            trueNegRhythm = trueNegRhythm + 1
        print("Correct Rhythm Guess")
    else:
        if result == 1:
            falsePosRhythm = falsePosRhythm + 1
        else:
            falseNegRhythm = falseNegRhythm + 1
        print("Incorrect Rhythm Guess")
    print("-----------------SEPARATOR-----------------")
    #counterCase = counterCase + 1

print("-----------------SEPARATOR 2-----------------")
confusionMatrixMelody = confusionMatrix(truePosMelody, falsePosMelody, trueNegMelody, falseNegMelody)
confusionMatrixRhythm = confusionMatrix(truePosRhythm, falsePosRhythm, trueNegRhythm, falseNegRhythm)

accuracyMelody = confusionMatrixMelody[0]
accuracyRhythm = confusionMatrixRhythm[0]
precisionMelody = confusionMatrixMelody[1]
precisionRhythm = confusionMatrixRhythm[1]
recallMelody = confusionMatrixMelody[2]
recallRhythm = confusionMatrixRhythm[2]
fMeasureMelody = confusionMatrixMelody[3]
fMeasureRhythm = confusionMatrixRhythm[3]
```

```
print("===================MELODY========================")
print("Accuracy: " + str(round(accuracyMelody, 2)))
print("Precision: " + str(round(precisionMelody, 2)))
print("Recall: " + str(round(recallMelody, 2)))
print("F-Measure: " + str(round(fMeasureMelody, 2)))
print("===================RHYTHM========================")
print("Accuracy: " + str(round(accuracyRhythm, 2)))
print("Precision: " + str(round(precisionRhythm, 2)))
print("Recall: " + str(round(recallRhythm, 2)))
print("F-Measure: " + str(round(fMeasureRhythm, 2)))
```

Lampiran: Systematic Literature Review (SLR)

Lampiran berikut merupakan hasil tinjauan yang dilakukan terhadap paper – paper yang membawa topik deteksi plagiarisme musik.

<table>
<tr><th>No</th><th>Nama Jurnal</th><th>Metode</th><th>Hasil</th><th>Keterangan</th><th>Hasil Kuantitatif</th></tr>
<tr>
<td>1</td>
<td>A proposal to compare the similarity between musical products. (García et al., 2022)</td>
<td>Fuzzy-Ordered C-Means Clustering (FOCM)</td>
<td>Algoritma efektif dalam mendeteksi tingkat kesamaan musik.</td>
<td>FOCM dikembangkan sebagai pengembangan dari Fuzzy C-Means Clustering (FCM) untuk memasukkan susunan alamiah dari urutan set data dan susunan centroid sebagai faktor penting dalam algoritma. FOCM digunakan untuk menghitung ketidaksamaan, bukan kesamaan. Semakin tinggi angkanya, semakin tidak mirip sebuah musik dengan musik lainnya.</td>
<td>
Eksperimen No.1 (Someone Like you)
Versi yang digunakan:
official, brit, jordan, nursera, masha, imy2, leo
<table>
<tr><th></th><th>Versi</th><th>Versi</th><th>Nilai</th></tr>
<tr><td>Paling mirip</td><td>leo</td><td>imy2</td><td>5.405</td></tr>
<tr><td>Paling beda</td><td>nursera</td><td>leo</td><td>22.847</td></tr>
</table>
Eksperimen No.2 (When I was your man)
Versi yang digunakan:
bmo, bml, stewart, smith, imy2, scaccia
<table>
<tr><th></th><th>Versi</th><th>Versi</th><th>Nilai</th></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Percobaan telah dilakukan dengan menggunakan algoritma ini terhadap versi resmi sebuah lagu yang dibandingkan dengan versi cover. Mengingat versi resmi dan versi cover memiliki struktur yang sama dengan penyanyi yang berbeda, algoritma dapat mengidentifikasi lagu-lagu ini sebagai lagu yang mirip.</td>
<td>
<table>
<tr><td>Paling mirip</td><td>smith</td><td>bml</td><td>1.018</td></tr>
<tr><td>Paling beda</td><td>stewart</td><td>bmo</td><td>2.042</td></tr>
</table>
Eksperimen No.3 (All of me)
Versi yang digunakan:
jlo, jll, leroy, zogbi, scaccia, hoying
<table>
<tr><th></th><th>Versi</th><th>Versi</th><th>Nilai</th></tr>
<tr><td>Paling mirip</td><td>scaccia</td><td>jll</td><td>1.935</td></tr>
