# **Apex Music Engine: Functionality & Logic (v2.1)**

# **The "Neuro-Stimulation" Architecture**

## **1. Core Philosophy**

The Apex Engine is not a linear generator; it is an **Evolutionary Tree**. It treats every song concept as a seed that can branch into multiple variations based on "Tradeoff Logic."

**Code Location:** src/core/orchestrator.py (Class: ApexOrchestrator)

## **2. The "Council of Agents" (Analysis Layer)**

### **Stream Alpha: Lyrical Engineering**

*Located in: src/agents/lyrical/*

1. **Agent Flow (agent\_flow.py)**
   * **Function:** Measures Plosive Density (Impact) and Syllabic Velocity (Speed).
   * **New Feature:** **Breath Injection**. Auto-splices long lines with (breath) tokens based on lung capacity limits (16 syllables).
   * **Metric:** velocity\_syllables\_per\_beat.
2. **Agent Bars (agent\_bars.py)**
   * **Function:** Measures Rhyme Density and Multi-Syllabic Complexity.
   * **New Feature:** **Slant Rhyme Detection**. Uses Phonetic Neighborhoods (e.g., AA1 ≈ AA2) to detect assonance.
   * **Metric:** multi\_syllabic\_chains.
3. **Agent Context (agent\_context.py)**
   * **Function:** Narrative Consistency. Checks Persona vocabulary usage.
   * **New Feature:** **Persona Drift Detection**. Ensures the "Vibe" stays consistent from Verse to Chorus.

### **Stream Beta: Psycho-Acoustic Engineering**

*Located in: src/agents/audio/*

1. **Agent Spectral (agent\_spectral.py)**
   * **Function:** Frequency Analysis.
   * **Metric:** **Neuro-Saturation**. Measures energy in the 30-60Hz (Chest Hit) band.
   * **Metric:** **Mud Ratio**. Measures clutter in 200-500Hz.
   * **Metric:** **Frisson Events**. Detects rapid spectral centroid expansion (Drops).
2. **Agent Groove (agent\_groove.py)**
   * **Function:** Micro-Timing.
   * **Metric:** **Humanization Score**. Detects ms offsets from the grid. Enforces "Dilla Swing" or "Quantized Rage" based on genre.

## **3. The "Auto-Corrector" (Creative Problem Solver)**

*Located in: src/core/feedback\_logic.py*

The system uses **Branching Logic** to solve problems.

The Tradeoff Mechanism:

If a song scores 75/100 (Mediocre), the logic generates two paths:

* **Path A (The Engineer):** "Fix the mix." Injects [Dry Mix], [High Pass Filter].
* **Path B (The Artist):** "Embrace the grit." Injects [Lo-Fi], [Distortion], and changes Genre to "Grime".

## **4. The Positive Attribute Checklist**

*Located in: src/utils/report\_generator.py*

Instead of just errors, every report includes a **Dopamine Drivers** section:

* ✅ **Tactile Bass:** Sub-bass > 40% (Physical sensation confirmed).
* ✅ **Hypnotic Flow:** Rhyme density > 60% (Trance state induction).
* ✅ **Viral Hook:** Chorus structure creates an "Earworm Loop" (Repetition penalty disabled for hooks).

## **5. Input/Output Protocol**

* **Inputs:** project.json (Lyrics, Genre Key, Persona Key).
* **Outputs:**
  + data/processing/Report\_{Title}.md: The Brutal Critique.
  + data/processing/Candidates\_{Title}.json: The Forking Paths (Prompt Options).
  + data/stems/: Decomposed audio files.

# **Apex System Audit & Critique**

# **Date: 2025-05-22**

# **Auditor: Junior Dev "Crash Test" Persona**

## **1. The Junior Developer Critique Session**

*Questions raised by a junior dev trying to run this for the first time.*

### **Q1: "Where does the recursion actually happen?"**

**Critique:** The orchestrator.py has a loop (for iteration in range(max\_retries)), but FeedbackLogic only returns a single patch. If I want to try *two different ways* to fix a song (e.g., "Add Bass" vs "Change Genre"), the current system forces me down one path linearly. We are missing **Branching Logic**.

