Laporan Arsitektur Mendalam: Desain dan Spesifikasi Sistem Pengenalan Emosi Suara Real-Time Berbasis Web

1. Pendahuluan: Paradigma Baru dalam Komputasi Afektif

Evolusi teknologi web dalam dekade terakhir telah mengubah browser dari sekadar penampil dokumen menjadi platform aplikasi yang mampu menjalankan komputasi kinerja tinggi. Salah satu batas terjauh dari evolusi ini adalah integrasi *Artificial Intelligence* (AI) langsung di sisi klien (*edge computing*), khususnya dalam domain *Speech Emotion Recognition* (SER) atau pengenalan emosi suara. Laporan ini menyajikan analisis komprehensif dan spesifikasi teknis untuk membangun sistem SER berbasis web yang memanfaatkan arsitektur neural network wav2vec2, dieksekusi melalui ONNX Runtime Web dalam ekosistem React dan Vite.
Tujuan utama dari laporan ini adalah untuk memberikan landasan teoretis dan praktis yang lengkap guna menjawab kebutuhan pengembangan sistem yang mampu mendeteksi delapan emosi dasar manusia secara *real-time* dengan latensi minimal. Berbeda dengan pendekatan tradisional yang bergantung pada pemrosesan server (*cloud-based inference*), pendekatan yang diusulkan di sini memindahkan beban inferensi ke perangkat pengguna. Hal ini tidak hanya mengurangi biaya bandwidth dan server, tetapi juga meningkatkan privasi pengguna karena data suara mentah tidak perlu meninggalkan perangkat lokal.[1]
Dalam konteks interaksi manusia-komputer (*Human-Computer Interaction*/HCI), kemampuan mesin untuk memahami tidak hanya *apa* yang dikatakan (teks) tetapi juga *bagaimana* cara mengatakannya (emosi) adalah langkah krusial menuju antarmuka yang empatik. Laporan ini akan mengupas tuntas integrasi teknologi *Web Audio API* modern, khususnya AudioWorklet dan SharedArrayBuffer, yang menjadi tulang punggung pemrosesan sinyal digital (*Digital Signal Processing*/DSP) di browser, serta penerapan teori psikologi emosi Robert Plutchik dalam desain antarmuka pengguna.[3]

1.1 Pergeseran dari Cloud ke Edge dalam AI

Secara historis, model *deep learning* seperti wav2vec2 membutuhkan sumber daya komputasi masif yang hanya tersedia di server GPU. Namun, dengan munculnya *WebAssembly* (WASM) dan optimisasi model melalui kuantisasi (misalnya, mengubah bobot model dari 32-bit float menjadi 8-bit integer), kini dimungkinkan untuk menjalankan model-model ini di browser dengan performa mendekati *native*.[2] Transisi ini membawa tantangan teknis tersendiri, mulai dari manajemen memori di browser hingga kebijakan keamanan lintas asal (*Cross-Origin Security*) yang ketat, yang akan dibahas secara mendalam dalam laporan ini.

2. Landasan Teoretis: Psikologi Emosi dan Dataset

Sebelum masuk ke detail teknis implementasi, sangat penting untuk memahami kerangka kerja teoretis yang mendasari klasifikasi emosi dalam sistem ini. Pemilihan model klasifikasi tidak boleh sembarangan, melainkan harus didasarkan pada teori psikologi yang valid dan dataset yang representatif.

2.1 Teori Roda Emosi Robert Plutchik

Sistem ini mengadopsi teori psiko-evolusioner yang diajukan oleh Robert Plutchik pada tahun 1980. Plutchik mengusulkan bahwa terdapat delapan emosi dasar yang berkembang sebagai mekanisme adaptasi biologis untuk kelangsungan hidup.[6] Emosi-emosi ini disusun dalam pasangan berlawanan (misalnya, kegembiraan berlawanan dengan kesedihan, kepercayaan berlawanan dengan kejijikan).

