Overtone

Architecture of a Cross-Cultural Generative Music Engine in Rust

Simon Knight

Adelaide, Australia

March 2026 • Version 1.0.0

Last updated: March 15, 2026

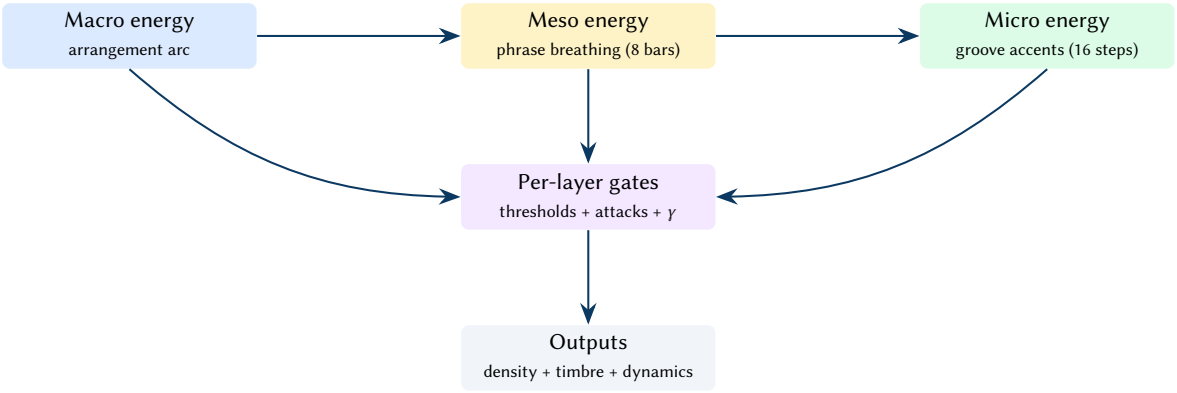
Abstract. Overtone is a terminal-based cross-cultural generative music engine written in Rust that produces full arrangements from declarative parameter configurations. The system combines manual oscillator synthesis, spectral sample analysis, and a hierarchical energy model to generate arrangement-aware music spanning Western electronic and Indian classical traditions in real time. This report presents the architecture of the synthesis pipeline, formalises the four-level energy model that drives pattern density and timbral evolution, describes the DSP chain (biquad filters, Schroeder reverb, ping-pong delay, sidechain compression), details the sample intelligence subsystem that classifies and blends audio samples with synthesised output, and documents the Ableton Live integration via a custom TCP/JSON bridge. On an Apple M2 workstation the engine renders a 64-bar stereo track in 18 ms—approximately 470× real time—while maintaining 64-bit internal precision throughout the signal path.

Contents

Recent Changes (March 2026 Update)

This edition also captures the most recent engineering work that landed after the initial report freeze:

- **DSP primitives expanded.** The signal toolbox now adds a ladder-style four-pole low-pass filter (section 9.2), a reusable envelope follower for sidechain/control extraction (section 9.3), an ultra-slow deterministic macro modulator for long-form movement (section 9.5), a mono-safe Mid/Side processor (section 9.4), and a reusable tanh saturator for bus glue (fig. 77).
- **Generator behaviour upgraded.** The render engine now supports chapter-level bass and hi-hat syncopation overrides (`src/engine/chapter.rs`), preserves Euclidean rhythm phase across bar boundaries (section 11.6), stores the effective bass wavetable position per bar, and applies deterministic macro modulation to both bass timbre and ambience (section 9.5).
- **Low-end and master-bus processing improved.** The former full-band bass ducking stage has been replaced with split-band sidechain ducking so the kick clears sub energy without collapsing the bass upper harmonics. The master stage can now apply optional Mid/Side widening with mono-safe lows, followed by controlled soft saturation.
- **Live control surfaces broadened.** The CLI, native GUI, and TUI all expose the new filter-mode selection (section 16.4), macro-mod toggle, Mid/Side enable switch, and associated parameters; the playback path also supports live auditioning of the selected filter mode during output processing.
- **Harmonic planning and tension architecture.** A new multi-dimensional tension model (section 12) operates independently of the energy model, controlling chord voicing (section 12.2), per-voice harmonic/rhythmic response (section 12.3), cadential patterns (section 12.4), key modulation planning (section 12.5), voice leading (section 12.6), syncopation–tension coupling (section 12.7), resolution hierarchy (section 12.8), harmonic rhythm (section 12.9), register/tessitura (section 12.10), textural density (section 12.11), metric displacement (section 12.12), and pedal tones (section 12.13).
- **Composition Studio designed.** The native GUI gains a phrase-level composition environment (section 13) with a drag-and-drop phrase palette (section 13.1), context-aware theory advisor with interactive chord map (section 13.2), real-time voice constellation (section 13.3), five-level zoom timeline, Live Jam Mode with gesture recording (section 13.4), and phrase-level editing transforms (section 13.5).
- **Theme and chapter workflow refined.** Built-in themes now populate chapter metadata with explicit bass/hi-hat syncopation defaults, and the chapter editors in both interfaces can override those rhythmic relationships locally. This makes timeline chapters a stronger unit of arrangement identity rather than just a bar-range wrapper.
- **Ableton and web reporting improved.** Clip pushes now name target tracks using both element and mood, which reduces ambiguity in mood-specific track groups in Live. The Axum/WebSocket server and React UI now surface Ableton connection state, track counts, missing-track diagnostics, and user intents for the node-based web control surface (section 21.2.2).
- **Visualisation architecture designed.** A new section (section 19) catalogues seven visual modes—spectral waterfall, mood space navigator, harmonic series tree, rhythmic phase wheel, motif lineage graph, tension heatmap, and energy landscape—with a unified render pipeline and colour-blind safe palettes (section 19.5).
- **UX workflow patterns formalised.** Six workflow modes (section 20)—top-down, bottom-up, hybrid, performative, teaching, and iterative—are defined as a lossless state machine with A/B comparison, branching undo tree, and version snapshots.
- **Sound design workflow added.** Macro-knob parameter mapping, preset inheritance trees, A/B patch comparison, bounded randomisation, automation recording, and global timbral coherence controls are documented in a new Composition Studio subsection (section 13.6).



Conceptual point. Overtone treats "energy" as a multi-scale control signal: arrangement-level intent (macro) is shaped by phrase breathing (meso) and groove accents (micro), then lowered into per-layer gating that directly controls pattern density and timbral brightness.

Figure 1. Energy as a multi-scale control signal. This is the organising abstraction that connects high-level intent to low-level synthesis and pattern decisions.

- **Middle Eastern and African expansion (Research).** New sections on the Maqam modal system (including 24-TET and Jins structures, section 3.8.1), Iqa'at rhythmic cycles (section 3.8.1), and West African polyrhythmic branching (including bell timelines and talking drum pitch-contour language, section 3.8.2) have been added to the formal research foundation.
- **Genre DNA and composition archetypes.** Genre profiles are formalised as interpolable 8-D vectors (section 21.5) with historical era emulation. Six composition archetypes (section 21.6) provide named structural contracts for energy and tension envelopes.
- **Set building and export ecosystem expanded.** Multi-track set building UX (section 21.7) with crowd energy simulation and rehearsal mode; export pipeline (section 21.8) targeting Rekordbox, Eurorack CV, VJ sync, podcast chapters, and streaming platform metadata.

Part I — Introduction

1 Introduction

Psytrance is a subgenre of electronic dance music characterised by driving kick drums at 140–150 BPM, hypnotic bass lines built from filtered sawtooth oscillators, layered percussive textures, and carefully sculpted energy arcs that build tension and release over multi-minute arrangements. Producing a psytrance track in a conventional digital audio workstation (DAW) requires manually programming drum patterns, designing filter sweeps, tuning sidechain compression, and arranging dozens of clips across a timeline—a process that demands both musical knowledge and significant technical skill.

This report describes *Overtone*, a Rust-based generative music engine that automates the core production workflow. The user specifies high-level parameters—tempo, mood, key, energy curve—and the engine generates a complete stereo mix with kick, bass, hi-hat, clap, lead, and sample layers, all shaped by a hierarchical energy model that controls pattern density, filter modulation, and sample placement across the arrangement.

The system is designed around three principles:

1. **Energy-driven composition.** A four-level energy model (macro curve, meso breathing, micro groove, sub-channel gating; see fig. 1) determines *what* plays *when* and *how loudly*, replacing manual arrangement with a declarative energy specification.
2. **Hybrid synthesis.** Each musical element can be rendered from manual oscillators, from classified audio samples, or from a configurable blend of both—adapting the timbral palette to the energy state of the arrangement.
3. **DAW integration.** MIDI export/import and real-time clip pushing via a custom TCP/JSON bridge to a purpose-built Ableton Control Surface allow the generator to function as a compositional front-end for professional production workflows.

The remainder of this report is organised in five parts. **Part I** (this introduction) motivates the project. **Part II** (section 2–section 4) presents the musical foundations: Western electronic music theory, Indian classical

theory (including the extended eastern traditions, rhythmic devices, and implemented theory features), and the musical lineage that connects these traditions. **Part III** (section 5) describes the nine instruments in Overtone's palette—their musical character, timbral qualities, and roles in the arrangement—without reference to synthesis implementation. **Part IV** (section 6–section 15) details the implementation: system architecture, the energy model, synthesis engines, DSP processing, sample intelligence, pattern generation, the streaming engine, and session management. **Part V** (section 16–section 22) covers the workflow and integration layer: the terminal user interface, Ableton integration, evaluation, and future directions.

Part II — Musical Foundations

2 Theoretical Foundations

Overtone is built upon two distinct yet complementary musical traditions: Western electronic dance music (specifically psytrance) and Indian classical music (both Hindustani and Carnatic). This section details the theoretical frameworks from both traditions and explains how they are integrated within the engine.

2.1 Music Theory Primer for Parameterised Generation

The generator exposes musical concepts as user-facing parameters (key, scale, mood, energy curve) rather than as low-level note events. Consequently, a compact, implementation-oriented music theory vocabulary is useful to explain how scale choices, rhythmic emphasis, and harmonic density interact with the engine.

Pitch, Key, and Scales All pitched content is derived from a root pitch class (key) and a scale defined as semitone offsets within the octave. A scale thus becomes a constraint function: it filters candidate MIDI pitches (or oscillator frequencies) down to a musically coherent subset.

In Western terms, common scale families include major (Ionian), natural minor (Aeolian), and modal rotations (Dorian, Phrygian, Mixolydian). In eastern terms, the project also includes raga-coloured interval sets that emphasise characteristic seconds and sixths. Because the engine operates primarily in 12-TET, these are approximations; microtonal extensions (shruti offsets) are treated as future work (section 3).

Degrees, Stability, and Nyasa The system models pitch selection primarily in terms of scale degrees. This is compatible with both modal western usage and raga-based melodic grammar. A practical implementation benefit is that stability can be encoded as a simple weighting curve over degrees: at low energy the engine prefers stable tones (root, fifth) and nyasa-like landings; at higher energy it introduces more colour tones (second, fourth, sixth) and wider register.