### **Q2: "How do I know if my API keys are actually working without wasting credits?"**

Critique: The main.py just runs. If my Suno key is invalid, it defaults to simulation, but doesn't explicitly tell me "HEY, YOU ARE IN SIMULATION MODE" loudly enough. I might think I'm generating real music.

Fix: We need a strict EnvironmentCheck at startup that prints a status table.

### **Q3: "The Error Logs are scary but vague."**

Critique: If AgentSpectral crashes, the log says CRITICAL FAILURE. But it doesn't tell me who called it. Was it the Orchestrator? Was it a unit test?

Fix: We need Hierarchical Context Logging. Pass a trace\_id or caller\_context string down the stack (e.g., Orchestrator::Iteration\_1::AudioAnalysis).

### **Q4: "What if I like the 'Bad' version better?"**

Critique: The system discards previous iterations. If Iteration 2 is technically "better" (higher PVS) but Iteration 1 had more "soul" (better Persona match), I lose Iteration 1.

Fix: The Orchestrator must return a Candidate List, not just the final result.

## **2. Weak Point Analysis (The "Blind Spots")**

### **A. The "Positive Attribute" Gap**

Current State: We report failures ("Muddy Mix", "Bad Rhyme").

Missing: We do not report Successes. We need a Dopamine Checklist output.

* "✅ Frisson Detected at 0:45"
* "✅ 808s hitting optimal 45Hz saturation"
* "✅ Lyrics contain high-value keywords for 'Rage' persona"

### **B. The Audio Simulation Gap**

Current State: Without a Suno key, the system generates a 0-byte dummy MP3.

Consequence: librosa crashes immediately on 0-byte files. The test suite fails unless we provide a synthetic audio file (like a sine wave) to trick the agents.

### **C. Tradeoff Mechanism**

Current State: Optimizes for Score.

Missing: Optimizes for Variance. We should offer:

* **Path A (Safe):** Fix the mixing errors.
* **Path B (Creative):** Lean into the errors (e.g., "It's not muddy, it's Lo-Fi").

## **3. Next Steps (Action Plan)**

1. **Update Orchestrator:** Implement BranchingCandidate logic to track multiple versions of the song.
2. **Update FeedbackLogic:** Allow generating multiple distinct patches (Conservative vs. Radical).
3. **Create Test Harness:** A script that synthesizes a dummy .wav file so AgentSpectral actually runs and produces data, even without Suno.

# **🎵 Apex Music Engine (v2.0)**

**The Psycho-Acoustic Dopamine Engineering System.**

Apex is not a music generator. It is a **Computational Critic** and **Prompt Compiler**. It uses a "Council of Agents" to analyze your lyrics and concepts against mathematical models of viral success (Rhyme Density, Plosive Impact, Spectral Balance) before generating the optimal instruction set for Suno AI.

## **🛠️ Installation**

### **1. Prerequisites**

* Python 3.9+
* **eSpeak-NG** (Required for the phonemizer agent):
  + *Mac:* brew install espeak
  + *Windows:* Download installer from [espeak-ng releases](https://github.com/espeak-ng/espeak-ng/releases)
  + *Linux:* sudo apt-get install espeak-ng

### **2. Setup**

# Clone the repository  
git clone [https://github.com/your-repo/apex-engine.git](https://github.com/your-repo/apex-engine.git)  
cd apex-engine  
  
# Create virtual environment  
python -m venv venv  
source venv/bin/activate # Windows: venv\Scripts\activate  
  
# Install Python dependencies  
pip install -r requirements.txt  
  
# Download NLTK data (Run once)  
python -c "import nltk; nltk.download('cmudict')"

## **🚀 Usage**

### **Option 1: The Quick Simulation**

Run the engine with default mock data to see the critique system in action.

