Optimasi Algoritma Cosine Similarity dalam Pencarian Audio MIDI Serupa

Jovandra Otniel P. S. - 13523141[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]jovandra.siregar@gmail.com, [2]13523141@std.stei.itb.ac.id

***Abstract*— Audio similarity search is a challenging problem in the fields of digital signal processing and machine learning, especially when dealing with large datasets. Traditional similarity measurement methods, such as cosine similarity, are widely used but often computationally expensive in their conventional implementations. This study explores the optimization of similarity measurement algorithms using advanced matrix operations and data storage techniques. By leveraging efficient libraries and compressed data formats, the approach aims to improve computation speed while maintaining accuracy. The findings highlight significant improvements in processing efficiency, offering practical solutions for applications in music and multimedia.**

***Keywords*— audio similarity, cosine similarity, optimization, signal processing, computational efficiency.**

I. PENDAHULUAN

Perkembangan teknologi digital telah membawa perubahan signifikan dalam berbagai bidang, termasuk industri musik dan multimedia. Salah satu tantangan utama dalam bidang pemrosesan sinyal digital adalah pencarian audio MIDI (Musical Instrument Digital Interface) serupa, terutama ketika menghadapi dataset yang besar. Pencarian ini memerlukan algoritma yang efisien untuk mengidentifikasi dan membandingkan kemiripan antar data audio secara akurat dan cepat.

Gambar 1.1 Aplikasi Shazam Pencari Lagu Berdasarkan Kueri Audio
(sumber: www.shazam.com)

Cosine similarity merupakan salah satu metode yang sering digunakan untuk mengukur kemiripan antara dua vektor fitur audio. Metode ini efektif dalam membedakan karakteristik audio berdasarkan orientasi vektor dalam ruang fitur. Namun, implementasi konvensional dari cosine similarity cenderung memakan waktu komputasi yang tinggi, terutama ketika diterapkan pada dataset yang besar dan kompleks seperti MIDI.
Penelitian ini bertujuan untuk mengoptimalkan algoritma cosine similarity dengan memanfaatkan library NumPy dan format file .npz. NumPy menyediakan operasi matriks yang dioptimalkan, sementara format .npz memungkinkan penyimpanan vektor fitur audio dalam format yang terkompresi. Kombinasi kedua teknologi ini diharapkan dapat mempercepat proses perhitungan kemiripan tanpa mengorbankan akurasi pengenalan audio.

Optimasi algoritma cosine similarity ini tidak hanya meningkatkan efisiensi komputasi tetapi juga memberikan solusi yang lebih andal untuk aplikasi musik dan multimedia. Dengan demikian, penelitian ini dapat memberikan kontribusi signifikan dalam pengembangan sistem pencarian audio yang lebih responsif dan efektif.

Struktur makalah ini dimulai dengan pendahuluan yang menjelaskan latar belakang dan tujuan penelitian. Bab berikutnya membahas landasan teori yang mendasari penelitian ini, termasuk algoritma vektor dan matriks yang relevan dengan pencarian audio serupa. Selanjutnya, metodologi penelitian, hasil dan pembahasan, serta kesimpulan dan saran akan dibahas secara mendetail.

II. LANDASAN TEORI

A. Similaritas Kosinus

Cosine similarity adalah metode yang digunakan untuk mengukur tingkat kemiripan antara dua vektor dalam ruang vektor multidimensi [7]. Dalam konteks pencarian audio MIDI, setiap file MIDI direpresentasikan sebagai vektor fitur yang menggambarkan karakteristik musiknya. Cosine similarity dihitung menggunakan rumus berikut:

$$\cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$$

Di mana A dan B merupakan vektor dua file MIDI yang dibandingkan, dan $\cos\theta$ merupakan besaran kemiripan file MIDI yang bernilai dari -1 hingga 1. Nilai 1 menunjukkan bahwa kedua file MIDI memiliki kemiripan yang tinggi.

B. Representasi Vektor dalam Pencarian Audio MIDI

Representasi vektor adalah tahap penting dalam pengolahan data audio MIDI. Setiap file MIDI diubah menjadi serangkaian fitur numerik yang dapat diproses oleh algoritma pembelajaran mesin. Proses ini melibatkan ekstraksi informasi musik yang relevan, seperti notasi

musik, durasi, dan instrumen yang digunakan. Dalam algoritma yang diimplementasikan, tiga jenis fitur utama diekstraksi: Distribusi Nada Absolut (ATB), Distribusi Nada Relatif (RTB), dan Distribusi Nada Pertama (FTB).