In the raga-oriented features, *nyasa* is treated as a cadence target instead of a strict rule system: the harmony layer can hold a nyasa tone for a bar (`src/engine/harmony.rs`), and melodic generators can bias phrase endings toward vadi/samvadi when they are annotated on the active scale.

Harmony and Implied Progression Psytrance often relies on implied harmony rather than functional chord progressions: bass + lead motion can suggest a tonal center even when no triads are voiced. Within the engine, harmony is therefore modelled as a lightweight layer:

- A scale constrains pitch selection.
- A degree-weighting profile biases landings toward stable tones (root, fifth) at low energy and toward colour tones (second, fourth, sixth) at higher energy.
- Energy can modulate harmonic density by enabling additional voices or widening pitch ranges.

This approach is compatible with both Western modes and raga-inspired sets, and avoids over-specifying chord symbols that are uncommon in the genre.

Rhythm, Meter, and Accent The system renders onto a fixed 4/4 transport grid with 16-step subdivision, but rhythm is perceived primarily through accent patterns. Classic EDM emphasis (beats 1 and 3) is augmented by syncopation styles and by tala-derived accent cycles (section 3). As a result, the engine can create the illusion of longer cycles (e.g., 7 or 10) while remaining compatible with DAW timelines and MIDI export.

Mode Rotation (Murchhana / Graha Bhedam) Beyond choosing a scale, the generator supports deriving *related* scales via mode rotation: selecting a new starting degree (graha) and re-normalising the resulting

interval set to a zero-root representation. This is implemented in `src/theory/murchhana.rs` and is applied via an `EffectiveKey` abstraction so the rest of the engine can remain degree-based.

Musically, murchhana provides a controlled way to evolve harmonic colour over time while keeping the rhythmic surface stable. It can be applied as a section boundary device (e.g., at bar 17) or as a longer journey in combination with raga modulation.

Form and Energy Arrangement is expressed as an energy function over bars. Macro energy selects sections (intro, build, peak, breakdown), meso energy creates breathing, and micro energy controls per-hit density. Music theory concepts connect directly to this model:

- Lower energy favours fewer notes, narrower ranges, and stronger tonic anchoring.
- Higher energy supports faster note rates, wider ranges, and more dissonant colour tones.
- Cadences (e.g., tihai-like resolutions) provide audible section boundaries even without explicit chord changes.

2.2 Psytrance Theory: Rhythmic Drive and Energy Arcs

Psytrance is fundamentally a music of *texture* and *momentum*. Unlike functional harmony which relies on chord progressions, psytrance derives its narrative from the evolution of timbre over a static tonal center.

2.2.1 The Kick and Bass Relationship

The "engine room" of any psytrance track is the phase-coherent interplay between the kick drum and the bassline. The kick provides a steady quarter-note pulse (~140–150 BPM), while the bass occupies the 16th-note subdivisions between the kicks. The most common rhythmic relationship is the *offbeat* or *triplet* bass:

- **Offbeat:** Bass hits on every 16th note *except* the downbeat (e.g., [Kick, Bass, Bass, Bass]).
- **K&B Syncopation:** Sidechain compression is critical here; the bass signal must "duck" (attenuate) when the kick triggers to prevent low-frequency masking and speaker clipping.

2.2.2 Psytrance Sub-genres and Rhythmic Character

While the core kick-and-bass relationship is universal, different sub-genres of psytrance employ distinct rhythmic and timbral strategies.

- **Full-On:** High-energy, melodic, and fast (~145 BPM). Characterised by "rolling" basslines (three 16th notes per beat) and frequent melodic hooks.
- **Darkpsy / Forest:** Higher tempos (150 BPM+) with a focus on organic, eerie soundscapes and complex, glitchy percussion. The bass is often more syncopated or "galloping."
- **Progressive:** Slower tempos (135–140 BPM) with a focus on deep, atmospheric grooves and minimal melodic changes. Employs "offbeat" bass patterns for a spacious feel.
- **Hi-Tech:** Extreme tempos (160–200 BPM) with ultra-dense percussive patterns and rapid-fire synth stabs.

Figure 2. Comparison of foundational rhythmic patterns across psytrance sub-genres. *K* indicates a kick drum hit; *B* indicates a bass note. Full-On relies on a continuous 16th-note roll, Progressive creates space with offbeats, and Forest uses a syncopated gallop.

2.2.3 Energy-Driven Arrangement

Structure in psytrance is defined by *energy density*. Tracks typically follow a "Long Journey" arc:

1. **Intro:** Minimal percussion, atmospheric pads, drone elements.
2. **Build:** Progressive addition of percussion layers (hi-hats, claps) and increasing filter brightness.
3. **Drop/Peak:** Full rhythmic density, aggressive leads, maximum energy.
4. **Breakdown:** Sudden removal of rhythmic drive, focus on melodic motifs and atmospheric textures.

2.3 Indian Classical Theory: Raga and Tala

Indian classical music is built on two pillars: *Raga* (melodic mode) and *Tala* (rhythmic cycle).

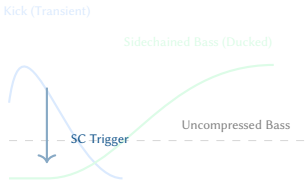


Figure 3. Phase-coherent sidechain alignment. To prevent frequency masking in the critical 40–100 Hz region, Overtone's sidechain compressor attenuates the bass signal during the first 10–20 ms of each kick hit, creating the "pumping" sensation foundational to the genre.

2.3.1 Raga: The Melodic Framework

A raga is more than a scale; it is a "color" or "mood" with specific grammatical rules.

- **Aroh/Avroh:** The specific paths for ascending and descending. Some notes may be skipped in ascent (vakra) but included in descent.
- **Vadi and Samvadi:** The "King" and "Queen" notes. These are the most important degrees of the raga, where phrases typically land or resolve.
- **Pakad:** A characteristic melodic phrase that uniquely identifies the raga.

2.3.2 Tala: The Rhythmic Framework

A Tala cycle is organised into a clear hierarchy of temporal units.

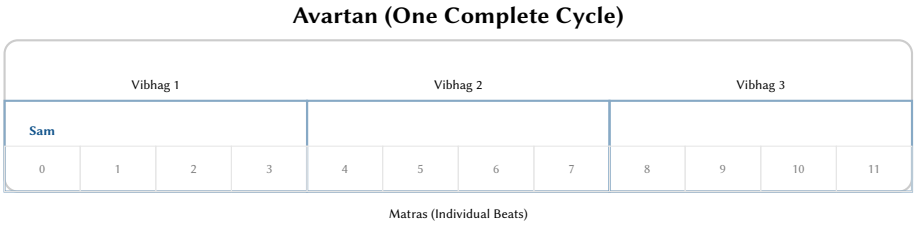


Figure 4. The temporal hierarchy of a Tala. Individual beats (*matras*) are grouped into sections (*vibhags*), which together form the full cycle (*avartan*). Overtone uses this hierarchy to determine where to place accents and rhythmic flourishes.

2.3.3 Additive vs. Divisive Rhythm

Unlike Western music, which is primarily *divisive* (dividing a whole note into halves, quarters, etc.), Indian rhythm is *additive*—constructing larger cycles from smaller units of time.

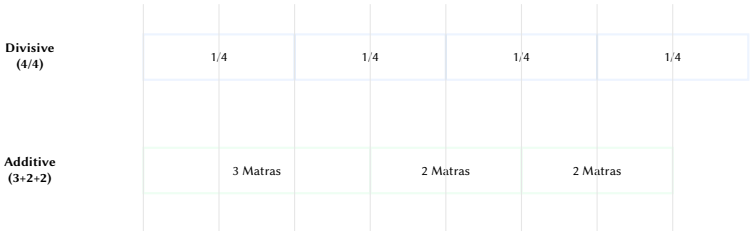


Figure 5. Additive rhythm construction. While Western 4/4 meter is built by dividing a bar into four equal parts, Indian Talas like Rupak are built by adding unequal sections (3+2+2) together. Overtone uses this additive logic to generate asymmetric syncopations.

Tala is a cyclical time system. A cycle (*avartan*) is divided into beats (*matras*), which are grouped into sections (*vibhags*).

- **Sam:** The first beat of the cycle, and the point of ultimate resolution.
- **Khali:** The "empty" beat, typically indicated by a wave of the hand, providing a structural counterpoint to the emphasized beats (*tali*).

2.3.4 Gamaka: Microtonal Ornamentation

Identity in raga music is carried by *gamakas*—ornaments that move between or around the notes.

- **Meend:** A continuous, expressive glide (portamento) between two notes.
- **Andolan:** A slow, gentle oscillation around a note.
- **Kampita:** A faster vibrato-like oscillation.

2.4 Philosophical Foundations: Nada Brahma

The engine's approach to sound is grounded in the ancient concept of *Nada Brahma* ("The world is sound"). In this worldview, the universe is not composed of matter, but of vibrations.

> **Design Decision 1: Nada Brahma in Overtone**
>
> This philosophy manifests in Overtone through the **Tanpura Drone**. The drone is not merely an accompaniment; it is the *prana* (breath) of the track. It represents the unmanifested potential from which all melody and rhythm emerge. In the engine, the drone is the only layer that is never truly silent, even at zero energy, providing a vibrational constant that anchors the listener.

2.5 Aesthetic Framework: The Navarasa

The emotional character of Overtone's output is guided by the *Navarasa*—the nine "juices" or aesthetic essences defined in the *Natya Shastra*. Rather than simple "Happy" or "Sad" binary moods, Overtone uses these nuanced states to bias its generative parameters.

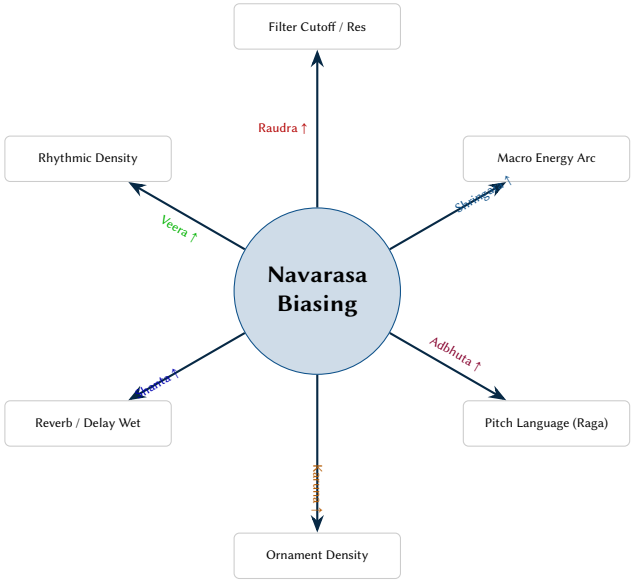


Figure 6. Rasa-to-Parameter mapping. The nine aesthetic states act as high-level constraints that bias lower-level synthesis and generative knobs. For example, *Raudra* (Fury) biases the engine toward high filter resonance and distortion, while *Shanta* (Peace) increases spatial wetness and reduces rhythmic density.

2.6 Cross-Cultural Synthesis in Overtone

Overtone bridges these worlds by mapping the grammatical rules of Indian classical music onto the 16-step grid and energy-driven arrangement of electronic music.