Dalam visualisasi "Roda Emosi" Plutchik, setiap emosi dasar diasosiasikan dengan warna tertentu. Intensitas emosi digambarkan melalui saturasi warna: semakin kuat emosinya, semakin gelap warnanya. Pendekatan ini sangat relevan untuk desain antarmuka pengguna (UI) karena memberikan umpan balik visual yang intuitif tanpa membebani kognitif pengguna dengan teks.[4]

Taksonomi Warna dan Emosi

Berdasarkan literatur desain dan teori warna Plutchik, berikut adalah pemetaan spektrum warna yang diadopsi untuk sistem ini:

Emosi Dasar (Plutchik/RAVDESS)	Warna Representatif	Kode Hex	Makna Psikologis
Marah (Anger)	Merah Tua / Crimson	#DC143C	Berkaitan dengan darah, bahaya, dan respons *fight*. [6]
Takut (Fear)	Hijau Gelap	#228B22	Asosiasi dengan keinginan untuk bersembunyi atau perlindungan alam. [6]
Sedih (Sadness)	Biru / Royal Blue	#4169E1	Berkaitan dengan penarikan diri, kedinginan, dan isolasi. [6]
Bahagia (Joy/Happy)	Kuning / Emas	#FFD700	Simbol matahari, kehangatan, dan koneksi sosial. [6]
Jijik (Disgust)	Ungu / Violet	#9370DB	Berkaitan dengan penolakan atau racun

			(secara metaforis). [7]
Terkejut (Surprise)	Biru Langit	#87CEEB	Reaksi cepat terhadap stimulus tak terduga, orientasi. [6]
Tenang (Calm/Acceptance)	Hijau Pucat	#98FB98	Varian intensitas rendah dari penerimaan atau kepercayaan. [6]
Netral (Neutral)	Abu-abu	#A9A9A9	Titik nol emosi, ketiadaan bias afektif. [9]

2.2 Dataset RAVDESS sebagai Standar Emas

Model wav2vec2 yang digunakan dalam sistem ini di-*fine-tune* menggunakan dataset RAVDESS (*Ryerson Audio-Visual Database of Emotional Speech and Song*). Dataset ini terdiri dari 7.356 rekaman yang dibuat oleh 24 aktor profesional (12 pria, 12 wanita) yang menyuarakan kalimat dengan berbagai intensitas emosi.[10]
Penting untuk dicatat bahwa RAVDESS menyediakan delapan kelas emosi yang sejalan dengan teori Plutchik, meskipun dengan sedikit variasi terminologi. RAVDESS membedakan antara "Netral" (*Neutral*) dan "Tenang" (*Calm*). Secara teknis, "Netral" adalah ketiadaan emosi yang kuat, sedangkan "Tenang" adalah keadaan afektif positif yang rileks.[11] Kemampuan model wav2vec2 untuk membedakan nuansa halus ini adalah salah satu keunggulan utamanya dibandingkan model akustik tradisional yang hanya melihat fitur spektral dasar.

3. Arsitektur Neural Network: Wav2Vec 2.0

Inti dari kecerdasan sistem ini adalah wav2vec2, sebuah arsitektur yang dikembangkan oleh Facebook AI (Meta). Berbeda dengan sistem *Automatic Speech Recognition* (ASR) konvensional yang membutuhkan data berlabel transkrip dalam jumlah besar, wav2vec2 belajar dari audio mentah melalui pembelajaran mandiri (*self-supervised learning*).[10]

3.1 Mekanisme Kerja Model

Arsitektur wav2vec2 terdiri dari tiga komponen utama yang bekerja secara berurutan:

1. Feature Encoder (CNN):
   Bagian ini terdiri dari lapisan Convolutional Neural Network (CNN) temporal multi-layer. Encoder ini menerima sinyal audio mentah (waveform) yang biasanya di-sample pada frekuensi 16kHz. Tugas utamanya adalah mengubah gelombang suara kontinu menjadi representasi laten diskrit berdurasi 20ms-25ms. Dalam konteks implementasi web, efisiensi encoder ini sangat krusial karena beroperasi langsung pada data PCM yang dikirim dari AudioWorklet.5
2. Context Network (Transformer):
   Representasi laten dari CNN kemudian diproses oleh serangkaian lapisan Transformer. Menggunakan mekanisme self-attention, jaringan ini mempelajari ketergantungan

kontekstual di seluruh urutan audio. Artinya, model tidak hanya mendengar suara "saat ini", tetapi memahami konteksnya berdasarkan suara yang mendahului dan mengikutinya. Untuk pengenalan emosi, konteks prosodik—seperti intonasi naik pada akhir kalimat tanya atau getaran suara saat takut—ditangkap secara efektif di lapisan ini.13