Distribusi Nada Absolut (ATB) menghitung distribusi absolut nada dalam jendela waktu tertentu, memberikan informasi tentang frekuensi kemunculan setiap not dalam segmen waktu tersebut. ATB direpresentasikan sebagai vektor dengan banyaknya komponen 128, sesuai dengan rentang MIDI note number (0-127). ATB merupakan histogram ternormalisasi H_{ATB} di mana setiap n berkorespondensi dengan nomor not MIDI dari 0 sampai dengan 127.

$$H_{ATB}(n) = \frac{count((note)_n)}{\sum_{k=0}^{127} count((note)_k)}$$

Di mana $n \in \mathbb{Z}$ dan $0 \leq n \leq 127$. Pada rumus ini, $count((note)_n)$ adalah jumlah kemunculan not MIDI nomor n dalam jendela waktu tertentu, dan $\sum_{k=0}^{127} count((note)_k)$ adalah total jumlah not yang muncul dalam jendela tersebut. Normalisasi memastikan bahwa nilai histogram berada dalam rentang [0,1], memungkinkan perbandingan yang konsisten antar jendela.

Distribusi Nada Relatif (RTB) menghitung distribusi perubahan relatif antara nada berturut-turut, menangkap interval yang terjadi dalam melodi dan memberikan wawasan tentang dinamika serta gerakan melodi. RTB direpresentasikan sebagai vektor dengan 255 komponen, mencakup interval dari -127 hingga 127. Relative Tone Bag (RTB) adalah histogram ter-normalisasi H_{RTB} di mana setiap bin merepresentasikan interval pitch yang mungkin antara dua nada berurutan, berkisar dari -127 hingga +127 semiton.

$$H_{RTB}(\delta) = \frac{count(\Delta pitch_\delta)}{\sum_{\delta'=-127}^{127} count(\Delta pitch_{\delta'})}$$

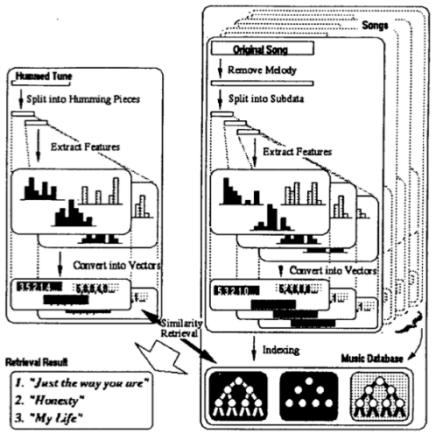
Di mana $\delta \in \mathbb{Z}$ dan $-127 \leq \delta \leq 127$. Dalam rumus ini, $\Delta pitch$ adalah perbedaan pitch antara dua nada berturut-turut dalam melodi. $count(\Delta pitch_\delta)$ adalah jumlah kemunculan interval pitch δ, dan $\sum_{\delta'=-127}^{127} count(\Delta pitch_{\delta'})$ adalah total jumlah interval yang dihitung dalam jendela tersebut. Normalisasi ini memungkinkan perbandingan distribusi interval antar jendela dengan skala yang seragam.

Distribusi Nada Pertama (FTB) menghitung distribusi nada relatif terhadap nada pertama dalam jendela waktu, menyoroti perubahan nada yang terjadi relatif terhadap not awal, yang penting untuk memahami struktur harmonik dan motif dalam melodi. FTB juga direpresentasikan sebagai vektor dengan 255 komponen, mencakup interval dari -127 hingga 127. First Tone Bag (FTB) adalah histogram ter-normalisasi H_{FTB} di mana setiap bin (komponen vektor) merepresentasikan selisih antara setiap nada dalam jendela dengan nada pertama.

$$H_{FTB}(\delta) = \frac{count((note_i - note_1)_\delta)}{\sum_{\delta'=-127}^{127} count((note_i - note_1)_{\delta'})}$$

Di mana $\delta \in \mathbb{Z}$ dan $-127 \leq \delta \leq 127$. Pada rumus ini, $note_i$ adalah setiap not dalam jendela, dan $note_1$ adalah not pertama dalam jendela tersebut. $count((note_i - note_1)_\delta)$ adalah jumlah kemunculan selisih δ antara setiap not i dengan not pertama 1, sementara $\sum_{\delta'=-127}^{127} count((note_i - note_1)_{\delta'})$ adalah total jumlah selisih yang dihitung dalam jendela tersebut. Normalisasi memastikan bahwa distribusi selisih nada berada dalam rentang [0, 1], memungkinkan analisis yang konsisten terhadap perubahan relatif nada dalam melodi [1].

C. Ekstraksi Vektor-vektor (fitur) dari File MIDI dan Pencocokan MIDI serupa



Gambar 2.1 Proses Pencocokan audio antara satu file MIDI (query) dengan file MIDI yang lain
(sumber: informatika.stei.itb.ac.id)

Ekstraksi fitur dari file MIDI adalah proses yang mengubah data MIDI mentah menjadi representasi vektor yang dapat dianalisis lebih lanjut untuk tujuan pencarian kemiripan. Proses ini dimulai dengan pembacaan file MIDI menggunakan *library* mido, yang memungkinkan akses ke pesan-pesan musik seperti *note_on* dan *note_off*. Pesan *note_on* dengan *velocity* > 0 menandakan bahwa sebuah nada dimulai, sementara *note_off* atau *note_on* dengan *velocity* = 0 menandakan bahwa nada tersebut dihentikan. Fokus utama dalam ekstraksi melodi adalah pada pesan-pesan *note_on* yang relevan, mengabaikan not dengan *velocity* nol untuk mendapatkan rangkaian not yang bersih dan representatif dari melodi utama.