1. **Tala as Syncopation:** Rather than using complex time signatures, Overtone uses Tala cycles (e.g., Rupak 7, Jhaptal 10) to generate *accent patterns* across the 4/4 grid.
2. **Energy as Rasa:** The energy model is mapped to the *Navarasa* (nine aesthetic emotions). Low energy maps to *Shanta* (peace), while high energy maps to *Veera* (heroism) or *Raudra* (fury).
3. **Yoga-Flow Energy:** Arrangement arcs are modeled after *Yoga Asana* sequences—starting with grounding (Muladhara), building heat through the solar plexus (Manipura), and concluding in meditative dissolution (Sahasrara).

3 Eastern Musical Traditions in Electronic Music

While the initial design targeted classic Western modes and 16-step psytrance syncopation, the architecture generalises cleanly to other musical traditions. This project includes raga-inspired scales and tala-inspired accent cycles that can be applied without changing the global time signature.

3.1 Specific Raga Profiles and Their Applications

While all 72 melakarta scales are available, the generator provides several hand-crafted raga-inspired profiles used in the mood presets:

- **Bhairav (Hindustani):** Characterized by flattened 2nd (Re) and 6th (Dha) degrees. It creates a solemn, dawn-like character, making it effective for meditative intros and "Dark Phrygian" style psytrance.

- **Ahir Bhairav (Hindustani):** Combines the Bhairav lower tetrachord with a more major-sounding upper tetrachord. It is bright yet devotional, ideal for progressive melodies and "Goa Sunrise" themes.
- **Todi (Hindustani):** An intensely chromatic and tense raga, used to create maximum psychedelic tension during energy build-ups.
- **Malkauns (Hindustani):** A pentatonic (five-note) raga of deep introspection and late-night calm. Its sparse structure is ideal for ambient drones and "Samay" midnight presets.
- **Charukesi (Carnatic):** A lyrical and highly emotional raga, used for progressive themes that require a balance of melodic warmth and mystic color.

3.2 Tala Accent Cycles and Rhythmic Structures

Indian classical rhythm (*tala*) is modeled as a cyclic system of accentuation. Rather than using complex time signatures, Overtone applies tala-derived velocity multipliers onto the 16-step grid.

(a) Rupak Tala cycle. **(b)** Jhaptal Tala cycle.

Figure 7. Cyclic representations of Rupak and Jhaptal Talas. These cycles define the underlying "pulse" onto which Overtone superimposes its 16-step electronic patterns, creating complex cross-rhythms.

The relationship between the cyclic tala and the linear DAW grid is a source of "polyrhythmic" interest.

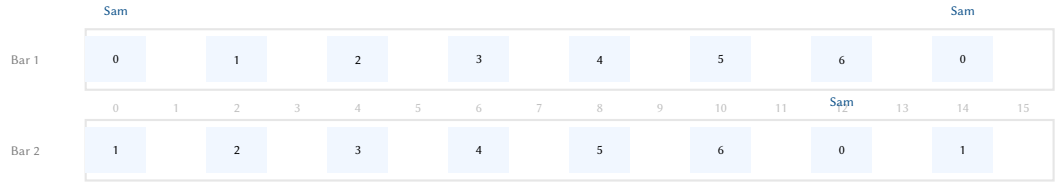


Figure 8. Tala-to-grid alignment. A 7-beat Rupak cycle (mapped here to 8th-note steps) drifts relative to the 16-step bar boundary. The *Sam* (beat 0) moves from step 0 in Bar 1 to step 14 in Bar 1, and step 12 in Bar 2, creating evolving syncopation.

Rather than using complex time signatures, Overtone applies tala-derived velocity multipliers onto the 16-step grid:

- **Rupak (7 beats):** Often interpreted as $[3, 2, 2]$. Its syncopation against the 16-step grid creates an evolving cross-rhythmic pattern that resolves every 7 beats.
- **Jhaptal (10 beats):** interpreted as $[2, 3, 2, 3]$. It creates a steady yet complex groove suitable for mid-tempo progressive tracks.
- **Ektal (12 beats):** Often used in fast-tempo compositions, providing a high-density rhythmic drive.

3.3 Indian Classical Structural Devices

Beyond interval sets, the generator implements several structural devices to maintain authentic formal development:

Figure 9. Tihai as a generative constraint. The generator selects motif and spacing so the third repetition resolves on the cycle downbeat (sam).

- **Aroh and Avroh:** Many ragas have different paths for ascending (*aroh*) and descending (*avroh*). The melodic generator maintains a directional traversal state and selects pitch candidates based on these paths.

- **Vadi and Samvadi:** The "King" and "Queen" notes are emphasized through a weighting bias in the pitch selection algorithm, ensuring the melodic center of the raga is properly established.
- **Gamakas (Ornaments):** The system can attach `GamakaSpec` metadata to notes (meend, andolan, kampita), which the internal synthesis engines interpret as per-sample frequency or amplitude modulations.
- **Tihai:** A cadential device where a phrase is repeated three times to resolve on the downbeat (*sam*). The engine can generate a tihai as a final bar-override before a section boundary (e.g., from Build to Peak).
- **Sangati:** Repetition with progressive elaboration. Each time a phrase is repeated across multiple bars, the engine can slightly increase the melodic density or add ornaments to mimic the live development of a composition.
- **Jugalbandi:** A call-and-response duet mode where two synthesis layers (e.g., Lead and FM String) exchange phrases, with the "respondent" layer varying the "caller's" motif.

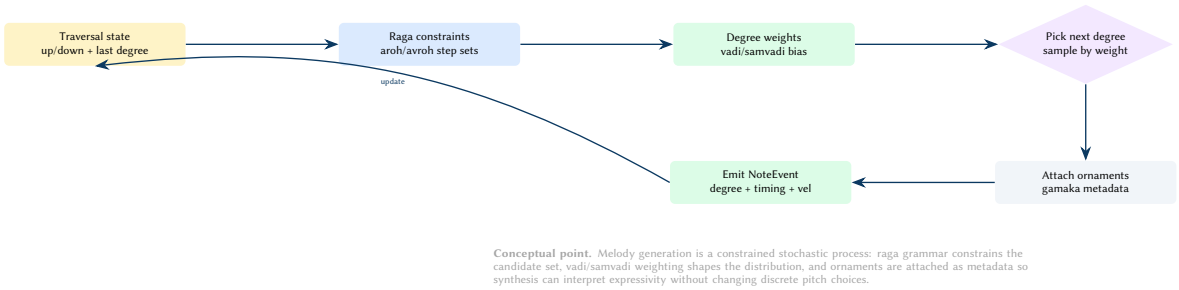


Figure 10. Raga-aware pitch selection as a decision DAG. Constraints define what is legal, weights define what is likely, and ornaments carry expressive motion as metadata.

3.4 Yoga Flow and Rasa-to-Energy Mapping

Overtone extends the traditional Western arrangement by mapping the energy model to the aesthetic and philosophical frameworks of Indian music:

1. **Navarasa:** The "Nine Emotions" framework. The generator can bias parameters based on a selected rasa:
   - **Shanta (Peace):** Lower energy, slow breath LFO, drone-dominant.
   - **Veera (Heroic):** Rising energy, sharp attacks, Ahir Bhairav scale.
   - **Adbhuta (Wonder):** Moderate energy, high filter resonance, random walk arpeggios.
2. **Chakra Architecture:** Arrangement arcs are modeled after *Yoga Asana* sequences. The "Yoga Flow" energy curve moves from Muladhara (grounding/intro) to Manipura (strength/peak) and then to Sahasrara (transcendence/cool-down).

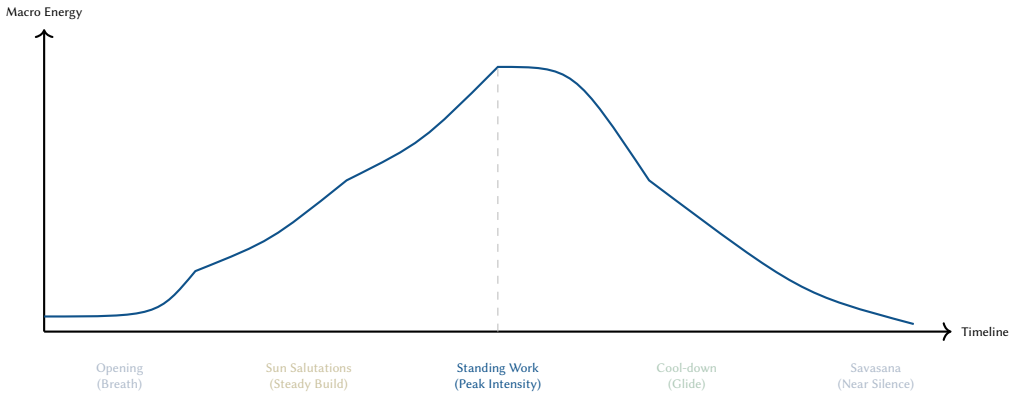


Figure 11. The 7-phase "Yoga Flow" macro energy curve. Unlike the multi-peak "Long Journey" EDM curve, this arc represents a continuous, single-peak meditation structure, moving smoothly from grounding (Muladhara) to high-energy effort (Manipura), resolving in peaceful dissolution (Sahasrara).

3.5 Motif Foreshadowing via Sub-Channel Preview

Sub-channel gating includes a preview window below the main activation threshold. In this region elements can appear as sparse "ghost" hits, hinting at future motifs before the full density arrives.

The result is a stronger sense of narrative continuity: motifs are introduced, developed, and then brought into the foreground.

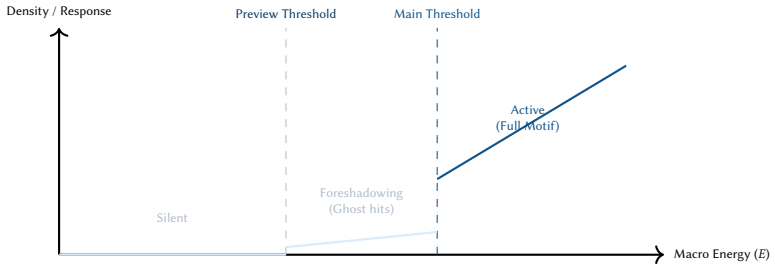


Figure 12. The Sub-Channel Preview mechanism. Instead of an abrupt binary switch when an element becomes active, the engine provides a "preview window." In this energy band, the element emits occasional low-velocity hits, telegraphing its arrival to the listener before full activation.

3.6 Drone, Ornamentation, and Cadential Devices

Beyond interval sets and accent cycles, Indian classical identity is often carried by a small set of structural devices:

- **Tanpura drone** — a continuous pitch reference (Sa–Pa–Sa) that anchors intonation and fills sparse sections. The engine can render a four-string drone layer tuned to the active key and mixed inversely with macro energy.
- **Aroh/avroh constraints** — ragas are not only scales; they encode preferred ascending and descending motion. A raga-aware melody generator uses directional step sets and avoids stepwise motion that contradicts the raga's character.
- **Vadi/samvadi weighting** — note hierarchy influences phrase landing points and perceived emotional center. The melodic generator can weight pitch selection toward the vadi (primary) and samvadi (secondary) degrees.
- **Gamaka** — ornaments such as meend (glide), kan-swar (grace approach), and andolan/kampita (oscillation) provide micro- expression that distinguishes ragas even in 12-TET approximations.
- **Tabla bols and articulation** — percussive syllables (e.g., Dha, Tin, Ghe) encode stroke families and resonance. The tom layer can attach articulation metadata to events and render different envelopes/spectra per stroke.
- **Tihai and korvai** — cadential patterns that resolve on sam (downbeat). These devices can be inserted at section boundaries to increase formal coherence and create a clear sense of arrival.