<tr><td>Paling beda</td><td>zogbi</td><td>leroy</td><td>3.196</td></tr>
</table>
</td>
</tr>
<tr>
<td>2</td>
<td>A Structural Similarity Index Based Method To Detect</td>
<td>Match Measure (MM) yang membandingkan similaritas birama</td>
<td>Efektif mendeteksi similaritas melodi musik,</td>
<td>Algoritma MM mengalami kesulitan dalam mengidentifikasi pola melodi ketika ada</td>
<td> Performa Algoritma MM dibanding dengan Algoritma Deterministik dan Recurrent Neural Network (RNN)
Dataset yang digunakan: </td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>Symbolic Monophonic Patterns In Real-Time (Silva & Turchet, 2022)</td>
<td>musik dalam bentuk gambar</td>
<td>melampaui metode Deterministik serta Recurrent Neural Network (RNN)</td>
<td>not yang dihapus atau durasinya berubah signifikan. Menggunakan Dynamic Window, deteksi melodi menjadi efisien. Akurasi algoritma 100% dalam set data sintetis yang tidak memiliki variasi</td>
<td>Rekaman Manusia; Sintetis dari sistem; Standar Industri Universitas Johannes Keppler (JKUPPD); Kombinasi dari ketiganya.
<table>
<tr><th></th><th></th><th>MM</th><th>Det.</th><th>RNN</th></tr>
<tr><td rowspan="4">Kombinasi</td><td>Akurasi</td><td>95.0%</td><td>53.6%</td><td>51.0%</td></tr>
<tr><td>Presisi</td><td>0.95</td><td>0.94</td><td>0.86</td></tr>
<tr><td>Recall</td><td>0.97</td><td>0.53</td><td>0.51</td></tr>
<tr><td>F-measure</td><td>0.96</td><td>0.68</td><td>0.64</td></tr>
<tr><td rowspan="4">Manusia</td><td>Akurasi</td><td>96.6%</td><td>56.0%</td><td>55.0%</td></tr>
<tr><td>Presisi</td><td>0.93</td><td>0.94</td><td>0.82</td></tr>
<tr><td>Recall</td><td>0.98</td><td>0.54</td><td>0.54</td></tr>
<tr><td>F-measure</td><td>0.96</td><td>0.69</td><td>0.65</td></tr>
<tr><td rowspan="4">Sintetis</td><td>Akurasi</td><td>100%</td><td>51.8%</td><td>49.4%</td></tr>
<tr><td>Presisi</td><td>0.97</td><td>0.95</td><td>0.89</td></tr>
<tr><td>Recall</td><td>0.98</td><td>0.51</td><td>0.49</td></tr>
<tr><td>F-measure</td><td>0.97</td><td>0.66</td><td>0.63</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td>
<table>
<tr><td rowspan="4">JKUPPD</td><td>Akurasi</td><td>70.5%</td><td>57.3%</td><td>50.7%</td></tr>
<tr><td>Presisi</td><td>0.95</td><td>0.90</td><td>0.76</td></tr>
<tr><td>Recall</td><td>0.97</td><td>0.61</td><td>0.59</td></tr>
<tr><td>F-measure</td><td>0.96</td><td>0.73</td><td>0.66</td></tr>
</table>
<table>
<tr><td></td><td>MM</td><td>Det.</td><td>RNN</td></tr>
<tr><td>Rata – rata Waktu Eksekusi (ms)</td><td>0.6</td><td>2.1</td><td>24.8</td></tr>
<tr><td>Rata – rata penggunaan memori (MB)</td><td>69</td><td>62</td><td>384</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>An adaptive meta-heuristic formusic plagiarism detection based on text</td><td>Kombinasi meta-heuristic yang terdiri atas gabungan metode text-based dengan clustering-based.</td><td>Algoritma memiliki akurasi yang lebih optimal dibanding dengan algoritma</td><td>Metode yang diusulkan masih berpotensi mengalami overfitting, berakibat kepada kinerja algoritma</td><td>Algoritma yang diuji:

Adaptive meta-heuristic (AM), Text similarity-based (TM), Clustering-based (CM), Fuzzy Vectorial (fuzzy), Tversky feature-based (tversky), Ukkonen (ukk), Edit distance (edit)</td></tr>
</table>

<table>
<tr>
<td></td>
<td>similarity and clustering (Malandrino et al., 2022)</td>
<td></td>
<td>seperti fuzzy-vectorial, tversky, ukkonen, edit distance, serta metode text-based dan clustering-based lainnya.</td>
<td>Overfitting dapat diatasi dengan memanfaatkan k-fold cross-validation untuk meningkatkan kemampuan generalisasi dari algoritma.