Setelah melodi diekstraksi, langkah selanjutnya adalah melakukan *windowing*, yaitu membagi urutan not yang telah diekstrak menjadi jendela-jendela waktu tertentu menggunakan metode *sliding window*. Misalnya, dengan ukuran jendela 40 nada dan langkah pergeseran 6 nada, melodi dibagi menjadi segmen yang tumpang tindih. Teknik ini memungkinkan analisis fitur dalam segmen

waktu yang lebih kecil dan terfokus, meningkatkan kemampuan algoritma untuk menangkap pola-pola lokal dalam melodi.

Dalam setiap jendela melodi yang telah dibagi, fitur-fitur ATB, RTB, dan FTB diekstraksi. Distribusi Nada Absolut (ATB) dihitung sebagai histogram frekuensi setiap not dalam jendela, yang kemudian dinormalisasi sehingga jumlah seluruh frekuensi menjadi 1. Distribusi Nada Relatif (RTB) dihitung sebagai histogram dari perbedaan nada antara not berturut-turut dalam jendela, juga dinormalisasi. Distribusi Nada Pertama (FTB) dihitung sebagai histogram dari perbedaan setiap not dengan not pertama dalam jendela, dan dinormalisasi dengan cara yang sama. Proses normalisasi ini memastikan bahwa distribusi nada dapat dibandingkan secara konsisten antar file MIDI, tanpa dipengaruhi oleh jumlah not dalam setiap jendela. Hasilnya, didapat $128 + 255 + 255 = 638$ dimensi vektor fitur dari masing-masing file MIDI. Vektor tersebut disimpan dalam 3 buah senarai (*list*), masing-masing merepresentasikan ATB, RTB, dan FTB.

.

D. Mekanisme Pencarian Audio MIDI Serupa

Algoritma pencarian MIDI serupa yang diimplementasikan dalam penelitian ini melibatkan beberapa langkah utama, mulai dari pemuatan file MIDI hingga pengembalian hasil pencarian berdasarkan tingkat kemiripan yang dihitung menggunakan cosine similarity.

Proses dimulai dengan pemuatan file MIDI dari direktori database. Setiap file MIDI dibaca dan diekstrak melodi utamanya menggunakan library mido. Melodi yang diekstraksi kemudian dibagi menjadi jendela-jendela waktu tertentu menggunakan metode sliding window. Dari setiap jendela, fitur-fitur ATB, RTB, dan FTB diekstraksi dan dinormalisasi. Hasil ekstraksi ini disimpan dalam format terkompresi .npz untuk efisiensi penyimpanan dan akses cepat.

Ketika pengguna mengunggah file MIDI sebagai query, proses pencarian serupa dimulai dengan memuat file MIDI query dan mengekstrak melodi utamanya. Melodi query kemudian dibagi menjadi jendela-jendela waktu tertentu, dan dari setiap jendela, fitur-fitur ATB, RTB, dan FTB diekstraksi serta dinormalisasi. Selanjutnya, similarity antara fitur query dan fitur database dihitung menggunakan cosine similarity untuk menentukan tingkat kemiripan antara query dan seluruh dataset MIDI.

Fungsi calculate_similarity merupakan inti dari algoritma pencarian kemiripan, yang menghitung tingkat kemiripan keseluruhan antara query dan database berdasarkan fitur-fitur ATB, RTB, dan FTB menggunakan cosine similarity. Proses ini melibatkan perhitungan similarity untuk setiap tipe fitur secara terpisah, kemudian menggabungkannya dengan bobot tertentu untuk mendapatkan nilai similarity total. Fungsi search_midi bertanggung jawab untuk mencari file MIDI dalam database yang memiliki similarity di atas ambang batas tertentu, menyortir hasil berdasarkan similarity tertinggi, dan mengembalikan daftar hasil yang paling mirip dengan query.

E. Optimasi dengan NumPy dan Format .npz

Gambar 2.2 Pustaka NumPy
(sumber:www.freecodecamp.org)

Optimasi algoritma cosine similarity dalam pencarian audio MIDI serupa merupakan aspek penting untuk memastikan efisiensi dan kecepatan sistem, terutama saat menangani dataset yang besar. Dua pendekatan utama yang digunakan untuk mencapai optimasi ini adalah melalui pemanfaatan library NumPy dan penggunaan format penyimpanan .npz. Kedua pendekatan ini saling melengkapi dalam meningkatkan performa sistem, baik dari segi komputasi maupun penyimpanan data.

NumPy adalah library fundamental dalam ekosistem Python yang dirancang untuk komputasi ilmiah, menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi. Dalam konteks pencarian kemiripan MIDI, NumPy dimanfaatkan untuk melakukan operasi vektorisasi pada matriks dan vektor, yang secara signifikan mempercepat proses komputasi dibandingkan dengan implementasi berbasis kalang (*loop*) tradisional. Operasi vektorisasi memungkinkan eksekusi simultan pada seluruh elemen array tanpa memerlukan iterasi eksplisit dalam kode Python. Sebagai contoh, perkalian matriks yang diperlukan untuk menghitung cosine similarity dapat dilakukan dengan satu panggilan fungsi np.dot, yang jauh lebih efisien dibandingkan dengan melakukan perkalian elemen per elemen menggunakan loop [2].