3. Classification Head:
   Untuk tugas SER, lapisan terakhir dari model pra-latih (pre-trained) diganti dengan classification head sederhana. Output dari lapisan Transformer di-pooling (biasanya rata-rata atau max pooling) untuk mendapatkan satu vektor representasi untuk seluruh klip audio, yang kemudian dipetakan ke delapan logit emosi RAVDESS.14

3.2 Optimisasi untuk Web: Kuantisasi dan ONNX

Menjalankan model Transformer "Base" atau "Large" (dengan ratusan juta parameter) di browser sangat berat. Oleh karena itu, model harus dikonversi ke format ONNX (*Open Neural Network Exchange*) dan dikuantisasi.

- **Pruning:** Beberapa penelitian menunjukkan bahwa untuk tugas emosi, tidak semua lapisan Transformer diperlukan. Model dapat di-*prune* (dipangkas) dari 24 lapisan menjadi 12 lapisan tanpa penurunan akurasi yang signifikan, namun dengan peningkatan kecepatan inferensi yang drastis.[13]
- **Kuantisasi:** Mengubah presisi bobot model dari 32-bit floating point (FP32) menjadi 8-bit integer (INT8). Teknik ini dapat mengurangi ukuran model hingga 4x (misalnya dari 400MB menjadi 100MB), yang sangat penting untuk mengurangi waktu muat (*load time*) di browser dan penggunaan memori RAM perangkat pengguna.[15]

4. Ekosistem Web Audio API: Mengatasi Limitasi Thread Utama

Tantangan terbesar dalam pemrosesan audio *real-time* di web adalah arsitektur *single-thread* JavaScript. Jika pemrosesan audio dilakukan di *main thread* (utas utama), setiap kali browser merender ulang UI (misalnya karena animasi React atau *scrolling*), pemrosesan audio dapat terhenti sejenak, menyebabkan "glitch" atau suara terputus-putus.

4.1 Evolusi: Dari ScriptProcessorNode ke AudioWorklet

Dahulu, pengembang menggunakan ScriptProcessorNode yang berjalan di *main thread*. Ini terbukti tidak andal untuk aplikasi serius karena latensi yang tidak terduga.[16] Solusi modern yang digunakan dalam spesifikasi ini adalah **AudioWorklet**.
AudioWorklet memungkinkan kode pemrosesan audio (DSP) berjalan di thread terpisah yang disebut *Audio Rendering Thread*. Thread ini memiliki prioritas sangat tinggi di sistem operasi, memastikan pemrosesan audio yang stabil dan bebas gangguan, terlepas dari seberapa berat beban kerja di UI utama.[3]

4.2 Masalah Komunikasi Lintas Thread

Meskipun AudioWorklet memisahkan pemrosesan, data audio yang ditangkap (PCM) harus dikirim ke model AI (yang berjalan di Web Worker lain atau main thread) untuk inferensi. Metode komunikasi standar port.postMessage() memiliki kelemahan fatal: ia menyalin data. Untuk audio *real-time* dengan *sample rate* 16.000 Hz, menyalin data terus-menerus menciptakan sampah memori (*garbage collection*) yang pada akhirnya akan memicu jeda kinerja.[17]

4.3 Solusi: SharedArrayBuffer dan Ring Buffer

Untuk mencapai komunikasi data *zero-copy* (tanpa penyalinan) dan latensi ultra-rendah, arsitektur ini menggunakan **SharedArrayBuffer**. Ini adalah blok memori mentah yang dapat diakses secara simultan oleh *Audio Thread* (penulis) dan *Worker Thread* (pembaca/AI).
Di dalam memori bersama ini, diimplementasikan struktur data **Ring Buffer** (Buffer Sirkular).