Selain algoritma, penelitian juga meneliti efektivitas deteksi plagiarisme menggunakan sistem. Sistem mampu membantu orang yang awam mendeteksi plagiarisme musik.</td>
<td>
Terhadap kasus plagiarisme (25 kasus)
<table>
<tr><th></th><th>Benar</th><th>Salah</th><th>Akurasi</th></tr>
<tr><td>AM</td><td>22</td><td>3</td><td>88%</td></tr>
<tr><td>TM</td><td>13</td><td>12</td><td>52%</td></tr>
<tr><td>CM</td><td>11</td><td>14</td><td>44%</td></tr>
<tr><td>fuzzy</td><td>16</td><td>9</td><td>64%</td></tr>
<tr><td>tversky</td><td>18</td><td>7</td><td>72%</td></tr>
<tr><td>Ukk</td><td>7</td><td>18</td><td>28%</td></tr>
<tr><td>edit</td><td>2</td><td>23</td><td>8%</td></tr>
</table>
Terhadap kasus non-plagiarisme (25 kasus)
<table>
<tr><th></th><th>Benar</th><th>Salah</th><th>Akurasi</th></tr>
<tr><td>AM</td><td>23</td><td>2</td><td>92%</td></tr>
<tr><td>TM</td><td>14</td><td>11</td><td>56%</td></tr>
<tr><td>CM</td><td>13</td><td>12</td><td>52%</td></tr>
<tr><td>fuzzy</td><td>12</td><td>13</td><td>48%</td></tr>
<tr><td>tversky</td><td>20</td><td>5</td><td>80%</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td><table><tr><td>Ukk</td><td>7</td><td>18</td><td>28%</td></tr><tr><td>edit</td><td>6</td><td>19</td><td>24%</td></tr></table></td></tr>
<tr><td>4</td><td>An Analysis of Melodic Plagiarism Recognition using Musical Similarity Algorithms (Schuitemaker et al., 2020)</td><td>-Edit Distance Algorithm

-Tversky.plaintiff.only

-Metode Komparatif</td><td>Algoritma yang diuji berpotensi menjadi sistem deteksi plagiarisme, namun masih belum matang untuk digunakan dalam industri</td><td>Penelitian memberikan aspek – aspek yang dapat diuji secara objective-based, seperti Harmoni, Ritme, Urutan not, Pergantian Kunci, Pergantian Mode, serta Remix musik.

Deteksi ritme dikecualikan pada penelitian dikarenakan kurangnya penelitian yang menggunakan algoritma menitikberat kepada ritme.</td><td>Akurasi dan Presisi tidak disediakan dikarenakan tidak signifikan.</td></tr>
<tr><td>5</td><td>An Evolutionary Multi-Objective</td><td>Metode Two-Objective Optimization</td><td>Pemilihan feature berdampak</td><td>Perluasan dataset untuk ekspansi limitasi, penerapan optimasi pada</td><td>Evaluasi dilakukan terhadap relevansi fitur – fitur berdasarkan Batasan seperti instrument, kunci, serta tempo. Tidak ada evaluasi akurasi pendeteksian.</td></tr>
</table>

	Feature Selection Approach for Detecting Music Segment Boundaries of Specific Types (Vatolkin et al., 2021)		signifikan kepada hasil klasifikasi Analisis terhadap aspek non-dominan penting untuk identifikasi feature yang paling relevan	skenario yang berbeda, serta evaluasi pendekatan terhadap algoritma segmentasi lain seperti jaringan saraf atau clustering perlu dilakukan. Pertimbangan model cadangan serta dan analisis minimasi kompromi juga penting untuk diteliti.	