python main.py

### **Option 2: Custom Project**

1. Create a JSON file (e.g., my\_song.json) with your concept:  
   {  
    "title": "Cyber Heartbreak",  
    "lyrics": "[Verse 1]\nWalking in the rain...",  
    "bpm": 140,  
    "genre\_key": "emotional\_drill",  
    "persona\_key": "the\_sad\_crooner"  
   }  
     
   *(See apex\_engine/config/genres.json for valid keys)*
2. Run the engine:  
   python main.py --project my\_song.json
3. Review the Report:  
   Check data/processing/Report\_CyberHeartbreak.md. It contains the "Brutal Critique" from the agents.
4. Generate:  
   Copy the outputted "Black Magic" tags from the console directly into Suno.

## **🧠 The Architecture**

* **Stream Alpha (Text):**
  + Agent Flow: Measures physical impact (Plosives) and Breath Control.
  + Agent Bars: Measures Rhyme Density and Multi-Syllabic complexity.
  + Agent Context: Checks narrative consistency and Persona fidelity.
* **Stream Beta (Audio Planning):**
  + Suno Interface: Injects undocumented meta-tags ([Wide Stereo], [Micro-Timing]) based on the genres.json database.

## **📂 File Structure**

* apex\_engine/config/: The Brains (Weights, Genres, Personas).
* src/agents/: The Logic (Python scripts analyzing input).
* src/core/: The Orchestrator (Main loop).
* data/processing/: Where your Critique Reports are saved.

# **Apex Engine - API Configuration**

# **Rename this file to .env and fill in your keys**

# **--- Intelligence Providers ---**

OPENAI\_API\_KEY=sk-your-key-here

GROK\_API\_KEY=grok-your-key-here

ANTHROPIC\_API\_KEY=sk-ant-your-key-here

# **--- Music Generation ---**

SUNO\_API\_KEY=suno-your-key-here

SUNO\_ENDPOINT=https://www.google.com/search?q=https://studio-api.suno.ai/api/generate

# **--- Configuration ---**

PREFERRED\_LLM=gpt-4o

LOG\_LEVEL=INFO

# **Apex Music Engine: File Architecture & System Design**

# **Version: 2.0.0 (The "Dopamine" Update)**

# **Status: Blueprint Phase**

## **1. Directory Structure**

apex\_engine/

│

├── config/ # "The Brains" - Static Knowledge Bases

│ ├── genres.json # Definitions of "BPM", "Key", "Tags" for specific sub-genres (e.g., Phonk, Rage)

│ ├── personas.json # Lyrical identities (e.g., "Sad Robot", "Aggressive Drifter")

│ ├── suno\_constants.py # Meta-tag libraries, banned words, prompt limits

│ ├── weights.json # Weighting logic for the PVS (Predicted Virality Score) algorithm

│ └── frisson\_triggers.json # Library of spectral shapes known to cause "chills" (e.g., 300Hz sweeps)

│

├── data/ # "The Vault" - Inputs and Outputs

│ ├── raw\_input/ # User concept notes, reference tracks

│ ├── suno\_downloads/ # Raw MP3s downloaded from Suno

│ ├── stems/ # Demucs-separated audio files (Vocals, Bass, Drums, Other)

│ └── analysis\_cache/ # Temporary analysis files (Spectrograms, Textgrids)

│

├── docs/ # "The Wisdom" - Source of Truth

│ ├── FILE\_ARCHITECTURE.md # This file

│ ├── FEATURES.md # The detailed functionality bible

│ └── CRITIQUE.md # The adversarial audit log

│

├── src/ # "The Engine" - Source Code

│ ├── init.py

│ │

│ ├── agents/ # The Council of 12 (Adversarial Critics)

│ │ ├── init.py

│ │ ├── agent\_base.py # Abstract base class for all agents

│ │ │

│ │ ├── lyrical/ # Stream Alpha: Text Analysis

│ │ │ ├── agent\_bars.py # Rhyme density, multi-syllabic chain count

│ │ │ ├── agent\_flow.py # Stress-timing, phoneme plosive analysis

│ │ │ ├── agent\_vowel.py # Assonance heatmaps & vowel color analysis

│ │ │ └── agent\_context.py# Narrative consistency check (LLM)