1. **Writer (AudioWorklet):** Menulis data mikrofon ke buffer. Saat mencapai ujung buffer, ia kembali ke awal (*wrap around*).
2. **Reader (AI Worker):** Membaca data dari buffer untuk dimasukkan ke model wav2vec2.
3. **Sinkronisasi dengan Atomics:** Menggunakan API Atomics JavaScript (seperti Atomics.store, Atomics.load, Atomics.wait) untuk mengelola pointer "baca" dan "tulis". Ini mencegah *race condition* (kondisi balapan) di mana pembaca mencoba membaca data yang belum ditulis, atau penulis menimpa data yang belum dibaca, tanpa perlu menggunakan kunci (*lock*) yang memblokir thread.[18]

Implementasi *Lock-Free Ring Buffer* ini adalah standar emas untuk aplikasi audio kinerja tinggi di web saat ini.[19]

5. Keamanan Browser: Isolasi Cross-Origin

Penggunaan SharedArrayBuffer membawa implikasi keamanan yang serius. Fitur ini sempat dinonaktifkan di semua browser utama setelah ditemukannya celah keamanan *Spectre* dan *Meltdown* pada CPU modern, yang memungkinkan situs web jahat membaca memori dari tab lain menggunakan pengukuran waktu presisi tinggi (yang dimungkinkan oleh SharedArrayBuffer).[20]

5.1 Persyaratan Header COOP dan COEP

Agar browser mengizinkan penggunaan SharedArrayBuffer, halaman web harus berada dalam status "Cross-Origin Isolated". Ini dicapai dengan mengirimkan dua header HTTP spesifik dari server:

1. Cross-Origin-Opener-Policy (COOP): same-origin
   Header ini mengisolasi proses browser dokumen Anda, memastikan bahwa tidak ada dokumen dari asal (origin) lain yang dapat berinteraksi dengannya dalam satu grup jendela.
2. Cross-Origin-Embedder-Policy (COEP): require-corp
   Header ini mencegah dokumen memuat sumber daya lintas asal (seperti gambar atau skrip dari CDN) kecuali sumber daya tersebut secara eksplisit mengizinkannya melalui header CORS atau CORP (Cross-Origin Resource Policy).21

5.2 Konfigurasi Vite

Dalam pengembangan menggunakan Vite, header ini harus dikonfigurasi di server pengembangan (vite.config.js). Tanpa ini, objek global SharedArrayBuffer tidak akan didefinisikan (undefined), dan aplikasi akan gagal saat inisialisasi. Selain itu, Vite perlu dikonfigurasi untuk menangani file .wasm dari onnxruntime-web sebagai aset statis, bukan modul JavaScript, seringkali memerlukan plugin seperti vite-plugin-static-copy untuk memindahkan file WASM ke direktori output yang benar saat *build*.[23]

6. Infrastruktur Backend: Supabase dan Keamanan Data

Meskipun inferensi dilakukan di sisi klien, aplikasi memerlukan backend untuk otentikasi pengguna dan penyimpanan riwayat rekaman suara. **Supabase** dipilih karena menyediakan layanan *Backend-as-a-Service* (BaaS) yang mencakup database PostgreSQL, otentikasi, dan penyimpanan file (Storage) dengan integrasi keamanan yang ketat.

6.1 Row Level Security (RLS) pada Storage

Keamanan data adalah prioritas utama. Kita tidak bisa hanya mengandalkan logika aplikasi (frontend) untuk mencegah pengguna A melihat file pengguna B. Supabase memungkinkan penerapan kebijakan keamanan langsung di level database menggunakan **Row Level Security (RLS)** PostgreSQL.

Untuk penyimpanan file audio (Blob), kebijakan RLS diterapkan pada tabel storage.objects. Kebijakan ini harus dirancang sedemikian rupa sehingga:

1. **Insert (Upload):** Hanya diizinkan jika pengguna terotentikasi DAN jalur file (*path*) dimulai dengan ID pengguna (uid) mereka sendiri.
2. **Select (Download):** Hanya diizinkan jika pemilik file (berdasarkan metadata atau path) cocok dengan uid pengguna yang sedang login.

Penerapan RLS ini memastikan bahwa bahkan jika seseorang berhasil memanipulasi kode frontend untuk mencoba mengunggah ke folder orang lain, database akan menolak transaksi tersebut di level terendah.[26]

Berikut adalah analisis tabel perbandingan kebijakan keamanan:

Operasi	Peran (Role)	Kondisi (SQL Policy)	Hasil
Upload	Anonim	-	Ditolak (403 Forbidden)
Upload	Terotentikasi	path!= auth.uid() + '/*'	Ditolak (Pelanggaran RLS)
Upload	Terotentikasi	path == auth.uid() + '/*'	**Diizinkan**
Lihat	Terotentikasi	owner_id!= auth.uid()	Ditolak (Data tidak terlihat)

7. Spesifikasi Implementasi Teknis dan Prompt

Bagian berikut ini merangkum seluruh analisis di atas menjadi sebuah spesifikasi teknis yang dapat dieksekusi. Ini menjawab permintaan pengguna untuk "membuatkan ulang prompt websitenya dalam bahasa Inggris". Prompt ini dirancang untuk diberikan kepada *AI Coding Assistant* (seperti Cursor, GitHub Copilot, atau ChatGPT) untuk menghasilkan kode produksi yang lengkap.