Perhitungan norma vektor merupakan langkah penting dalam menghitung cosine similarity. Dengan menggunakan fungsi np.linalg.norm, norma vektor dapat dihitung secara efisien untuk seluruh matriks fitur sekaligus. Fungsi ini dioptimalkan untuk memanfaatkan arsitektur komputer modern, termasuk penggunaan SIMD (Single Instruction, Multiple Data) dan cache memory, yang secara signifikan mempercepat proses perhitungan norma vektor. Selain itu, NumPy menyediakan manajemen memori yang efisien melalui penggunaan array kontigu yang dioptimalkan untuk akses cepat dan penggunaan memori yang minimal. Hal ini sangat penting ketika bekerja dengan dataset besar, di mana penggunaan memori yang tidak efisien dapat menyebabkan penurunan performa atau bahkan kegagalan sistem. Dengan memanfaatkan struktur data NumPy, sistem dapat menangani volume data yang besar tanpa mengorbankan kecepatan atau stabilitas [3].

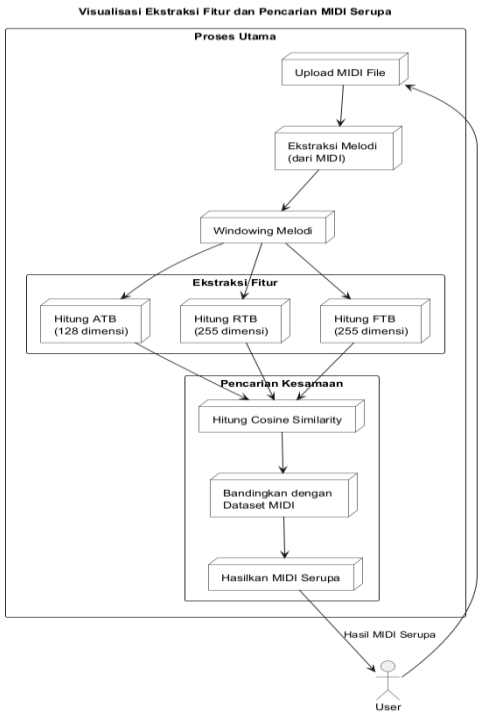
Operasi matriks seperti perkalian dan dekomposisi, yang esensial dalam teknik Principal Component Analysis (PCA) dan perhitungan cosine similarity, juga

dioptimalkan dalam NumPy. Fungsi-fungsi ini diimplementasikan menggunakan algoritma yang telah teruji dan dioptimalkan untuk performa tinggi, memastikan bahwa perhitungan dapat dilakukan dengan cepat dan akurat.

Salah satu keuntungan utama dari format .npz adalah kemampuannya untuk memuat data langsung ke dalam memori tanpa perlu dekompresi manual. Ini memungkinkan sistem untuk mengakses dan memproses data fitur dengan cepat, yang esensial untuk aplikasi pencarian real-time. Proses loading yang cepat ini mengurangi latensi (*delay*) dalam sistem, memastikan bahwa hasil pencarian dapat disajikan kepada pengguna dalam waktu yang minimal. Selain itu, format .npz mendukung penyimpanan dataset yang sangat besar dengan cara yang terstruktur dan terkompresi, memastikan bahwa sistem dapat skala dengan baik seiring bertambahnya jumlah data. Format ini juga bersifat portabel, memungkinkan data untuk dipindahkan antar sistem dengan mudah tanpa kehilangan integritas atau konsistensi data.

Dengan demikian, kombinasi penggunaan NumPy dan format .npz memberikan optimasi yang signifikan dalam sistem pencarian MIDI serupa. NumPy menyediakan kerangka kerja komputasi yang efisien dan teroptimasi untuk melakukan operasi matematis yang kompleks, sementara format .npz memastikan bahwa data fitur dapat disimpan dan diakses dengan cepat dan efisien. Kombinasi keduanya memungkinkan aplikasi untuk menangani dataset besar dengan performa tinggi, baik dari segi penyimpanan maupun komputasi, sehingga pencarian kemiripan MIDI dapat dilakukan secara cepat dan akurat. Implementasi kedua pendekatan ini secara sinergis memastikan bahwa sistem mampu menangani tantangan komputasi dan penyimpanan dalam skala besar, memberikan hasil pencarian yang cepat, akurat, dan dapat diandalkan.

III. IMPLEMENTASI



Gambar 3.1 Diagram Alur Use Case Aplikasi
(sumber: dokumentasi pribadi)

Bab ini membahas implementasi algoritma optimasi cosine similarity untuk pencarian audio MIDI serupa. Implementasi ini melibatkan pengolahan data MIDI, ekstraksi fitur, penyimpanan data terkompresi, serta pengembangan API menggunakan Flask untuk interaksi pengguna. Semua komponen ini diintegrasikan untuk membentuk sistem yang efisien dan responsif dalam melakukan pencarian audio MIDI serupa.