6	Detecting Music Plagiarism Based on Melodic Analysis (Nguyen et al., 2023)	Kombinasi metode Edit Distance dengan Bipartite Graph Matching	Algoritma efisien dalam melakukan komputasi terhadap kemiripan musik.	Algoritma menggunakan nilai 0.6 sebagai batasan plagiarisme musik. Namun hasil dari algoritma belum sepenuhnya efektif mendeteksi kemiripan musik. Penelitian	**Performa Algoritma Berbasis Edit Distance dan N-Gram** **Nama Algoritma** / **Akurasi** / **Indeks** Basic Edit Distance / 46% / 2,875 Optimized Edit Distance / 60% / 2,336

selanjutnya dapat berupa penelitian menggunakan feature melodi lain, serta penggunaan musik jenis remix atau live play.

Korelasi TF-IDF	12%	34,22
Sum Common	14%	34,15
Ukkonen	13%	29,18
Tversky	13%	34,55

Performa k-Nearest Neighbor (KNN)

Nilai Batas	**Akurasi**	**Presisi**	**Recall**
0.5	68,36%	0,88	0,4
0.6	92,4%	1	0,58
0.7	90,23%	0,5	0,27

Performa Support Vector Model (SVM)

Nilai Batas	**Kernel SVM**	**Akurasi**	**Presisi**	**Recall**
0,5	rbf	80,35%	0,79	0,79
	poly	67,37%	0,75	0,46

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td><table><tr><td rowspan="2">0,6</td><td>rbf</td><td>89,24%</td><td>1</td><td>0,43</td></tr><tr><td>poly</td><td>87,08%</td><td>0,9</td><td>0,37</td></tr><tr><td rowspan="2">0,7</td><td>rbf</td><td>89,18%</td><td>0,2</td><td>0,1</td></tr><tr><td>poly</td><td>92,46%</td><td>0,8</td><td>0,4</td></tr></table></td></tr>
<tr><td>7</td><td>Detecting Text Reuse and Similarities between Artists in Rap Music through Visualization (Meinecke & Jänicke, 2021)</td><td>fastText model untuk deteksi similaritas terhadap data teks secara kontekstual</td><td>Algoritma mampu mendeteksi style penulisan lirik rap artis dan menentukan korelasi terhadap seorang penulis lirik yang sama</td><td>Penelitian ini tidak berfokus terhadap deteksi plagiarisme musik, melainkan terhadap pencarian karya musik yang ditulis oleh pembuat lirik yang sama dan melakukan rekomendasi musik.</td><td>Hasil evaluasi bukan dalam bentuk akurasi kemiripan, melainkan dalam bentuk rekomendasi nama artis berdasarkan lirik lagu.</td></tr>
<tr><td>8</td><td>E3MSD: A New Music Information Retrieval Architecture for an Original</td><td>Music Definition Language (MDL) dan Music Manipulation Language (MML) sebagai model</td><td>E3MSD menunjukkan hasil yang reliabel, berpotensi digunakan sebagai sistem</td><td>MDL melakukan pemrosesan audio dengan menggunakan skema koding yang simbolis, sementara MML melakukan pemrosesan langsung terhadap</td><td>Implementasi MDL&MML dengan Weighted Similarity terhadap melodi dan ritme menghasilkan akurasi 84%.

Implementasi MDL&MML dengan tambahan berupa Ensemble Learning berbasis Reinforced</td></tr>
</table>

<table>
<tr><td></td><td>Music Identifier (Li et al., 2020)</td><td></td><td>untuk deteksi plagiarisme secara online maupun offline.</td><td>gelombang audio musik tanpa merusak aliran dari melodi.

Penelitian selanjutnya dapat berupa implementasi fungsi stokastik sehingga memperhitungkan feature harmoni musik. Untuk penggunaan industri, penelitian terhadap musik remix juga dapat dilakukan.</td><td>Learning meningkatkan akurasi deteksi sampai 96%.</td></tr>
<tr><td>9</td><td>Enhanced Plagiarism Detection Through Advanced Natural</td><td>NLP dengan metode Deep NLP berupa word2vec, dengan feature extraction E-BERT, dan klasifikasi k-means Clustering.</td><td>Akurasi 95.5% ketika mendeteksi plagiarisme lintas bahasa.</td><td>Penelitian ini tidak menggunakan musik, namun berfokus terhadap deteksi plagiarisme akademis.</td><td>(Plagiarisme Akademis)

Performa Algoritma yang dirancang penulis

<table><tr><th>Algoritma</th><th>Akurasi</th><th>Presisi</th><th>Recall</th><th>F-Score</th></tr><tr><td>Rancangan</td><td>95,5%</td><td>77,75%</td><td>78,67%</td><td>78,21%</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td>Language Processing (Antonius et al., 2023)</td><td>Algoritma Smith-Waterman.</td><td></td><td>Enhanced Bidirectional Encoder Representations from Transformers (E-BERT) menggunakan vektor per kata serta posisinya untuk menentukan konteks semantik dalam kalimat.