7.1 Strategi Prompting

Prompt yang efektif untuk sistem sekompleks ini harus bersifat modular dan eksplisit. Ia tidak boleh ambigu mengenai versi library (misalnya, memaksa penggunaan onnxruntime-web versi terbaru) atau pola arsitektur (memaksa AudioWorklet vs ScriptProcessor).
Prompt di bawah ini mencakup:

1. **Setup Proyek:** Vite, React, struktur direktori.
2. **Konfigurasi Keamanan:** Header COOP/COEP di Vite.
3. **Logika Audio:** Kode AudioProcessor untuk Worklet dan Ring Buffer.
4. **Logika AI:** Worker untuk inferensi ONNX.
5. **Integrasi Supabase:** SQL untuk RLS.
6. **Desain UI:** Kode warna Hex Plutchik.

---

8. The Ultimate System Prompt (English Artifact)

Below is the comprehensive system prompt designed to generate the entire application described in this report.
Role: Act as a Senior Frontend Architect and specialist in WebAssembly/DSP.
Task: Create a production-ready, real-time Speech Emotion Recognition (SER) web application.
Technical Stack:

- **Framework:** React 18+ (Functional Components, Hooks)
- **Build Tool:** Vite (Must be configured for WASM & Security Headers)
- **ML Runtime:** onnxruntime-web (WebAssembly backend)
- **Backend:** Supabase (Auth & Storage)
- **Styling:** Tailwind CSS (preferred) or Styled Components.

Core Constraints & Requirements:

1. **Neural Network Model:**
    - The app must load a quantized .onnx version of a **wav2vec2** model fine-tuned on the **RAVDESS** dataset.
    - **Labels:** The model must classify the input into these 8 specific emotions: *Neutral, Calm, Happy, Sad, Angry, Fearful, Disgust, Surprised*.
    - *Context:* Do not implement standard ASR; this is an emotion classification task.
2. **High-Performance Audio Pipeline (Non-Negotiable):**

- **AudioWorklet:** You MUST use the Web Audio API's AudioWorklet for signal processing. Do NOT use the deprecated ScriptProcessorNode.
- **Thread Communication:** Implement a **SharedArrayBuffer** with a **Lock-Free Ring Buffer** pattern to transport PCM data (Float32) from the AudioWorklet (audio thread) to a dedicated Web Worker (inference thread).
- **Synchronization:** Use JavaScript Atomics (load/store/notify) to manage the read/write indices of the ring buffer safely to prevent race conditions without blocking the audio thread.

3. **Browser Security & Vite Configuration:**
   - The application requires SharedArrayBuffer, which mandates **Cross-Origin Isolation**.
   - Configure vite.config.js to serve the following headers on the development server:
     - Cross-Origin-Opener-Policy: same-origin
     - Cross-Origin-Embedder-Policy: require-corp
   - Include a Vite plugin (like vite-plugin-static-copy) to ensure onnxruntime-web WASM files (.wasm) are copied from node_modules to the distribution/public folder and served correctly.
4. **Supabase Integration & Security:**
   - Implement specific SQL Policies for Row Level Security (RLS) on the storage.objects table.
   - **Policy Rule:** Authenticated users can ONLY upload audio blobs to a folder path that exactly matches their User UID. (e.g., uploads/{user_id}/{filename}.wav).
   - Provide the exact SQL commands to set up the Bucket and Policies.
5. **UI/UX - Plutchik's Wheel of Emotions:**
   - The UI must visualize the detected emotion using color-coded indicators based on Plutchik's theory.
   - **Color Mapping (Strictly use these Hex codes):**
     - Angry: #DC143C (Crimson)
     - Fearful: #228B22 (Forest Green)
     - Sad: #4169E1 (Royal Blue)
     - Happy: #FFD700 (Gold)
     - Disgust: #9370DB (Medium Purple)
     - Surprised: #87CEEB (Sky Blue)
     - Calm: #98FB98 (Pale Green)
     - Neutral: #A9A9A9 (Grey)
   - Implement a visual "confidence meter" or dynamic opacity that changes based on the probability score of the predicted emotion.