A. Arsitektur Sistem

Sistem yang diimplementasikan terdiri dari beberapa komponen utama: Pengolahan Data MIDI, Ekstraksi Fitur, Penyimpanan Terkompresi, API Flask, dan Mapper. Pengolahan Data MIDI mengelola proses pengunggahan dan ekstraksi fitur dari file MIDI. Ekstraksi Fitur menghasilkan representasi numerik dari data MIDI melalui distribusi nada absolut (ATB), relatif (RTB), dan pertama (FTB). Fitur-fitur ini kemudian disimpan dalam format terkompresi `.npz` untuk efisiensi penyimpanan dan akses cepat. API Flask menyediakan endpoint bagi pengguna untuk mengunggah file MIDI, melakukan pencarian, dan mengelola database. Mapper menghubungkan file MIDI dengan metadata seperti judul lagu, artis, album, dan tahun, memungkinkan penyajian informasi tambahan kepada pengguna.

B. Pemilihan Teknologi dan Library

Implementasi sistem ini memanfaatkan beberapa library Python. Flask dipilih sebagai framework web ringan untuk membangun API karena kemudahannya dalam pengembangan dan kemampuannya menangani

permintaan HTTP dengan cepat. NumPy digunakan untuk komputasi numerik dan operasi matriks yang efisien, esensial dalam perhitungan similarity. mido digunakan untuk membaca dan memproses file MIDI karena kemampuannya menangani berbagai format MIDI dengan mudah. Library tambahan seperti logging, zipfile, dan json digunakan untuk pencatatan log sistem, pengarsipan ZIP, dan manipulasi data JSON, masing-masing, memastikan kestabilan dan efisiensi sistem.

C. Pengolahan Data MIDI

Pengolahan data MIDI dimulai dengan pembacaan dan ekstraksi melodi dari file MIDI menggunakan library `mido`. Fungsi `load_midi` membuka dan membaca file MIDI, sementara fungsi `get_melody` mengekstrak not yang relevan dari pesan `note_on` dengan velocity > 0, memastikan hanya not yang benar-benar dimainkan yang diikutsertakan.

```python
def load_midi(file_path):
    return midi.MidiFile(file_path)

def get_melody(midi_file):
    melody = []
    for track in midi_file.tracks:
        for msg in track:
            if msg.type == "note_on" and msg.velocity > 0:
                melody.append(msg.note)
    return melody
```

Gambar 3.2 Pengolahan Nada MIDI
(sumber: dokumentasi pribadi)

Setelah melodi diekstraksi, fungsi `windowing` membagi melodi menjadi segmen-segmen yang tumpang tindih menggunakan metode sliding window dengan ukuran jendela 40 nada dan langkah pergeseran 6 nada. Ini memungkinkan analisis fitur dalam segmen waktu yang lebih kecil dan terfokus.

```python
def windowing(melody, window_size, slide_size):
    melody_in_windows = []
    for i in range(0, len(melody) - window_size + 1, slide_size):
        window = melody[i:i + window_size]
        melody_in_windows.append(window)
    return melody_in_windows
```

Gambar 3.3 Pembagian melodi dengan *windowing*
(sumber: dokumentasi pribadi)

D. Ekstraksi Fitur ATB, RTB, dan FTB

Setiap jendela melodi diekstrak menjadi tiga jenis fitur utama: Distribusi Nada Absolut (ATB), Distribusi Nada Relatif (RTB), dan Distribusi Nada Pertama (FTB). Fitur-fitur ini dihitung sebagai histogram yang dinormalisasi untuk memastikan konsistensi antar file MIDI.

- ATB menghitung frekuensi kemunculan setiap not dalam jendela.

```python
def get_normalized_ATB(window):
    hist, _ = np.histogram(window, bins=128, range=(0, 127))
    if len(window) == 0:
        return np.zeros_like(hist, dtype=np.float32)
    return hist / np.sum(hist)
```

Gambar 3.4 Penghitungan Vektor Fitur ATB yang ternormalisasi
(sumber: dokumentasi pribadi)

- RTB menghitung distribusi perubahan relatif antar not berturut-turut.

```python
def get_normalized_RTB(window):
    if len(window) < 2:
        return np.zeros(255, dtype=np.float32)
    rtb = np.diff(window)
    hist, _ = np.histogram(rtb, bins=255, range=(-127, 127))
    if np.sum(hist) == 0:
        return hist
    return hist / np.sum(hist)
```

Gambar 3.5 Penghitungan Vektor Fitur RTB yang ternormalisasi
(sumber: dokumentasi pribadi)

- FTB menghitung distribusi perubahan relatif terhadap not pertama dalam jendela.

```python
def get_normalized_FTB(window):
    if len(window) == 0:
        return np.zeros(255, dtype=np.float32)
    ftone = window[0]
    ftb = np.array(window) - ftone
    hist, _ = np.histogram(ftb, bins=255, range=(-127, 127))
    if np.sum(hist) == 0:
        return hist
    return hist / np.sum(hist)
```