Smith-Waterman digunakan untuk melakukan translasi bahasa. Dalam penelitian ini, translasi bahasa Inggris ke bahasa Spanyol dibantu dengan kamus Inggris-Spanyol.</td><td><table>
<tr><td>Word2vec + CNN</td><td>91,18%</td><td>59,77%</td><td>58,51%</td><td>59,13%</td></tr>
<tr><td>Doc2vec + LR</td><td>89,27%</td><td>51,06%</td><td>50,67%</td><td>50,87%</td></tr>
<tr><td>One-hot + LR</td><td>88,82%</td><td>49,19%</td><td>48,46%</td><td>48,82%</td></tr>
</table></td></tr>
<tr><td>10</td><td>Fine-Grained Music</td><td>Bipartite Graph Matching dibantu</td><td>Algoritma dapat mengkuantifikasi</td><td></td><td></td></tr>
</table>

<table>
<tr>
<td></td>
<td>Plagiarism Detection (Liu et al., 2023)</td>
<td>dengan Melody Sequence Representation

Algoritma Kuhn-Munkres untuk minimum weight perfect matching.</td>
<td>tingkat plagiarisme musik</td>
<td>Algoritma dikembangkan dengan tujuan analisis similaritas musik secara rinci. Bipartite Graph digunakan untuk menggambarkan musik sebagai node “kiri” dan “kanan”. Matching dilakukan untuk mencari bagian yang non-overlap. Semakin tinggi nilai yang dihasilkan, semakin mirip satu musik terhadap muasik yang diuji.

Algoritma Kuhn-Munkres digunakan untuk mencari nilai weight minimum yang</td>
<td>Performa BMM-Deterministik dibanding dengan algoritma lainnya
<table>
<tr><th></th><th colspan="2">Data MPD</th><th colspan="2">Data Riil</th></tr>
<tr><th></th><th>Indeks rata - rata</th><th>Acc</th><th>Indeks rata - rata</th><th>Acc</th></tr>
<tr><td>TF-IDF</td><td>10,52</td><td>69,5%</td><td>3,24</td><td>65,5%</td></tr>
<tr><td>Tversky</td><td>16,37</td><td>68,5%</td><td>3,14</td><td>65,5%</td></tr>
<tr><td>Sum Common</td><td>12,00</td><td>65,3%</td><td>3,10</td><td>72,4%</td></tr>
<tr><td>Ukkonen</td><td>10,08</td><td>67,5%</td><td>3,00</td><td>75,9%</td></tr>
<tr><td>BMM-Det.</td><td>7,07</td><td>70,5%</td><td>2,17</td><td>82,8%</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td>digunakan untuk optimalisasi algoritma.</td><td></td></tr>
<tr><td>11</td><td>Identification and Detection of Plagiarism in Music using Machine Learning Algorithms (Ramachandran Nair, 2021)</td><td>- Gaussian Naïve Bayes
- Logistic Regression
- Random Forest Classifier
- Decision Tree
- k-Nearest Neighbor</td><td>Empat dari lima model yang diuji berhasil mendeteksi plagiarisme dengan baik
Random Forest Classifier konsisten dalam mendeteksi plagiarisme dengan akurasi 98%
Naïve Bayes memiliki akurasi 22%</td><td>Dataset menggunakan file MIDI dari lagu yang akan diidentifikasi. File MIDI dikumpulkan dari hasil transkripsi penggemar terhadap lagu. Ekstraksi melodi dilakukan dengan cara reduksi harmoni, sehingga meninggalkan melodi atau progresi chordnya.