3.7 Time, Notation, and Breath

Two additional ideas extend the system beyond purely pattern-driven EDM generation:

- **Samay** — time-of-day associations can bias raga selection toward traditional prahar groupings.
- **Sargam + bols in the TUI** — displaying Sa/Re/Ga/Ma or bol initials in the step grid provides a musically meaningful view of eastern-mode patterns.
- **Breath-synced modulation** — a slow asymmetric LFO can modulate global dynamics and timbre (filter cutoff, wetness), and reduce high-frequency density during exhale to support meditative outputs.

3.8 Middle Eastern and African Musical Foundations

The expansion into Middle Eastern and West African traditions integrates the *Maqam* modal system, the *Iqa'at* rhythmic cycles, and the complex polyrhythmic layering of West African traditions.

3.8.1 Maqam and Iqa'at: Arabic Generative Theory

Maqamat utilize a 24-tone equal temperament (24-TET) grid, where each step is 50 cents. This allows for the "half-flat" (*sikah*) and "half-sharp" intervals that define the tradition's melodic character.

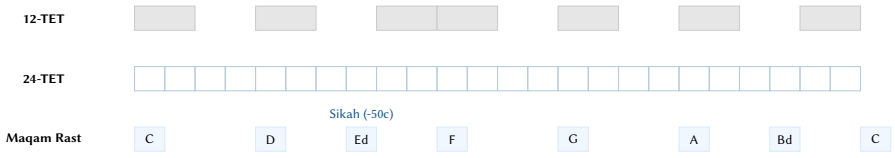


Figure 13. Maqam Rast on the 24-TET grid. The third (Ed) and seventh (Bd) degrees are "half-flats," sitting exactly between the minor and major intervals of the Western 12-TET scale.

Jins Modulation (Tetrachord Switching) Maqamat are constructed from overlapping tetrachords called *ajnās*. Modulation occurs by switching the upper jins while maintaining the lower jins (Jins al-Asl).

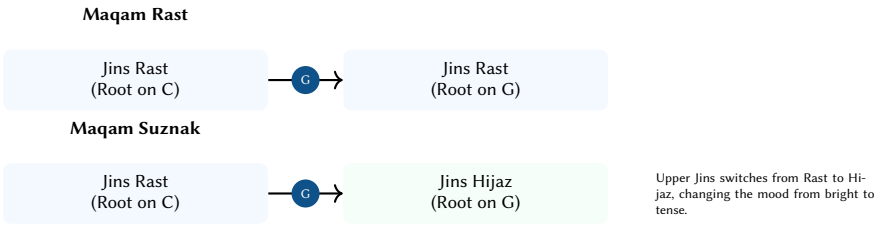


Figure 14. Jins modulation from Maqam Rast to Maqam Suznak. Both share the same lower Jins (Rast on C), but the upper Jins switches at the pivot note (G).

Iqa'at: Rhythmic Masks Middle Eastern rhythms (*iqa'at*) are defined by sequences of *Dum* (D) and *Tak* (T) strokes.

Figure 15. The Maqsum iqaa (4/4). The "Dum" (D) maps to the kick drum with a low-pass profile, while the "Tak" (T) maps to the snare/clap with a high-pass profile.

3.8.2 West African Polyrhythms

The expansion into West African traditions focuses on the interplay of independent rhythmic streams and the concept of the "bell timeline" as a master clock.

Bell Timelines (Master Clocks) The bell pattern (*Gankogui*) provides the reference pulse for all other layers. Unlike the Western downbeat, the bell often uses asymmetric, non-divisive cycles.

Figure 16. The Agbadza bell timeline (12/8). This asymmetric pattern acts as the "master clock" to which all other polyrhythmic layers are phase-locked.

Stream Phasing (3-over-2 and 4-over-3) Polyrhythms are implemented as independent temporal streams (n over m).

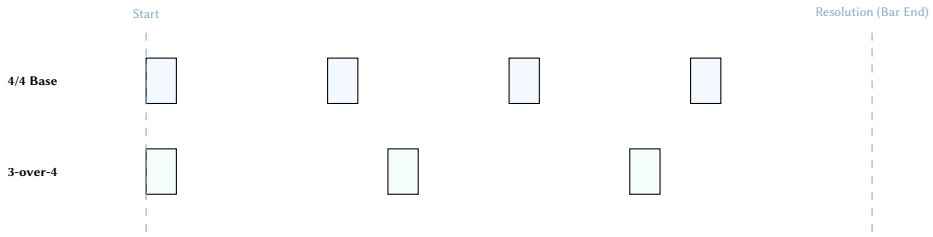


Figure 17. 3-over-4 polyrhythm phasing. The 3-beat stream phases against the 4/4 base, resolving exactly at the bar boundary. This creates cyclical tension that is mathematically predictable but rhythmically complex.

Talking Drum: Pitch-Contour Language Talking drums mimic tonal languages by varying pitch during a stroke.

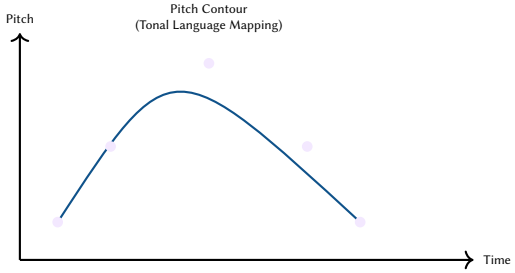


Figure 18. Talking drum pitch-contour language. The synthesis engine sweeps the pitch of the percussive strike following a linguistic contour (High/Mid/Low), allowing the drum to "talk."

3.9 Global Textures: Aksak, Ombak, and Ma

Further expansion into global traditions (section 3.9) addresses the additive rhythms of the Balkans, the shimmering acoustics of Indonesia, and the aesthetic silence of Japan.

3.9.1 Balkan Aksak: Additive Meters

Aksak rhythms utilize additive cells of unequal length (e.g., 2+3).

Figure 19. Aksak 7/8 rhythm. The additive structure (2+2+3) creates the characteristic "limp" or "aksak" feel of Balkan dance music.

3.9.2 Indonesian Ombak: Shimmer Tuning

Ombak is achieved by detuning paired instruments to create acoustic beating.

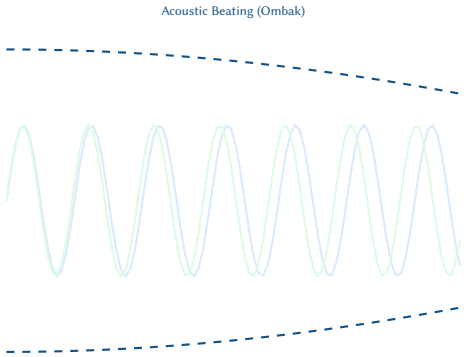


Figure 20. Ombak shimmer tuning. Two waves with a slight frequency offset (e.g., 10 cents) create a slow interference pattern (beating), providing a shimmering, "living" texture.

3.9.3 Japanese Ma: Contextual Silence

The concept of *Ma* is implemented as a generative silence engine that calculates the necessity of "gaps" based on preceding complexity.

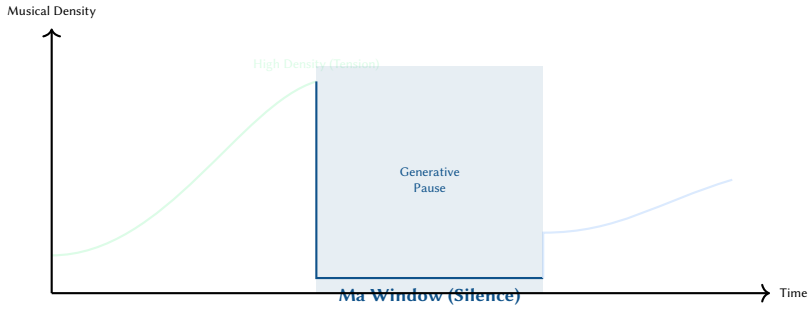


Figure 21. Japanese Ma (Negative Space). Following a peak in musical complexity and tension, the generative engine forces a structured silence window, providing aesthetic contrast before resuming.

3.10 Scientific and Bio-Acoustic Foundations

To support functional audio use cases (e.g., meditation, focus), Overtone integrates features based on the natural harmonic series and psychoacoustic principles (section 3.10).

3.10.1 Spectralism: The Natural Harmonic Series

Spectral scales are derived from the integer multiples of a fundamental frequency ($f = f_0 \cdot n$), providing a mathematically "pure" harmonic resonance.

3.10.2 Binaural Beats: Psychoacoustic Entrainment

Binaural beats are perceived when slightly different frequencies are presented to each ear ($f_{beat} = |f_L - f_R|$).

3.10.3 Genetic Modal Interpolation (Hybridization)

Musical DNA hybridization performs set-theory operations on modal intervals to create new, culturally-blended scales.

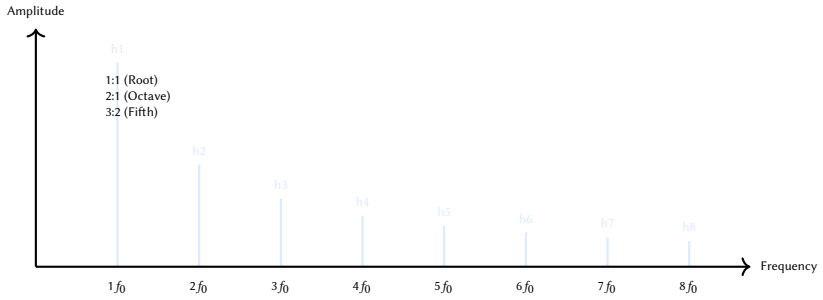


Figure 22. The Natural Harmonic Series. Spectralism utilizes these integer-ratio intervals to achieve a state of high acoustic consonance and resonance.

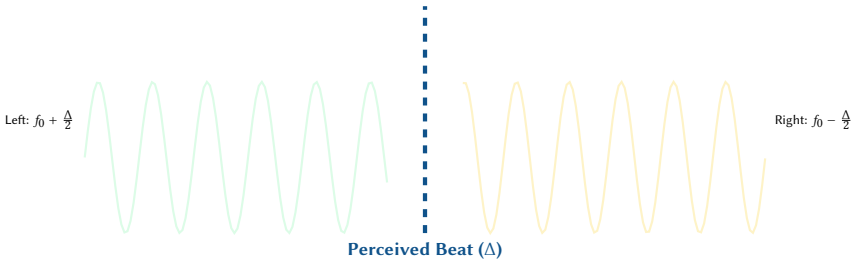


Figure 23. Binaural beat generation. By offsetting the L/R frequencies, the engine induces a phantom "beat" frequency in the listener's brain, supporting brainwave entrainment (e.g., Theta for meditation).