│ │ │

│ │ ├── audio/ # Stream Beta: Sound Analysis

│ │ │ ├── agent\_groove.py # Micro-timing, swing detection, "human" feel

│ │ │ ├── agent\_spectral.py # Frequency masking, sub-bass saturation

│ │ │ ├── agent\_frisson.py# Detects "chill-inducing" spectral centroids

│ │ │ ├── agent\_hook.py # Time-to-Impact analysis (0-3s rule)

│ │ │ └── agent\_split.py # Wrapper for Demucs/Spleeter

│ │ │

│ │ └── cultural/ # Stream Gamma: Context Analysis

│ │ ├── agent\_trend.py # Scrapes current BPM/Key trends

│ │ └── agent\_meme.py # Analyzes "Quotability" for captions

│ │

│ ├── core/ # System Logic

│ │ ├── orchestrator.py # Main loop: Inputs -> Debate -> Prompt -> Analysis

│ │ ├── predictor.py # Calculates the PVS (Hit Prediction Score)

│ │ ├── suno\_interface.py # Generates the JSON-to-Text prompt string

│ │ └── stem\_manager.py # Handles file I/O for stem separation

│ │

│ └── utils/ # Helper Functions

│ ├── audio\_tools.py # Librosa wrappers, LUFS metering

│ ├── nlp\_tools.py # Spacy/Phonemizer wrappers

│ ├── visualization.py # Generates plots for the dashboard

│ └── logger.py # Detailed event logging

│

├── tests/ # Unit Tests for Agents

│ ├── test\_rhyme\_density.py

│ ├── test\_bass\_detection.py

│ └── test\_prompt\_format.py

│

├── main.py # Entry point (CLI)

├── requirements.txt # Dependencies (librosa, demucs, spacy, torch, etc.)

└── README.md

# **Apex Music Engine: Comprehensive Features & Protocols**

# **"The Source of Truth"**

## **1. System Philosophy: The "Dopamine-First" Approach**

We do not build songs; we build **Psycho-Acoustic Assets**. Every decision, from the phoneme choice in the lyrics to the millisecond delay of the snare drum, is calculated to trigger a specific neuro-chemical response (Dopamine, Adrenaline, or Oxytocin).

## **2. Stream Alpha: Lyrical & Vocal Engineering**

*Goal: Maximum Phonetic "Stickiness" and Rhythmic Lock.*

### **Agent "Phoneme-Vector" (The Flow Scientist)**

* **Function:** Maps the physical impact of mouth shapes.
* **Metric 1: Plosive Density Index (PDI):**
  + *Logic:* Hard consonants (P, B, T, K, D, G) trigger a percussive response in the brain.
  + *Target:* >15 Plosives per bar for "Aggressive" tracks.
* **Metric 2: Syllabic Velocity (SV):**
  + *Logic:* Measures syllables per second.
  + *Constraint:* Must match the "Genre Speed Limit" (e.g., Migos-style triplet flow = 12 syllables/bar).
* **New Feature:** **"Breath Injection"**: Auto-inserts (breath) tokens into the lyrics if a phrase exceeds 14 syllables, forcing Suno to generate a realistic inhalation sound.

### **Agent "Vowel-Color" (The Assonance Architect)**

* **Function:** Ensures sonic cohesion (The "Earworm" Effect).
* **Metric: Vowel Entropy Score:**
  + *Logic:* Viral hooks often rely on a single dominant vowel sound (e.g., "Drake's 'Ay' flow").
  + *Algorithm:* Maps every vowel to a color. Generates a "Heatmap." If the heatmap is too chaotic, the hook is rejected.