Nilai 0.5 ditentukan sebagai threshold nilai apakah sebuah musik plagiat atau tidak. Akurasi model diambil</td><td>Performa Algoritma Machine Learning dalam deteksi plagiarisme musik
<table>
<tr><th rowspan="2">Algoritma</th><th colspan="4">Nilai Batas</th></tr>
<tr><th>0,5</th><th>0,6</th><th>0,7</th><th>0,8</th></tr>
<tr><td>Naïve Bayes</td><td>22%</td><td>31%</td><td>41%</td><td>62%</td></tr>
<tr><td>Logistic Regression</td><td>91%</td><td>83%</td><td>74%</td><td>63%</td></tr>
<tr><td>Decision Tree</td><td>91%</td><td>84%</td><td>73%</td><td>59%</td></tr>
<tr><td>Random Forest</td><td>98%</td><td>97%</td><td>95%</td><td>93%</td></tr>
<tr><td>KNN</td><td>95%</td><td>92%</td><td>89%</td><td>86%</td></tr>
</table></td></tr>
</table>

				dengan patokan nilai tersebut terhadap nilai data yang dideteksi. Untuk penelitian selanjutnya dapat dilakukan terhadap MIDI dengan multitrack, meningkatkan akurasi dedteksi melodi, Menghitung menggunakan nilai threshold yang lebih besar atau lebih kecil, serta mengumpulkan data lebih banyak untuk meningkatkan akurasi deteksi.	

12	Methods of Automated Music Comparison Based on Multi-Objective Metrics of Network Similarity (Muszynski & Tarapata, 2023)	-Distance calculation Hamming dan Levenshtein -Graph representasi musik dalam matrix -Implementasi weighted average terhadap matrix	-Algoritma mampu mendeteksi kemiripan musik -Algoritma belum dapat mendeteksi harmoni dalam musik -Algoritma belum bisa membedakan musik bervokal dan musik instrumental	Pendekatan Hamming dan Levenshtein merupakan pendekatan yang digunakan untuk mendeteksi kemiripan pada data teks. Penelitian ini melakukan translasi pendekatan ini yang sebelumnya menggunakan huruf menjadi satu birama musik. Distance calculation diimplementasi di dalam matrix yang terdiri atas feature musik, membentuk sebuah graph. Semakin mirip suatu musik, maka hasil	Hasil Penelitian tidak menyediakan metode evaluasi performa algoritma. Hasil Penelitian hanya menyediakan nilai hasil kemiripan antar musiknya.

				kalkulasi ini akan semakin kecil antar titiknya. Weighted average digunakan untuk menggabungkan kriteria – kriteria yang ada menjadi satu kriteria yang maksimal. Algoritma ini tidak menggunakan representasi musik seperti pada MIDI	
13	Music Plagiarism Detection Based on Siamese	-Vektorisasi menggunakan Siamese CNN	-Algoritma memiliki akurasi 98.7% dalam identifikasi	Algoritma ini dinamakan Music Plagiarism Detector for Single (MPD-S) karena	**MPD-S menggunakan CNN** <table><tr><th>Model CNN</th><th>Akurasi</th></tr><tr><td>SimpleCNN</td><td>98,70%</td></tr></table>

<table>
<tr>
<td></td>
<td>CNN (Park et al., 2022)</td>
<td>- Imaging, translasi vector menjadi gambar

- Deteksi kemiripan gambar menggunakan Siamese CNN</td>
<td>kemiripan melodi.</td>
<td>mendeteksi kemiripan dari dua lagu yang berbeda.