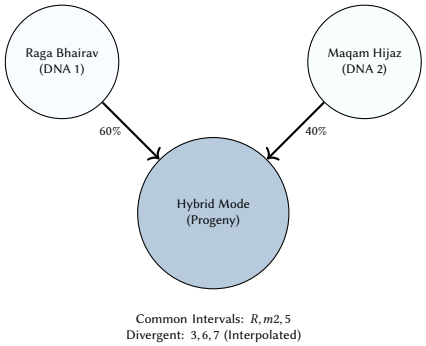


Figure 24. Genetic Modal Hybridization. The engine calculates the shared and divergent intervals between two parent traditions to produce a mathematically coherent hybrid mode.

3.11 Indian Classical Performance Forms

Both Hindustani and Carnatic traditions follow structured performance arcs that progressively introduce melodic and rhythmic elements.

3.11.1 Hindustani Performance Form: Alap–Jor–Jhala–Gat

A typical Hindustani performance begins with a slow, non-metered introduction and builds toward a fast, rhythmic climax.

- **Alap:** A free-form, meditative introduction where the artist explores the raga's personality. In Overtone, this is mapped to the Intro section with drones and sparse pads.
- **Jor:** A steady pulse is introduced, but without a fixed rhythmic cycle (tala). Melodic density increases.
- **Jhala:** Rapid, rhythmic strumming or striking toward the end of the Alap/Jor section, creating a sense of anticipation.
- **Gat:** The main rhythmic composition begins, accompanied by a tala. This maps to the Build and Peak sections of the EDM arrangement.

3.11.2 Carnatic Performance Form: Alapana–Kriti–Tani

Carnatic music emphasizes composed devotional songs (*kriti*) and virtuosic improvisation.

- **Alapana:** The melodic introduction (similar to Alap).
- **Kriti:** The main composed piece, often with three sections: *Pallavi* (refrain), *Anupallavi* (second section), and *Charanam* (final section).
- **Niraval and Kalpanaswaram:** Improvisation based on a line of the kriti or on solfege (*sargam*).

- **Tani Avartanam:** A dedicated percussion solo section, often appearing toward the end of a performance. Overtone implements this as a percussion-only breakdown.

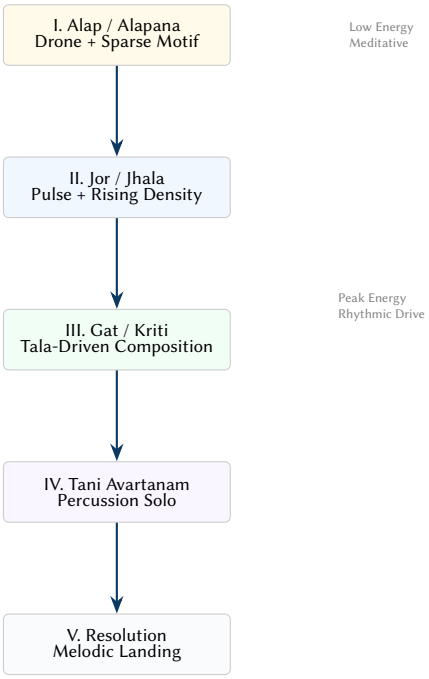


Figure 25. The performance arc implemented in Overtone. The engine maps the traditional Indian classical progression onto the EDM energy model, moving from non-metered melodic exploration to high-density rhythmic cycles. Figure 47 shows the detailed feature gating per phase.

3.12 Cross-Cultural Fusion: The Raga-Trance Logic

Overtone's unique contribution is the algorithmic fusion of these styles. The "Raga-Trance" logic is implemented as follows:

1. **Structural Mapping:** The Hindustani *Gat* phase is mapped to the psytrance "Drop." The transition from Alap to Gat mirrors the EDM "Build."
2. **Rhythmic Interplay:** Tala-derived accent cycles are superimposed on the four-to-the-floor kick foundation. This creates a "polyrhythmic" feel within a stable 4/4 timeline.
3. **Timbral Blending:** Synth leads use *Meend* (glide) and *Kampita* (vibrato) modulations to sound like a Sitar or Veena while maintaining the sonic weight of an acid bassline.

The `theory/melakarta` module implements the complete Carnatic 72-melakarta system as a generative lookup table. Each melakarta is defined by its canonical number (1–72), name, seven-note interval set, chakra group, and tonal character description.

3.12.1 Hindustani Thaat System

While the Carnatic tradition uses the exhaustive 72-melakarta system, the Hindustani tradition (North India) typically uses a subset of ten foundational scales called *thaats*. Overtone implements these as presets for quick access to traditional Hindustani colors.

The ten thaats are: Bilawal, Kalyan, Khamaj, Bhairav, Bhairavi, Asavari, Todi, Purvi, Marwa, and Kafi. Each corresponds to a specific set of *komal* (flat) and *tivra* (sharp) notes.

The 72 melakartas are derived by combining variations of the lower tetrachord (*purvanga*) and the upper tetrachord (*uttaranga*).

The 72 melakartas are generated combinatorially from three independent choices:

1. **Ma (madhyama)** — 2 options: shuddha (5 semitones) or prati (6 semitones).
2. **Ri–Ga pairs** — 6 combinations from $\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$ semitones.
3. **Da–Ni pairs** — 6 combinations from $\{(8,9),(8,10),(8,11),(9,10),(9,11),(10,11)\}$ semitones.

The total $2 \times 6 \times 6 = 72$ ragas are indexed into 12 chakras (6 ragas per chakra) via chakra $= \lfloor (n-1)/6 \rfloor$. Nine notable ragas with hardcoded personality descriptions are highlighted for UI display: Kanakangi ("mysterious, tense"), Mayamalavagowla ("grand, devotional"), Natabhairavi ("pathos"), Kharaharapriya ("versatile, deep"), Harikambhoji ("bright, joyful"), Dheerasankarabharanam ("complete, bright"), Chalanata ("alien, otherworldly"), Kamavardhini ("intense, dramatic"), and Mechakalyani ("serene, luminous").

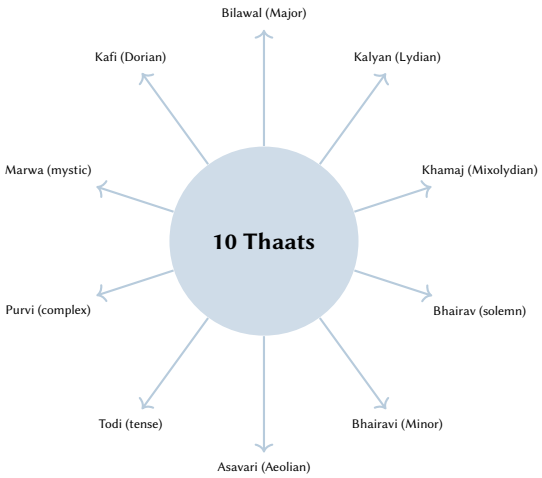


Figure 26. The Hindustani Thaat classification. Each thaat represents a heptatonic interval set that serves as a parent scale for many ragas. Overtone's Thaat selector allows users to pivot between these North Indian modal centers instantly.

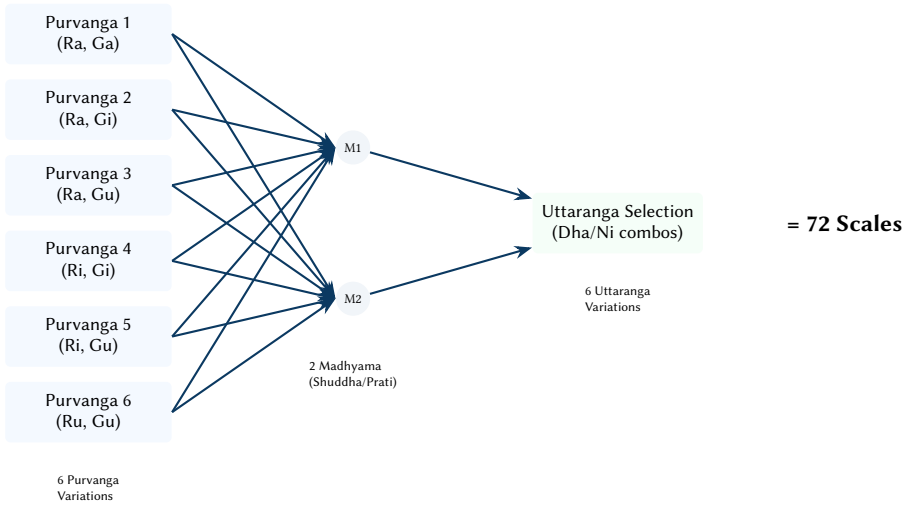


Figure 27. The combinatorial logic of the 72 Melakartas. Six purvanga variations are multiplied by two madhyama positions and six uttaranga variations ($6 \times 2 \times 6 = 72$). Each group of six purvanga/madhyama combinations forms a *Chakra*.

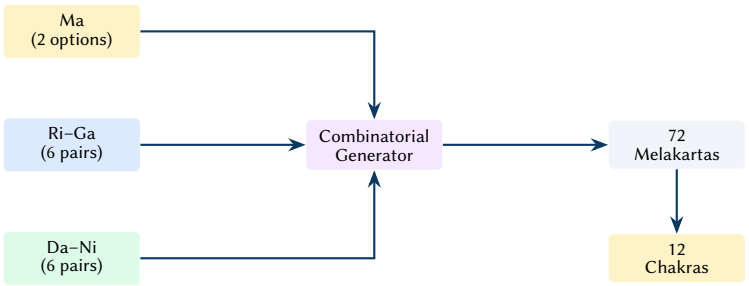


Figure 28. Melakarta generation: three independent pitch choices yield 72 canonical ragas grouped into 12 chakras.

3.13 Thaat Classification (Hindustani)

The `theory/thaat` module defines the 10 canonical thaats of Hindustani music, each with a seven-note interval set, tonal character, time-of-day association, and example ragas. Table 1 summarises the system.

The `thaat_for_scale()` function maps raga names used elsewhere in the engine (e.g., `"bhairav"`, `"todi"`) to their parent thaat, enabling the TUI to display tradition context alongside the active scale.

3.14 Rasa (Aesthetic Emotion)

The `theory/rasa` module implements the *navarasa* framework—nine canonical aesthetic emotions from Indian aesthetics —as complete synthesis parameter profiles. Each rasa maps to a tempo, preferred ragas, energy curve, filter character, reverb character, gamaka style, laya, nadai, drone volume, and shruti purity.

Table 1. The 10 Hindustani thaats with interval sets and time associations.