### **Agent "Bars" (The Rhyme Auditor)**

* **Function:** Mathematical complexity analysis.
* **Metric: Rhyme Density (RD):**
  + *Formula:* (Total Rhyming Syllables / Total Syllables) \* 100.
  + *Target:* >40% for Lyrical Rap, >20% for Vibe Rap.
* **Metric: Chain-Link Count:**
  + *Logic:* Detects multi-syllabic rhyme schemes spanning >2 lines.

## **3. Stream Beta: Instrumental & Psycho-Acoustic Engineering**

*Goal: Maximum Physical Response and Platform Optimization.*

### **Agent "Groove-Quant" (The Rhythm Surgeon)**

* **Function:** Analyzes "Swing" and "Feel."
* **Metric: Micro-Timing Offset (MTO):**
  + *Logic:* Measures the ms delay of the snare relative to the grid.
  + *Target:* "Dilla Swing" requires a 10-25ms delay on even 1/16th notes.
  + *Action:* If drums are 100% quantized (0ms offset), it injects [Loose Quantization] or [Live Drummer] tags.

### **Agent "Frisson-Seeker" (The Chill Detector)**

* **Function:** Detects moments that cause "Skin Orgasms."
* **Metric: Spectral Centroid Shift:**
  + *Logic:* A rapid shift from "Dark/Muffled" to "Bright/Open" triggers frisson.
  + *Target:* A 50% increase in Spectral Centroid within 500ms (The "Drop").
* **Metric: Dynamic Range Expansion:**
  + *Target:* The chorus must be +3dB louder (perceived) than the verse.

### **Agent "Spectrogram" (The Mix Engineer)**

* **Function:** Frequency masking and balance.
* **Metric: The "Mud" Ratio:**
  + *Logic:* Energy in 200-400Hz vs Total Energy.
  + *Constraint:* Must be < 30%.
* **Metric: Sub-Bass Saturation:**
  + *Logic:* Energy in 30-60Hz. If low, the song fails the "Car Test."

## **4. Stream Gamma: Cultural & Memetic Context**

*Goal: Social Media Viability.*

### **Agent "Meme-Vector" (The TikTok Strategist)**

* **Function:** Predicts "Quotability."
* **Metric: Caption Capability Score:**
  + *Logic:* Extracts 1-line couplets. Checks them against a database of trending Instagram captions.
  + *Pass Condition:* At least 2 lines must be viable as standalone text posts.

### **Agent "Trend-Jacker" (The Context Aware)**

* **Function:** Ensures BPM/Key compliance.
* **Metric:** **BPM Inflation Index.**
  + *Logic:* Checks the average BPM of the Billboard Hot 100. (Currently rising +2 BPM/year).
  + *Action:* Auto-adjusts user request from 140 BPM to 144 BPM to match future trends.

## **5. The Prediction Algorithm (PVS)**

The central mathematical formula determining generation.

$$PVS = (W\_{Ly} \cdot S\_{Ly}) + (W\_{Au} \cdot S\_{Au}) + (W\_{Cul} \cdot S\_{Cul}) + \epsilon$$

* $\epsilon$ **(The Chaos Factor):** A random variable (0-5%) allowing for "Happy Accidents" to pass through filters.

## **6. The Suno "Black Magic" Protocol**

We exploit undocumented Suno behaviors using specific meta-tag combinations.

* **The "Wall of Sound" Hack:** [Double-Tracked Vocals], [Wide Stereo Panned Synths].
* **The "Humanizer" Hack:** [Room Tone], [Analog Tape Hiss], [Singer Breaths].
* **The "Structure" Hack:**
  + [Break]: Silence.
  + [Build]: Rising tension.
  + [Drop]: Maximum energy.

## **7. Workflow Stages**

1. **Concept Phase:** User Input -> Persona Selection -> Trend Check.
2. **Draft Phase (Text):** Lyrics generated -> Stress-Test -> Rhyme Audit.
3. **Prompt Phase:** JSON-to-Text compilation using "Black Magic" tags.
4. **Generation Phase:** Suno API/Web Interface.
5. **Audit Phase (Audio):** Demucs Separation -> Spectral Analysis -> Frisson Check.
6. **Action Phase:**
   * **Pass:** Save to Vault.
   * **Fail:** Trigger "Repair Loop" (e.g., "Boost Bass," "Simplify Lyrics").