Algoritma efektif dalam identifikasi melodi, namun mengalami kendala dalam deteksi plagiarisme musik secara utuh, tidak mempertimbangkan struktur musik.</td>
<td>
<table>
<tr><td>ResNet-50</td><td>95,65%</td></tr>
<tr><td>ResNeXt-50</td><td>93,95%</td></tr>
<tr><td>EfficientNet</td><td>31,94%</td></tr>
</table>
MPD-S menggunakan perbandingan berbasis teks
<table>
<tr><th>Algoritma</th><th>Jenis data</th><th>Akurasi</th></tr>
<tr><td>Sum Common</td><td>Vektor</td><td>39,91%</td></tr>
<tr><td>Ukkonen</td><td>Vektor</td><td>46,07%</td></tr>
<tr><td>Tversky</td><td>Vektor</td><td>46,58%</td></tr>
<tr><td>Edit distance</td><td>Vektor</td><td>76,03%</td></tr>
<tr><td>MPD-S</td><td>Gambar</td><td>98,70%</td></tr>
</table>
Performa MPD-S berdasarkan jenis dataset
<table>
<tr><th>Dataset</th><th>Jumlah data musik</th><th>Jumlah data dekomposisi</th><th>Akurasi</th></tr>
<tr><td>POP909</td><td>909</td><td>1285</td><td>94,79%</td></tr>
<tr><td>Musedata</td><td>439</td><td>615</td><td>99,68%</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td><table><tr><td>MTC-LC-1.0</td><td>4830</td><td>3690</td><td>99,21%</td></tr><tr><td>MTC-ANN-2.01</td><td>360</td><td>200</td><td>99%</td></tr><tr><td>MTC-FS-1.0</td><td>4120</td><td>2526</td><td>99,05%</td></tr></table></td></tr>
<tr><td>14</td><td>Music Plagiarism Detector (James, 2019)</td><td>-Algoritma Fingerprinting menggunakan MFCC, Entropy, serta Mean-Energy Level

-Euclidean Distance untuk komparasi masing – masing karya.</td><td>Hasil berupa framework untuk deteksi plagiarisme musik.</td><td>Algoritma yang digagas oleh penelitian ini belum diuji terhadap data plagiarisme musik, sehingga penelitian selanjutnya dapat berupa pengujian terhadap algoritma ini.</td><td>Paper ini tidak menyediakan hasil dan evaluasi, melainkan hanya memberikan gagasan metode saja.</td></tr>
<tr><td>15</td><td>Music Similarity Estimation based on Feature</td><td>-Deep Learning

-Siamese Neural Network untuk audio processing</td><td>Hasil algoritma efektif untuk mendeteksi kemiripan musik, namun kurang</td><td>Dalam penelitian ini, Langkah metode dipangkas dengan cara menggunakan data fitur musik secara langsung</td><td>Hasil evaluasi menunjukkan akurasi 90%, namun penulis memberikan catatan bahwa data yang digunakan untuk pengujian jumlahnya berat sebelah, sehingga tidak merepresentasikan keakuratan model secara baik.</td></tr>
</table>

	Extraction and Deep Learning (Niu, 2023)		efektif untuk menggantikan keputusan secara manual.	untuk melatih model, sehingga meningkatkan efisiensi. Model ini tidak cukup akurat untuk menjadi sistem deteksi plagiarisme musik, tetapi memiliki aplikasi dalam sistem rekomendasi musik yang serupa. Untuk ke depannya, dataset dapat dibangun berdasarkan genre sebagai spesialisasi algoritma.	
16	Perceptual and automated estimates of infringement in 40 music	Survey terhadap orang awam mengenai kemiripan musik menggunakan	Deteksi plagiarisme musik secara manual lebih	Data persepsi dikumpulkan dari 51 partisipan untuk 40 kasus hak cipta yang telah terjadi dari tahun 1915-	Hasil evaluasi dari penelitian ini adalah, akurasi pendeteksian plagiarisme musik secara automated memiliki akurasi 75%. Sementara akurasi pendeteksian plagiarisme musik secara manual memiliki akurasi 83%.

	copyright cases (Yuan et al., 2023)	kasus plagiarisme yang telah terjadi. Data musik menggunakan audio lengkap, melodi (MIDI), serta lirik (teks)	efektif dibanding secara otomatis.	2018 di 7 negara beryurisdiksi hukum berbeda (Amerika Serikat, Inggris, Australia, Selandia Baru, Jepang, Republik Rakyat Tiongkok, dan Taiwan). Peserta yang mendengarkan audio lengkap memiliki hasil akurasi yang lebih tinggi dibanding algoritma deteksi plagiarisme musik dengan akurasi maksimal masing-masing 83% vs 75%. Ini menunjukkan bahwa musik, lirik, serta faktor kontekstual lainnya dapat	

<table>
<tr><td></td><td></td><td></td><td></td><td>mempengaruhi satu sama lain dengan tingkat pengaruh yang berbeda - beda.</td><td></td></tr>
<tr><td>17</td><td>TruMuzic: A Deep Learning and Data Provenance-Based Approach to Evaluating the Authenticity of Music (Gurjar et al., 2023)</td><td>Deep Learning untuk mengidentifikasi similaritas.