Thaat	Intervals	Time / Character
Bilawal	[0, 2, 4, 5, 7, 9, 11]	Late morning / bright
Kalyan	[0, 2, 4, 6, 7, 9, 11]	Early evening / serene
Khamaj	[0, 2, 4, 5, 7, 9, 10]	Late evening / romantic
Bhairav	[0, 1, 4, 5, 7, 8, 11]	Dawn / devotional
Purvi	[0, 1, 4, 6, 7, 8, 11]	Evening twilight / serious
Marwa	[0, 1, 4, 6, 7, 9, 11]	Twilight / restless
Kafi	[0, 2, 3, 5, 7, 9, 10]	Late evening / romantic
Asavari	[0, 2, 3, 5, 7, 8, 10]	Late morning / pathos
Bhairavi	[0, 1, 3, 5, 7, 8, 10]	Any time / conclusion
Todi	[0, 1, 3, 6, 7, 8, 11]	Late morning / dark

Table 2. The nine rasas with representative synthesis parameters.

Rasa	Emotion	BPM	Energy	Filter	Laya	Shruti
Shringara	Love	125	wave	Warm	Ekgun	0.80
Karuna	Sorrow	118	yoga_flow	Dark	Vilambit	0.90
Raudra	Fury	150	tension_release	Aggressive	Chaugun	0.70
Veera	Heroism	145	full_set	Warm	Dugun	0.75
Bhayanaka	Terror	135	tension_release	Unstable	Tigun	0.80
Bibhatsa	Disgust	130	flat	Harsh	Ekgun	0.60
Adbhuta	Wonder	132	long_journey	Wide	Ekgun	0.85
Hasya	Joy	140	build_and_drop	Bright	Dugun	0.75
Shanta	Serenity	110	ambient_med.	Soft	Vilambit	1.00

When a rasa is selected, its `RasaProfile` can be applied to `GeneratorConfig` to configure the entire synthesis chain from a single aesthetic intent.

3.15 Murchhana (Modal Rotation)

The `theory/murchhana` module implements scale rotation— the classical technique of deriving new ragas by starting a scale from a different degree.

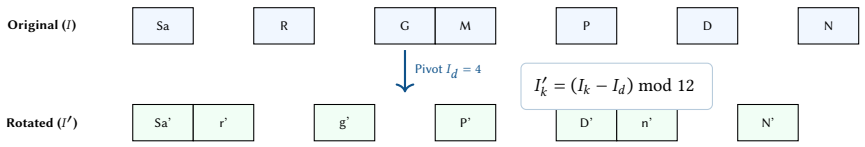


Figure 29. Murchhana modal rotation. The algorithm subtracts the pivot degree's semitone value I_d from all intervals, shifting the modal center and revealing a new raga structure.

The resulting interval set is cross-referenced against all known ragas and the 10 thaats to identify matches. This enables the engine to discover related ragas through transposition, supporting modulation planning and scale reinterpretation.

3.15.1 Live Murchhana Rotation

Murchhana is integrated into the render pipeline via the `murchhana_degree` field on `GeneratorConfig` (optional, 0–6). When set, an `EffectiveKey` abstraction wraps the rotated interval set and is threaded through all pitch generation:

- **Bass patterns** — the function `generate_bass_pattern_with_arp_effective()` generates bass notes normally, then quantises each pitch to the nearest effective pitch class by minimum semitone distance.
- **Melody, alankar, and pakad** — each generator's `generate_bar_with_effective()` method indexes scale degrees against the rotated intervals rather than the original scale.
- **Drone tuning** — the shruti box Sa frequency is shifted to the rotated degree's pitch class, keeping the drone consonant with the new modal center.

The TUI exposes murchhana via Shift+1–7 (select rotation degree) and Shift+0 (disable), with the current degree displayed in the status header. Because murchhana modifies the `EffectiveKey` rather than the underlying scale, the original raga identity is preserved in metadata while the audible pitch set transforms.

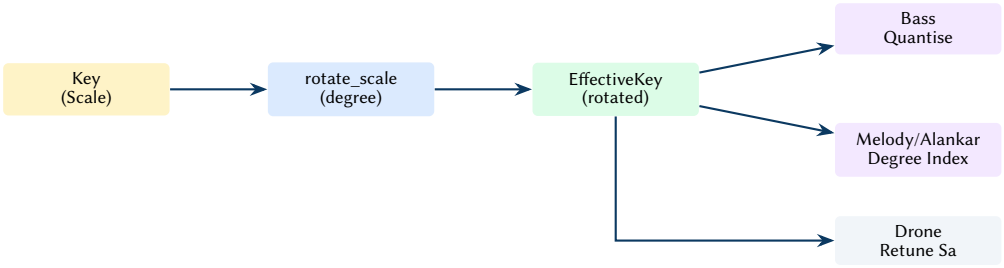


Figure 30. Murchhana integration: the original scale is rotated by degree, producing an EffectiveKey that is threaded through bass quantisation, melody generation, and drone retuning.

3.16 Chakra Energy Mapping

The `theory/chakra` module maps the macro energy level to a seven-chakra frequency model, where each chakra defines a spectral focus region, filter range, synthesis character, and element emphasis.

Table 3. Chakra-to-energy mapping. Each chakra defines the active spectral region as energy rises through the track.

Chakra	Energy Range	Filter (Hz)	Character
Muladhara	0.00–0.08	30–120	Deep, grounding
Svadhisthana	0.08–0.18	80–250	Flowing, warm
Manipura	0.18–0.35	150–500	Powerful, driving
Anahata	0.35–0.55	300–800	Open, melodic
Vishuddha	0.55–0.72	500–1200	Expressive, bright
Ajna	0.72–0.88	800–2000	Clear, piercing
Sahasrara	0.88–1.00	1500–8000	Ethereal, dissolving

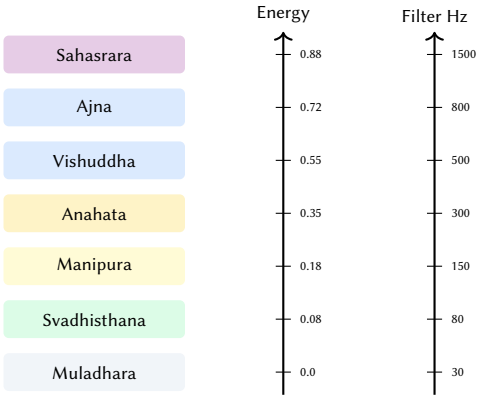


Figure 31. Chakra energy mapping: as macro energy rises, the active spectral focus ascends through the seven chakras.

3.17 Rhythmic Devices

The engine implements six Indian classical rhythmic devices as composable bar-level pattern transformations.

3.17.1 Tihai (Cadential Resolution)

A *Tihai* is a polyrhythmic device used to conclude a section. It consists of a melodic or rhythmic phrase repeated exactly three times so that the final note lands precisely on the *Sam* (the downbeat of the next cycle).

The `TihaiGenerator` produces 16-step resolution patterns by expanding a 4-step phrase three times with 2-step gaps: $[P_4] + [G_2] + [P_4] + [G_2] + [P_4] = 16$ steps. Three phrase variants are available, chosen randomly. The triple repetition resolving on the final step creates a strong sense of arrival at sam (the downbeat).

3.17.2 Korvai (Acceleration Build)

A *Korvai* is a rhythmic composition typically played by percussionists to transition between sections. In Overtone, it is used to create a 8-bar progressive build.

The `KorvaiBuilder` implements an 8-bar progressive acceleration:

3.17.3 Layakari (Time-Stretch)

The `LayakariState` rescales note positions within a bar by a multiplier drawn from five classical speed levels:

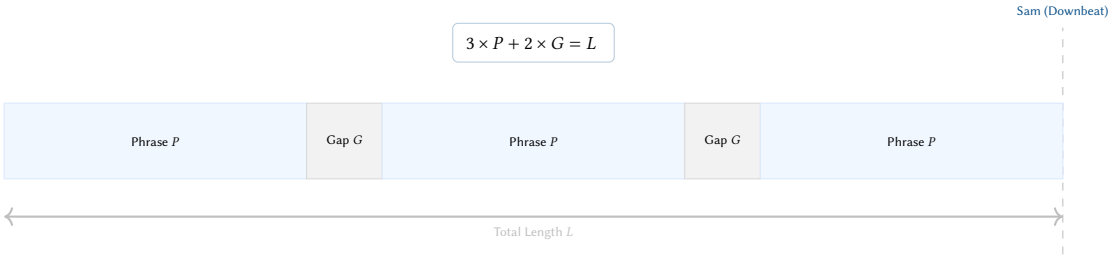
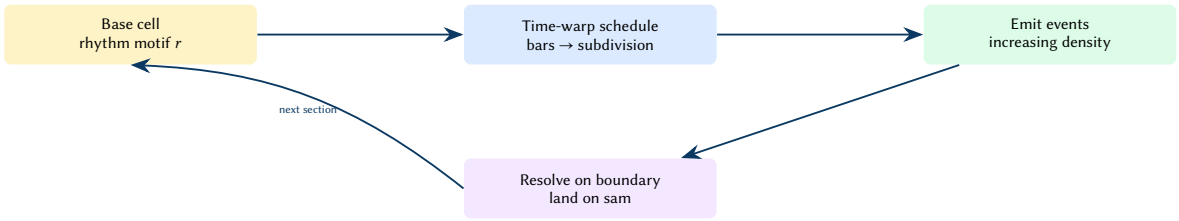


Figure 32. Tihai structural calculation. The generator calculates the phrase length P and gap length G such that the triple repetition resolves precisely on the Sam.



Conceptual point. A korvai is a structured acceleration: it keeps a recognisable motif while changing the internal subdivision schedule over bars. This preserves identity while increasing perceived speed, then resolves cleanly onto the next section downbeat.

Figure 33. Korvai as structured acceleration. The generator holds a motif constant while scheduling subdivision increases across bars, producing a build that still resolves on sam.

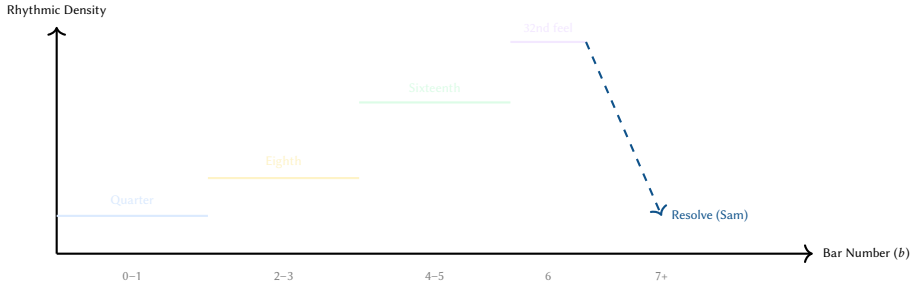


Figure 34. Korvai rhythmic acceleration. The density of note placement increases logarithmically over the 8-bar cycle, creating a "speeding up" sensation that resolves on the first beat of the next section.

Table 4. Korvai acceleration stages over 8 bars.