# **Brutal Critique & System Refinement**

# **Auditor: Apex Lead Architect (Adversarial Mode)**

## **1. Critique of "Quantization" Logic**

The Issue: Agent Groove-Quant assumes that "Swing" is always desirable. In "Rage" or "Hyperpop" genres, robotic quantization is the aesthetic.

The Fix: We must implement a genre\_strictness parameter in genres.json.

\* If Genre == "Boombap", Strictness = Low (High Swing).

\* If Genre == "Hyperpop", Strictness = High (Zero Swing).

## **2. Critique of "Frisson" Detection**

The Issue: The Agent Frisson-Seeker looks for spectral shifts. However, frisson is often context-dependent (lyrical meaning + melody). Audio analysis alone is insufficient.

The Fix: We calculate a "Composite Frisson Score". It combines the Spectral Shift (Audio) with the Sentiment Arc (Lyrics). Frisson is most likely when a "Negative" lyric is sung over a "Major/Uplifting" chord progression (Semantic Dissonance).

## **3. Critique of "Stem Separation" Artifacts**

The Issue: Demucs is good, but Suno's "lo-fi" generations often result in "watery" artifacts when separated. This ruins the spectral analysis.

The Fix: We implement a "Confidence Gate". If the Demucs separation quality score is low (detected by high-frequency noise floor), we bypass detailed spectral analysis and fallback to "Global Mix" analysis to avoid false positives.

## **4. Critique of the "Persona" System**

The Issue: Static JSON personas are too rigid. They produce caricatures.

The Fix: We implement "Persona Drift". The agent should allow the persona to evolve slightly over the course of the song (e.g., The "Hard" rapper becomes "Vulnerable" in the Bridge). This increases the "Parasocial" score.

## **5. Critique of "Viral" Metrics**

The Issue: We are optimizing for past viral trends.

The Fix: Agent Trend-Jacker must prioritize Derivative Rate. We don't just look at what is trending now; we look at the rate of change. If BPMs are getting faster every month, we predict the future BPM, not the current one.

## **6. Missing Feature: "Vocal Texture"**

The Oversight: We analyze what is said, but not how it sounds (Timbre).

Action Item: Add agent\_timbre.py. It uses OpenL3 to classify vocal texture: "Raspy," "Clean," "Autotuned," "Whisper."

\* Rule: A "Sad" song requires "Breathiness" > 50%.

\* Rule: A "Club" song requires "Autotune" > 80%.

## **7. Next Steps for Implementation**

1. **Phase 1:** Build the config files (genres.json, personas.json).
2. **Phase 2:** Build the lyrical agents (agent\_flow.py, agent\_rhyme.py). These can be tested immediately with text.
3. **Phase 3:** Build the audio agents (agent\_spectral.py). This requires the Demucs pipeline setup.

# **Apex Music Engine - Dependencies**

# **Install via: pip install -r requirements.txt**

# **--- Core Audio Processing ---**

librosa>=0.10.0 # Audio feature extraction (Spectrograms, BPM)

numpy>=1.24.0 # Array manipulation

scipy>=1.10.0 # Signal processing (FFT)

soundfile>=0.12.0 # Audio file I/O

# **--- Source Separation (Optional but recommended for audio analysis) ---**

demucs>=4.0.0 # Facebook's Music Source Separation

torch>=2.0.0 # Required for Demucs

# **--- NLP & Linguistics ---**

phonemizer>=3.2.1 # IPA conversion (Requires 'espeak-ng' installed on OS)

nltk>=3.8.1 # Natural Language Toolkit (CMU Dict)

textstat>=0.7.3 # Readability scores

# **--- Utilities ---**

colorama>=0.4.6 # Pretty terminal output

pyloudnorm>=0.1.1 # LUFS Loudness metering