Regresi Linear untuk mendapatkan weight learning yang optimal.</td><td>Algoritma menghasilkan akurasi 10% lebih baik dibanding dengan penelitian yang sudah ada sebelumnya.</td><td>Algoritma TruMuzic merupakan algoritma yang menggunakan Deep Learning sebagai mekanisme utama dengan dibantu data musik beserta sumbernya. Untuk mendapatkan weighting algoritma yang optimal, digunakan regresi linear. Akurasi algoritma diuji terhadap 3800 data musik.</td><td>Performa TruMuzic dibanding algoritma lainnya
<table>
<tr><th>Algoritma</th><th>Akurasi</th></tr>
<tr><td>Edit Distance</td><td>44%</td></tr>
<tr><td>Model Silva</td><td>53%</td></tr>
<tr><td>SiMPle</td><td>67%</td></tr>
<tr><td>Model Borkar</td><td>72%</td></tr>
<tr><td>MESMF</td><td>75%</td></tr>
<tr><td>TruMuzic (tanpa Deep Learning)</td><td>61%</td></tr>
<tr><td>TruMuzic (dengan Deep Learning)</td><td>85%</td></tr>
</table></td></tr>
</table>

<table>
<tr>
<td>18</td>
<td>Applicability of Similarity Coefficients in Social Circle Matching (Korepanova et al., 2020)</td>
<td>Komparasi antara berbagai macam algoritma kemiripan</td>
<td>Algoritma Szymkiewicz-Simpson memiliki performa yang paling baik dibanding algoritma kemiripan lainnya</td>
<td>Algoritma Szymkiewicz-Simpson memiliki akurasi yang kurang lebih seimbang dibanding algoritma Sorensen-Dice, Kulczynski, dan Otsuka-Ochiai. Namun Szymkiewicz-Simpson memiliki akurasi yang lebih tinggi dibanding ketiganya.</td>
<td>Performa antar berbagai algoritma kemiripan
<table>
<tr><th>Algoritma Kemiripan</th><th>Akurasi</th></tr>
<tr><td>Jaccard</td><td>84%</td></tr>
<tr><td>Sorensen-Dice</td><td>90%</td></tr>
<tr><td>Kulczynski</td><td>91%</td></tr>
<tr><td>Otsuka-Ochiai</td><td>90%</td></tr>
<tr><td>Szymkiewicz-Simpson</td><td>92%</td></tr>
<tr><td>Braun-Blanquet</td><td>86%</td></tr>
</table>
</td>
</tr>
</table>

Berdasarkan literature review yang telah dilakukan, dapat dilihat bahwa metode yang digunakan jarang berkaitan satu sama lain. Penelitian yang berkaitan satu sama lain umumnya memiliki sebuah karakteristik yang menonjol, yaitu penggunaan algoritma vector-based seperti Edit Distance, Ukkonen, atau Tversky.

Metode – metode yang digunakan dalam penelitian – penelitian di atas juga memiliki sebuah fitur yang umum digunakan sebagai data utama yang diproses, yaitu melodi. Cara mendapatkan melodi dari sebuah musik memiliki beberapa cara, seperti dengan mengambil informasi dari file MIDI, melakukan ekstraksi harmoni dari file musik secara langsung, image processing dari vektor yang telah ditranslasi menjadi sebuah gambar, atau dengan memprosesnya secara langsung menggunakan Deep Learning. Oleh karena itu, melodi menjadi fitur utama yang akan digunakan dalam proses deteksi plagiarisme musik ini.

Selain melodi, terdapat pula fitur yang dimiliki oleh musik, namun penelitian – penelitian pada umumnya mengecualikan fitur ini. Fitur ini adalah fitur ritme dari musik. Menurut (Schuitemaker et al., 2020), fitur ritme dikecualikan dikarenakan kurangnya kontribusi penelitian yang menggunakan fitur ini. Namun berdasarkan (Park et al., 2022), struktur musik dapat berperan penting dalam menentukan plagiarisme musik.

Berdasarkan penelitian oleh (Ramachandran Nair, 2021), dengan reduksi harmoni menjadi melodi yang kemudian diolah menggunakan algoritma Random Forest, didapat akurasi sampai 98%. Namun, data yang digunakan bukanlah dari kasus – kasus plagiarisme yang terjadi sebelumnya. Data yang digunakan adalah data yang dikumpulkan penulis dengan karakteristik musik yang kurang lebih sama sehingga terdapat bias yang mempengaruhi performa algoritma.