Deliverables:

1. **Project Structure:** A tree view of the recommended folder structure.
2. **Configuration:** The complete vite.config.js file.
3. **Audio Processor:** The public/audio-processor.js code implementing the Ring Buffer writer.
4. **Inference Worker:** The src/workers/inference.js code implementing the ONNX session

and Ring Buffer reader.
5. **React Logic:** A custom hook useEmotionInference that ties the AudioContext, Worklet, and Worker together.
6. **SQL:** The specific PostgreSQL commands for Supabase RLS.

---

9. Kesimpulan dan Rekomendasi Implementasi

Laporan ini telah menguraikan spesifikasi lengkap untuk sistem SER berbasis web yang canggih. Kompleksitas utama terletak pada orkestrasi tiga *thread* paralel: Main Thread (UI React), Audio Thread (Worklet), dan Inference Thread (Worker). Kegagalan dalam menyinkronkan ketiga komponen ini akan mengakibatkan aplikasi yang tidak responsif atau audio yang cacat.
Dari analisis riset yang dilakukan, ditemukan bahwa hambatan terbesar bagi pengembang adalah konfigurasi lingkungan pengembangan (Vite/Webpack) untuk mendukung SharedArrayBuffer dan WASM secara bersamaan. Oleh karena itu, mengikuti konfigurasi vite.config.js yang disajikan dalam prompt di atas adalah langkah kritis.
Selain itu, penggunaan dataset RAVDESS yang dipetakan ke Roda Emosi Plutchik memberikan validitas ilmiah pada output sistem. Ini bukan sekadar "deteksi marah/senang", tetapi sebuah sistem yang didasarkan pada taksonomi emosi yang mapan, divisualisasikan dengan prinsip desain psikologis yang kuat.
Dengan menerapkan arsitektur ini, pengembang tidak hanya membangun aplikasi demo, tetapi sebuah platform kelas industri yang menunjukkan potensi masa depan AI di sisi klien: cepat, privat, dan aman. Direkomendasikan untuk memulai implementasi dengan memastikan lingkungan "Cross-Origin Isolated" berfungsi terlebih dahulu (cek window.crossOriginIsolated di konsol browser) sebelum mencoba memuat model ONNX, karena ini adalah prasyarat mutlak bagi fitur-fitur canggih lainnya.

Langkah Selanjutnya

Untuk pengembangan tahap lanjut, disarankan untuk mengeksplorasi penggunaan *WebGPU* sebagai *backend* eksekusi ONNX Runtime di masa depan. Meskipun saat ini dukungan WebGPU masih dalam tahap awal dibandingkan WASM, ia menjanjikan akselerasi yang jauh lebih tinggi untuk model Transformer besar seperti wav2vec2, yang berpotensi memungkinkan penggunaan model tanpa kuantisasi (FP32) di browser dengan kinerja *real-time*.

Works cited

1. Implementing AudioWorklets with React | by Aphra Bloomfield | HackerNoon.com - Medium, accessed on December 24, 2025, https://medium.com/hackernoon/implementing-audioworklets-with-react-8a80a470474
2. Build for web | onnxruntime, accessed on December 24, 2025, https://onnxruntime.ai/docs/build/web.html