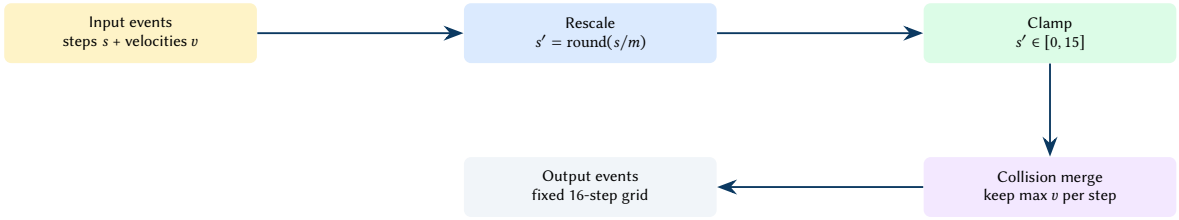
Bars	Stage	Density
0–1	Quarter	4 hits/bar
2–3	Eighth	8 hits/bar
4–5	Sixteenth	16 hits/bar
6	Thirty-second feel	16 hits (accented)
7+	Resolve	1 hit (downbeat only)

- **Vilambit** (0.5×) — half-time, meditative.
- **Ekgun** (1.0×) — normal speed.
- **Dugun** (2.0×) — double-time.
- **Tigun** (3.0×) — triple-time (polyrhythmic).
- **Chaugun** (4.0×) — quadruple-time.

The rescaling algorithm maps each note's step position: $s' = \text{round}(s/m)$, clamps to $[0, 15]$, and resolves collisions by retaining the highest-velocity note. Smooth transitions between speeds are interpolated over a configurable bar window.

3.17.4 Nadai (Subdivision Grouping)

Nadai (or *Gati* in Hindustani) refers to the internal subdivision of a beat. While the underlying tempo remains constant, the number of pulses per beat changes, altering the rhythmic "feel" and syncopation. Overtone implements this as a velocity-multiplier mask applied over the 16-step grid.



Conceptual point. Layakari is a many-to-one mapping. Rescaling can collapse distinct events onto the same step; the collision rule (max velocity wins) preserves accents while preventing double-hits that would break the fixed-grid renderer.

Figure 35. Layakari as a rescale + merge transform. The collision rule is what makes time-warp safe on a fixed 16-step grid.

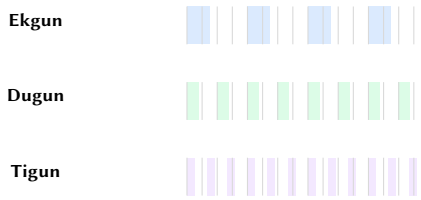
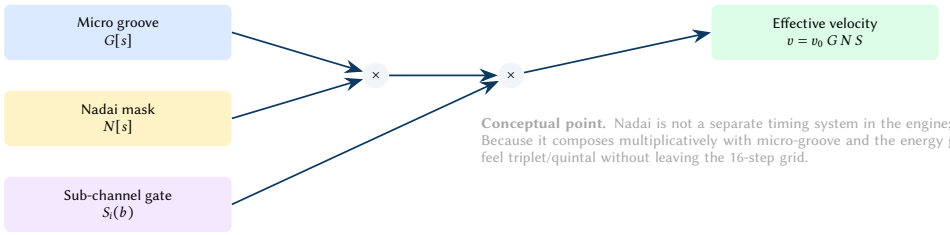


Figure 36. Layakari time-stretching: the same rhythmic phrase at normal (1×), double (2×), and triple (3×) speed within a fixed 16-step bar.



Conceptual point. Nadai is not a separate timing system in the engine; it is a multiplicative accent mask. Because it composes multiplicatively with micro-groove and the energy gate, the same base pattern can feel triplet/quintal without leaving the 16-step grid.

Figure 37. Nadai as mask composition. The final per-step velocity is a product of groove accents, nadai grouping, and the energy gate for the element/layer.

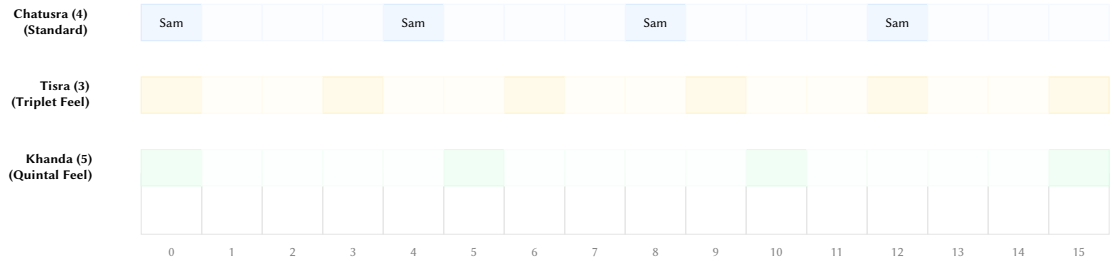


Figure 38. Nadai subdivision masks. By applying these velocity multipliers, Overtone creates the sensation of different pulse groupings (3, 4, 5) while maintaining a strict 4/4 temporal grid. This is used to drive "Eastern Swing" and polyrhythmic lead variations.

The `NadaiState` imposes grouping-based accent patterns over the 16-step grid. Five subdivision sizes are supported: Chatusra (4), Tisra (3), Khanda (5), Misra (7), and Sankirna (9). The accent pattern assigns velocity multipliers:

$$v_k = \begin{cases} 1.0 & k \bmod g = 0 \quad \text{(primary accent)} \\ 0.6 & k \bmod g = 1 \quad \text{(secondary accent)} \\ 0.3 & \text{otherwise} \quad \text{(weak)} \end{cases} \tag{1}$$

where g is the group size. Nadai selection can be driven automatically by macro energy: Chatusra below 0.55, Tisra from 0.55 to 0.72, Khanda from 0.72 to 0.86, Misra from 0.86 to 0.95, and a return to Chatusra above 0.95 for the final climax.

3.17.5 Tani Avartanam (Solo Structure)

The `TaniAvartanam` defines a four-phase solo section (Opening, Development, Climax, Resolution) scheduled at a specific bar range. During a tani, percussion elements are suppressed and melodic density increases progressively, creating a breath in the accompaniment.

3.17.6 Sangati (Progressive Elaboration)

The sangati module adds a subtle energy boost that rises linearly over 8-bar cycles: $\Delta E = (b \bmod 8)/8 \cdot 0.12$, creating a "second time through" feel where each repetition of a phrase is slightly more energetic than the last.

3.18 Prahar: Time-of-Day Mood Selection

In Hindustani classical music, ragas are traditionally associated with specific times of day or night, known as *prahars* (3-hour periods). Overtone implements this via the `samay` mood preset, which selects a raga based on the current system time.

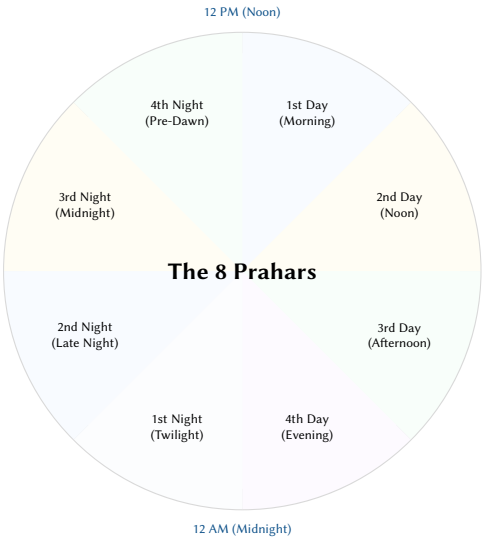


Figure 39. The Prahar cycle. Overtone's `samay` mode queries the system clock to determine the current 3-hour period and selects a raga with the appropriate emotional "juice" (Rasa). For example, pre-dawn tracks default to Raga Bhairav, while late-night tracks favor Malkauns.

3.19 Micro-Expression: Meend and Andolan

The identity of a raga is often found in the movement *between* the notes. Overtone implements two primary micro-expressions:

- **Meend (Glide):** A continuous portamento between two discrete pitches. In the engine, this is implemented as a per-sample frequency ramp over a duration D_m.
- **Andolan (Oscillation):** A slow, gentle vibrato around a specific note, often used on the *vadi* swar.

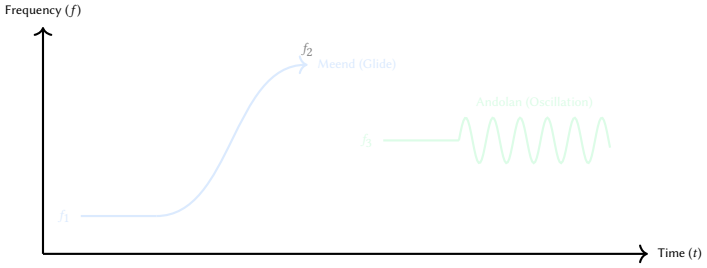


Figure 40. Visualisation of melodic micro-expressions. Meend provides a non-linear frequency transition between notes, while Andolan adds a characteristic slow oscillation to steady tones.

3.20 Sargam and the 12-Tone Grid

Indian music uses seven primary notes (*Saptak*): Sa, Re, Ga, Ma, Pa, Dha, Ni. These are mapped to the 12-tone chromatic grid based on the active scale's intervals.

Figure 41. The mapping of Sargam notes to the 12-semitone grid. Lower-case letters (r, g, d, n) indicate *komal* (flat) variants, while upper-case (R, G, D, N) indicate *shuddha* (natural) or *tivra* (sharp) variants. Sa and Pa are fixed (*achala*).

3.21 Bol Notation and Percussive Synthesis

Indian percussion uses a system of mnemonic syllables called *bols* to encode specific drum strokes. Overtone's percussion engine interprets these bols to dynamically adjust synthesis parameters.

- **Dha / Dhin:** Open, resonant strokes with high sustain and prominent low-frequency content.
- **Tin / Na:** Sharp, metallic strokes with high-frequency emphasis and short decay.
- **Ghe / Ga:** Bass-heavy "resonant" strokes, often used to provide the low-end foundation.

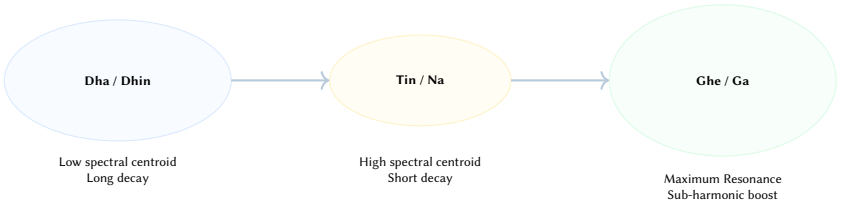


Figure 42. Mapping percussive syllables (bols) to spectral synthesis. The engine modifies the ADSR and filter bank of the percussive generator based on the syllable metadata attached to each event.

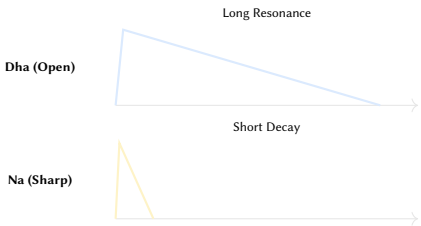


Figure 43. Time-domain envelopes for different Tabla strokes. The percussive engine switches between these envelope shapes based on the bol syllable, ensuring that "Open" strokes ring out while "Sharp" strokes remain percussive.