Gambar 3.6 Penghitungan Vektor Fitur FTB yang ternormalisasi
(sumber: dokumentasi pribadi)

E. Penyimpanan Fitur MIDI Terkompresi

Fitur-fitur yang diekstraksi disimpan dalam format `.npz` menggunakan `numpy.savez_compressed` untuk efisiensi penyimpanan dan akses cepat. Fungsi `preprocess_midi_dataset` memproses setiap file MIDI dalam direktori yang ditentukan, mengekstrak fitur-fitur, dan menyimpannya bersama dengan label.

```python
def preprocess_midi_dataset(folder_path):
    logging.info("Processing MIDI dataset...")
    processed_dataset = []
    labels = []

    if not os.path.exists(folder_path):
        logging.error("Folder path does not exist")
        return processed_dataset, labels

    for file_name in os.listdir(folder_path):
        if file_name.endswith(".mid"):
            audio_name = file_name.replace('.mid', '')
            labels.append(audio_name)

            file_path = os.path.join(folder_path, file_name)
            midi_file = load_midi(file_path)
            melody = get_melody(midi_file)
            melody_in_windows = windowing(melody, WINDOW_SIZE, SLIDE_SIZE)

            vectors = {
                'ATB': [],
                'RTB': [],
                'FTB': []
            }

            for window in melody_in_windows:
                vectors['ATB'].append(get_normalized_ATB(window))
                vectors['RTB'].append(get_normalized_RTB(window))
                vectors['FTB'].append(get_normalized_FTB(window))

            processed_dataset.append(vectors)

    # Save processed data
    np.savez_compressed(
        f"{PROCESSED_DATA_PATH_MIDI}/processed_midi.npz",
        dataset=processed_dataset,
        labels=labels
    )
    logging.info(f"Processed {len(processed_dataset)} MIDI files")
    return processed_dataset, labels
```

Gambar 3.7 Penyimpanan vektor-vektor dari MIDI dataset ke dalam database NPZ
(sumber: dokumentasi pribadi)

F. Pengembangan API dengan Flask

API dikembangkan menggunakan Flask untuk menyediakan endpoint bagi pengguna dalam mengunggah file MIDI, melakukan pencarian, dan mengelola database. Endpoint utama meliputi:

- Upload MIDI: Mengunggah file MIDI individu dan melakukan pencarian kemiripan.
- Upload ZIP MIDI: Mengunggah arsip ZIP yang berisi banyak file MIDI sekaligus.
- Upload Mapper: Mengunggah mapper yang menghubungkan file MIDI dengan metadata terkait.

Fungsi `handle_midi_upload` menangani pengunggahan file MIDI, ekstraksi fitur, perhitungan cosine similarity dengan database, dan pengembalian hasil pencarian [6].

```python
def handle_midi_upload(uploaded_file):
    global mapper
    if processed_midi_dataset is None or len(processed_midi_dataset) == 0:
        return jsonify({"similar_items": [], "message": "MIDI database is empty."})

    midi_path = os.path.join(UPLOADS_PATH, uploaded_file.filename)
    uploaded_file.save(midi_path)

    # Process uploaded MIDI file
    midi_file = load_midi(midi_path)
    melody = get_melody(midi_file)
    melody_in_windows = windowing(melody, WINDOW_SIZE, SLIDE_SIZE)
    query_vectors = {
        'ATB': [],
        'RTB': [],
        'FTB': []
    }
    for window in melody_in_windows:
        query_vectors['ATB'].append(get_normalized_ATB(window))
        query_vectors['RTB'].append(get_normalized_RTB(window))
        query_vectors['FTB'].append(get_normalized_FTB(window))

    # Search for similar MIDI files
    results = search_midi(query_vectors, processed_midi_dataset, midi_dataset_labels, threshold=0.5)

    # Prepare response
    similar_midis = []
    for result in results[:len(midi_dataset_labels)]:  # Get all results
        idx = result['index']
        midi_label = midi_dataset_labels[idx]
        midi_file_name = midi_label + '.mid'
        midi_info = {
            "label": midi_label,
            "similarity": result['similarity'],
            "url": f"{request.host_url}midi/{midi_file_name}"
        }
        # Get associated image and song details from mapper
        for image_name, song_info in mapper.items():
            if song_info['midi'] == midi_file_name:
                midi_info["associated_image"] = f"{request.host_url}images/{image_name}"
                midi_info["title"] = song_info.get("title", "")
                midi_info["artist"] = song_info.get("artist", "")
                midi_info["album"] = song_info.get("album", "")
                midi_info["year"] = song_info.get("year", "")
                break

        similar_midis.append(midi_info)

    return jsonify({
        "similar_items": similar_midis
    })
```

Gambar 3.8 Fungsi penerima MIDI kueri yang menghasilkan *list* MIDI yang serupa
(sumber: dokumentasi pribadi)