3. Web Audio API vs. native, closing the gap, take 2 - NTNU, accessed on December 24, 2025, https://www.ntnu.edu/documents/1282113268/1290797387/WAC2019-CameraReadySubmission-78.pdf/913d4cec-4cb0-26b9-a1cf-fe951c89b8cc?t=1575328682000
4. Putting Some Emotion into Your Design – Plutchik's Wheel of Emotions | IxDF, accessed on December 24, 2025, https://www.interaction-design.org/literature/article/putting-some-emotion-into-your-design-plutchik-s-wheel-of-emotions
5. AventIQ-AI/wav2vec2-base_speech_emotion_recognition - Hugging Face, accessed on December 24, 2025, https://huggingface.co/AventIQ-AI/wav2vec2-base_speech_emotion_recognition
6. Plutchik's Wheel of Emotions: A Designer's Guide to Impact - Bricx Labs, accessed on December 24, 2025, https://bricxlabs.com/blogs/plutchik-s-wheel-of-emotions
7. Plutchik's Wheel of Emotions: Feelings Wheel - Six Seconds, accessed on December 24, 2025, https://www.6seconds.org/2025/02/06/plutchik-wheel-emotions/
8. Colour Me Emotional, accessed on December 24, 2025, https://figshare.unimelb.edu.au/ndownloader/files/45650799
9. Common Emotions Color Scheme - Palettes - SchemeColor.com, accessed on December 24, 2025, https://www.schemecolor.com/common-emotions.php
10. ehcalabres/wav2vec2-lg-xlsr-en-speech-emotion-recognition - Hugging Face, accessed on December 24, 2025, https://huggingface.co/ehcalabres/wav2vec2-lg-xlsr-en-speech-emotion-recognition
11. Dpngtm/wav2vec2-emotion-recognition - Hugging Face, accessed on December 24, 2025, https://huggingface.co/Dpngtm/wav2vec2-emotion-recognition
12. MELT: Towards Automated Multimodal Emotion Data Annotation by Leveraging LLM Embedded Knowledge - arXiv, accessed on December 24, 2025, https://arxiv.org/html/2505.24493v1
13. Wav2vec2 Large Robust 12 Ft Emotion Msp Dim · Models - Dataloop, accessed on December 24, 2025, https://dataloop.ai/library/model/audeering_wav2vec2-large-robust-12-ft-emotion-msp-dim/
14. audeering/wav2vec2-large-robust-12-ft-emotion-msp-dim - Hugging Face, accessed on December 24, 2025, https://huggingface.co/audeering/wav2vec2-large-robust-12-ft-emotion-msp-dim
15. README.md · darjusul/wav2vec2-ONNX-collection at 5e8575171e3137f4f69c88cdf133005c742d3df9 - Hugging Face, accessed on December 24, 2025, https://huggingface.co/darjusul/wav2vec2-ONNX-collection/blob/5e8575171e3137f4f69c88cdf133005c742d3df9/README.md
16. Audio worklet design pattern | Blog - Chrome for Developers, accessed on

December 24, 2025, https://developer.chrome.com/blog/audio-worklet-design-pattern
17. Exposing buffers between audio worklet and context · Issue #784 · WebAudio/web-audio-api, accessed on December 24, 2025, https://github.com/WebAudio/web-audio-api/issues/784
18. Ability to append to AudioBuffer · Issue #1825 · WebAudio/web-audio-api - GitHub, accessed on December 24, 2025, https://github.com/WebAudio/web-audio-api/issues/1825
19. AudioWorklet | Web Audio Samples, accessed on December 24, 2025, https://googlechromelabs.github.io/web-audio-samples/audio-worklet/
20. Cross-Origin-Isolation with SvelteKit, Vite, and Firebase | Captain Codeman, accessed on December 24, 2025, https://www.captaincodeman.com/cross-origin-isolation-with-sveltekit-vite-and-firebase
21. How can I fix SharedArrayBuffer is not defined? | Vercel Knowledge Base, accessed on December 24, 2025, https://vercel.com/kb/guide/fix-shared-array-buffer-not-defined-nextjs-react
22. Cross-Origin-Embedder-Policy (COEP) header - HTTP - MDN Web Docs, accessed on December 24, 2025, https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cross-Origin-Embedder-Policy
23. Support Cross-Origin-Opener-Policy and Cross-Origin-Embedder-Policy on dev server · Issue #3909 · vitejs/vite - GitHub, accessed on December 24, 2025, https://github.com/vitejs/vite/issues/3909
24. Support Cross-Origin-Opener-Policy and Cross-Origin-Embedder-Policy on hmr dev server · Issue #16536 · vitejs/vite - GitHub, accessed on December 24, 2025, https://github.com/vitejs/vite/issues/16536
25. Unable to set response headers for SharedArrayBuffer · Issue #11293 · sveltejs/kit - GitHub, accessed on December 24, 2025, https://github.com/sveltejs/kit/issues/11293
26. Storage Access Control | Supabase Docs, accessed on December 24, 2025, https://supabase.com/docs/guides/storage/security/access-control