G. Algoritma Pencarian Cosine Similarity

Algoritma pencarian ini menggunakan cosine similarity untuk mengukur kemiripan antara vektor fitur query dengan vektor fitur dalam database. Fungsi `calculate_similarity` menghitung similarity keseluruhan berdasarkan fitur ATB, RTB, dan FTB dengan mempertimbangkan bobot masing-masing fitur.

```python
def calculate_similarity(query_vectors, audio_vectors, weights=None):
    if weights is None:
        weights = {'ATB': 1.0, 'RTB': 1.0, 'FTB': 1.0}

    similarities = []
    for tone_type in ['ATB', 'RTB', 'FTB']:
        query_matrix = np.array(query_vectors[tone_type])  # Query windows x features
        audio_matrix = np.array(audio_vectors[tone_type])  # Audio windows x features

        # Cosine Similarity
        dot_product = np.dot(query_matrix, audio_matrix.T)  # All pairs
        query_norms = np.linalg.norm(query_matrix, axis=1, keepdims=True)
        audio_norms = np.linalg.norm(audio_matrix, axis=1, keepdims=True).T
        cosine_similarities = dot_product / (query_norms * audio_norms + 1e-9)  # Avoid division by zero

        # Take maximum similarity for each query window
        max_similarities = np.max(cosine_similarities, axis=1)
        avg_tone_similarity = np.mean(max_similarities)  # Average
        similarities.append(weights[tone_type] * avg_tone_similarity)

    # Total similarity
    total_similarity = sum(similarities) / sum(weights.values())
    return total_similarity
```

Gambar 3.9 Algoritma Similaritas Kosinus pada Pencarian Midi Serupa
(sumber: dokumentasi pribadi)

Fungsi `search_midi` menggunakan `calculate_similarity` untuk setiap file MIDI dalam dataset dan menyaring hasil yang memenuhi ambang batas tertentu. Hasil kemudian diurutkan berdasarkan similarity tertinggi.

```python
def search_midi(query_vectors, dataset, labels, threshold, weights=None):
    results = []
    for idx, audio_vectors in enumerate(dataset):
        # Calculate similarity
        similarity = calculate_similarity(query_vectors, audio_vectors, weights)

        if similarity >= threshold:
            results.append({"index": idx, "similarity": similarity})

    # Sort results by similarity
    results.sort(key=lambda x: x['similarity'], reverse=True)
    return results
```

Gambar 3.10 Fungsi search_midi mengembalikan *list* berisi audio-audio MIDI yang serupa dengan kueri
(sumber: dokumentasi pribadi)

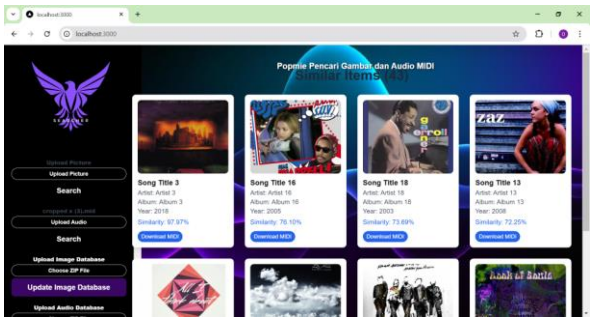
H. Optimasi dengan NumPy dan Format .npz

Optimasi algoritma ini dicapai melalui penggunaan library NumPy dan format penyimpanan .npz. NumPy memungkinkan operasi matriks dan vektor dilakukan secara batch dengan efisiensi tinggi, memanfaatkan optimisasi yang dilakukan dalam bahasa Python. Fungsi-fungsi seperti np.dot (perkalian dot vektor) dan np.linalg.norm (penghitungan panjang vektor) digunakan untuk mempercepat perhitungan cosine similarity.

```python
np.savez_compressed(
    f"{PROCESSED_DATA_PATH_MIDI}/processed_midi.npz",
    dataset=processed_dataset,
    labels=labels
)
```

Gambar 3.11 Bagian Kode yang Menyimpan Vektor dalam database
(sumber: dokumentasi pribadi)

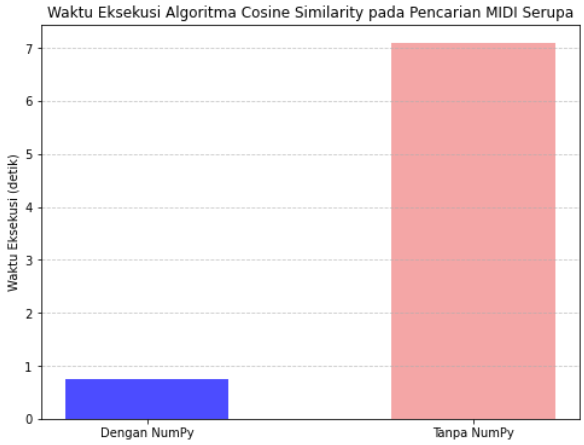
IV. EKSPERIMEN



Gambar 4.1 Tampilan Aplikasi Pencari MIDI Serupa
(sumber: dokumentasi pribadi)

Aplikasi Pencari MIDI serupa menggunakan website Next.JS sebagai antarmuka front-end. Sedangkan, flask berperan sebagai komponen back-end. Front-end berfungsi sebagai wadah upload MIDI yang akan dicari beserta tampilan yang menunjukkan informasi mengenai MIDI dari dataset yang serupa dengan kueri. Hasil pencarian hanya ditampilkan untuk MIDI yang memiliki kemiripan minimal 55%.

Flask digunakan sebagai *framework* backend. Ini berfungsi sebagai penggerak algoritma. Ia mengelola database audio MIDI dan vektor fitur yang berkorespondensi. Selain itu, ia juga melakukan vektorisasi audio MIDI kueri dan menghitung kemiripannya terhadap audio MIDI dari dataset. Pada penelitian ini, digunakan 50 file MIDI sebagai dataset dan 1 file MIDI sebagai kueri.



Gambar 4.2 Diagram batang yang menampilkan waktu eksekusi dengan vs tanpa NumPy
(sumber: dokumentasi pribadi)

Hasil optimasi algoritma ini menunjukkan peningkatan performa yang signifikan. Dengan menggunakan **NumPy**, waktu komputasi hanya membutuhkan **0,74 detik**, dibandingkan dengan skenario tanpa NumPy yang memerlukan **7,09 detik**. Perbandingan ini menunjukkan bahwa optimasi menggunakan **NumPy** mengurangi waktu pencarian hingga hampir sembilan kali lebih cepat. Angka-angka ini menyoroti betapa pentingnya pemilihan alat komputasi yang tepat dalam pengembangan algoritma pencarian audio, di mana efisiensi dan kecepatan adalah faktor krusial untuk kinerja sistem secara keseluruhan.

V. KESIMPULAN

Penelitian ini berhasil mengoptimalkan algoritma cosine similarity untuk pencarian audio MIDI serupa dengan memanfaatkan library NumPy dan format file .npz. Melalui penerapan operasi vektorisasi dan penggunaan fungsi-fungsi matriks yang dioptimalkan oleh NumPy, proses komputasi dapat dilakukan secara signifikan lebih cepat dibandingkan dengan implementasi konvensional berbasis loop tradisional. Hasil uji coba menunjukkan bahwa waktu komputasi dengan NumPy hanya membutuhkan 0,74 detik, sedangkan tanpa optimasi tersebut memerlukan waktu 7,09 detik.

Secara keseluruhan, penelitian ini memberikan kontribusi signifikan dalam pengembangan sistem pencarian audio MIDI yang lebih responsif dan efektif. Optimasi algoritma cosine similarity menggunakan NumPy tidak hanya meningkatkan kecepatan komputasi tetapi juga memperbaiki skalabilitas sistem dalam menghadapi dataset yang semakin besar. Ke depan, penelitian ini dapat dikembangkan lebih lanjut dengan mengeksplorasi optimasi tambahan atau penerapan teknik pembelajaran mesin lainnya untuk meningkatkan akurasi

dan efisiensi pencarian audio. Selain itu, penerapan metode ini pada berbagai jenis data audio lainnya juga dapat dijadikan sebagai acuan selanjutnya untuk memperluas aplikasi teknologi ini dalam industri musik dan multimedia.

VI. PENUTUP

Puji Syukur kepada Tuhan Yang Maha Esa atas rahmat-Nya dalam pembuatan makalah IF2123 Aljabar Linier dan Geometri dengan topik "Optimasi Algoritma Cosine Similarity dalam Pencarian Audio MIDI Serupa". Saya mengucapkan terima kasih kepada dosen IF2123 beserta para penulis yang tulisannya menjadi referensi dalam makalah ini.

REFERENSI

[1] N. Kosugi, Y. Nishihara, S. Kon'ya, M. Yamamuro dan K. Kushima, "Music retrieval by humming-using similarity retrieval over high dimensional feature vector space," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 2-3, 1999.

[2] W. McKinney, Python for Data Analysis : Data Wrangling with Pandas, NumPy, and IPython, Sebastopol: O'Reilly, 2017.

[3] T. N. A. A. S. f. E. N. Computation, "Stefan van der Walt; S. Chris Colbert; Gael Varoquaux," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22-30, 2011.

[4] S. H. Y. Malhotra dan S. H. Mankad, "Audio Similarity Detection," pp. 4-6, 29 Desember 2024.

[5] J. Urbano, M. Schedl dan X. Serra, "Evaluation in Music Information Retrieval," *Journal of Intelligent Information Systems*, vol. 41, pp. 345-369, 2013.

[6] GeeksforGeeks, "Introduction to Web development using Flask," GeeksforGeeks, 27 November 2024. [Online]. Available: https://www.geeksforgeeks.org/python-introduction-to-web-development-using-flask/. [Diakses 31 Desember 2024].

[7] R. Munir, "Vektor di Ruang Euclidean (bagian 1)-2024," 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2024-2025/Algeo-11-Vektor-di-Ruang-Euclidean-Bag1-2024.pdf. [Diakses 31 Desember 2024].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 31 Desember 2024

Jovandra Otniel P. S. (13523141)