3.22 Rhythm and Breath: Pranayama in Synthesis

In Indian classical thought, rhythm is intrinsically linked to the breath (*Prana*). A long exhale typically corresponds to a reduction in activity and a focus on sustained tones. Overtone implements this as a slow, asymmetrical LFO called the *Breath LFO*.

Figure 44. The Breath LFO. This slow, 8-bar modulation cycle mimics the human breathing pattern (Puraka/Rechaka). It creates organic motion in the synthesised output, with rising brightness on inhale and receding density on exhale.

The Breath LFO is a global modulation source that biases:

- **Filter Cutoff:** Higher frequency during "inhale" peaks.
- **Density:** Increases note probability during rising phases.
- **Space:** Increases reverb wetness and decay during the long "exhale."

3.23 Jugalbandi (Duet Framework)

Jugalbandi translates to "entwined twins" and refers to a duet between two soloists. In Overtone, this is implemented as an algorithmic handover between two synthesis layers (e.g., Lead and Drone/Pad).

The `JugalbandiMode` enum defines four call-and-response modes: Off, MelodicDuet (lead + pad/drone interaction), RhythmicDuet (kick + bass polyrhythmic conversation), and Full (all elements in duet mode).

The handover logic follows a three-step process:

1. **Snapshot:** The engine records the `BarPattern` of the caller during its active phase.
2. **Variation:** A transformation function is applied to the degrees and timing (e.g., $D' = (D_{max} - D)$ for inversion).

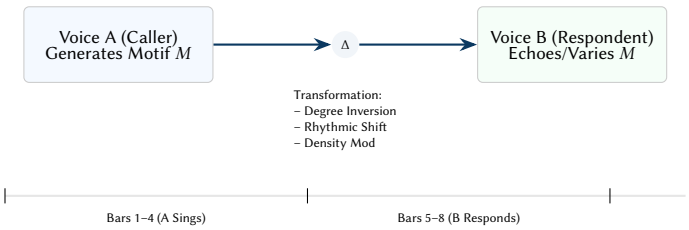


Figure 45. Jugalbandi pattern handover. The engine captures the melodic and rhythmic contour of the "Caller" voice and applies a stochastic transformation before assigning it to the "Respondent" voice. This mimics the live improvisational dialogue of Indian classical music.

3. **Trigger:** The respondent's sub-channel is activated while the caller is gated, creating the "answer."

The active mode is controlled by the performance arc (section 3.27).

3.24 Saptak Hierarchy and Register Mapping

Indian classical music operates across three primary octaves: *Mandra* (Lower), *Madhya* (Middle), and *Taar* (Upper). Overtone maps its musical elements to these registers to ensure a balanced spectral distribution.

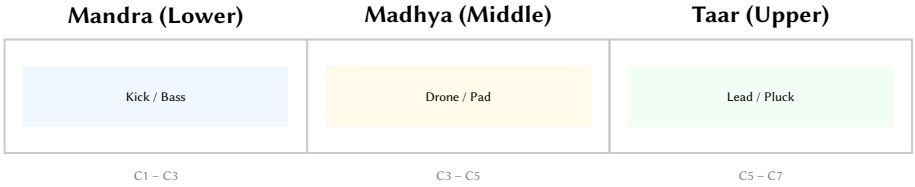


Figure 46. Register mapping across the Saptak hierarchy. By constraining elements to specific octaves, Overtone prevents spectral masking and ensures that the melodic lead (Taar) remains distinct from the foundational drone (Madhya) and rhythmic "engine room" (Mandra).

The `EffectiveKey` module handles octave transpositions automatically based on the element type.

3.25 Tanpura Drone: Harmonic Spectrum and Consonance

The Tanpura is a four-stringed instrument that provides the foundational drone for all raga music. Its unique timbre is characterised by a rich, shimmering harmonic spectrum produced by the *Jivari* (bridge): a slightly curved surface against which the string grazes during vibration, generating a dense series of upper partials (see figs. 67 and 68 in section 8.6 for the synthesis implementation). Four strings are traditionally tuned to Sa–Pa–Sa–Sa (or Sa–Ma–Sa–Sa), establishing the tonic and fifth from which all melodic exploration departs.

3.26 Shruti Box: Additive Synthesis of the Drone

The Shruti Box is a bellows-driven reed instrument used to provide a stable drone. Unlike the shimmering plucked strings of the Tanpura, the Shruti Box produces a solid, organ-like foundation through vibrating metal reeds, with a characteristic bellows-driven amplitude undulation. The synthesis implementation is detailed in section 8.9 (see fig. 70).

In Indian music, *Laya* (tempo) is categorized into three speeds: *Vilambit* (Slow), *Madhya* (Medium), and *Drut* (Fast). Overtone maps these to the macro energy curve to provide a non-linear intensity build (see fig. 58 in section 7).

The `LayaState` also supports *Layakari* (rhythmic multiplication), allowing for mid-bar tempo doubling or tripling.

3.27 Performance Arc (Alap–Jor–Gat)

The `PerformanceArc` orchestrates the entire track trajectory using the classical three-part raga concert structure.

3.27.1 Phase Boundaries

The arc divides the total bar count into three phases using mode-dependent ratios:

Table 5. Performance arc mode boundaries as percentages of total bars.

Mode	Alap (%)	Jor (%)	Gat (%)
Standard	30	30	40
AlapHeavy	40	25	35
GatHeavy	20	20	60

3.27.2 Feature Gating

Each phase activates a `FeatureSet` that controls which synthesis elements, rhythmic devices, and ornamental techniques are available:

- **Alap** — drone and lead only. Breath LFO active, gamaka intensity 1.0, no percussion. Solo melodic exploration.
- **Jor** — kick and bass enter. Jugalbandi set to MelodicDuet, sangati enabled, gamaka intensity 0.7.
- **Gat** — all elements active. Korvai, tihai, tani, and layakari enabled. Jugalbandi escalates from MelodicDuet (first 70%) to Full (final 30%). Jhala activates in the final 15%. Gamaka intensity decreases from 0.55 to 0.35 as rhythmic clarity takes priority.

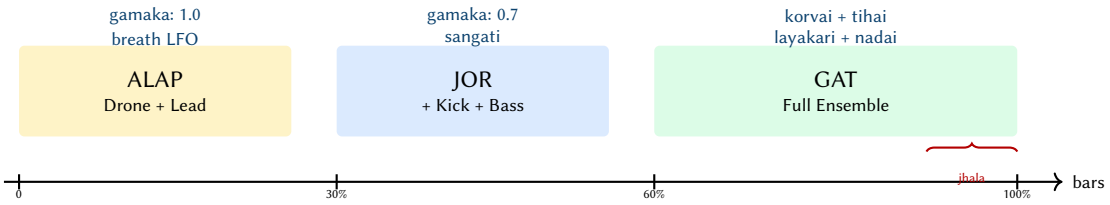


Figure 47. Performance arc feature gating: the classical Alap–Jor–Gat structure progressively activates synthesis elements and rhythmic devices. Gamaka intensity decreases as rhythmic complexity increases. See fig. 25 for the conceptual overview.

3.28 Planned Extensions

The same architecture supports additional eastern concepts as future work: extended murchhana-based modulation planning, a Vilambit– Madhya–Drut tempo layering within the Gat phase, and deeper rasa-to-parameter mapping for emotion-driven composition.

3.29 Implemented Music-Theory Features (Eastern Layer)

The eastern layer is not solely conceptual: it is implemented across the generator configuration, theory primitives, and TUI interaction model. This section documents the concrete features that currently ship, how they are represented in code, and how they influence rendering.

Raga Scale Sets and Phrase Seeds Raga-coloured scale sets are defined in the `theory` layer as interval lists (12-TET semitone offsets) with optional phrase metadata in `src/theory/scales.rs`. In addition to a scale *name* and *intervals*, selected ragas expose a *pakad* structure: a characteristic phrase plus explicit ascending/descending degree sequences. This enables two distinct uses:

- **Constraint** — all pitch selection is constrained to the active scale.
- **Identity injection** — pakad phrases can be injected as melodic seeds, giving a recognisable contour beyond raw interval choice.

Pakad Engine (Phrase-Based Melody Generation) When a selected scale provides a pakad, the engine can instantiate a `PakadEngine` (`src/engine/pakad.rs`) that cycles through a small phrase bank (characteristic, ascending, descending). For each bar it generates a sparse-to-dense pattern as a function of macro energy:

- Density increases with energy (fewer notes at low energy; more notes at high energy).
- Step stride shortens as energy rises (quarter-note feel → 8ths → 16ths).
- A controlled variation mode perturbs degrees by ±1 with probability increasing above mid energy, while remaining in-scale.

This mechanism is intentionally lightweight: it preserves the generator's parameterised design while allowing raga identity to emerge from recurring phrase contours.

Alankar Generator (Permutation Exercises as Ornamented Runs) The generator includes an `AlankarGenerator` (`src/engine/alankar.rs`) that emits bar-length note patterns derived from alankar-style degree permutations. Implemented variants include Sequential, Skip, Mirror, Spiral, Zigzag, and Vakra (a constrained random walk). For instance, the *Sequential* permutation generates degrees by sliding a window as shown in fig. 48.

where C is the current scale cursor, N is the number of intervals in the scale, and M is the number of notes generated per bar based on the laya (speed) and energy level. The generator exposes:

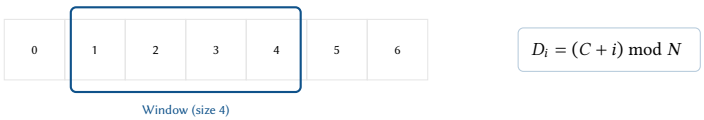


Figure 48. Sequential Alankar generation. The generator slides a window over the scale intervals, where C is the current cursor position.

- **Type** — selects the permutation strategy.
- **Window size** — controls the size of the scale-degree window used by some strategies.
- **Direction** — ascending, descending, or stochastic.
- **Speed (laya)** — maps to steps-per-note (e.g., slower at low energy; faster at high energy).

In the current composition logic the alankar layer is treated as a high-energy embellishment: it can take over lead generation above an energy threshold to produce fast, raga-constrained runs without requiring explicit note programming.

Bol Text Input (Tabla Syllables to Step Grid) The TUI provides a bol text entry mode for quickly imposing a tala/tabla-inspired rhythm onto the tom/percussion lane (`src/tui/app.rs`). A bol string is parsed by the bol parser (`src/engine/bol_parser.rs`) into tokens; each token can carry an articulation label and a velocity. Tokens are then mapped onto a 16-step bar by wrapping or truncation. The resulting per-step overrides are written into the next bar's pattern override state (steps, velocity, and articulation arrays) via the step-edit override structures in `src/engine/generator.rs`.

This feature provides a human-readable rhythmic interface: rather than editing 16 booleans, a user can type a mnemonic sequence and immediately audition it in context.

Shruti Purity (Microtonal Blend) Microtonal behaviour is exposed as a continuous `shruti_purity` parameter in the generator configuration (`src/engine/generator.rs`).

- **0.0** — strict 12-TET tuning.
- **1.0** — apply the full raga-specific shruti offset table (when available for the active scale).

At render time, note frequencies are derived via a tuning function *after* degree selection: the melodic logic remains scale-degree based, while the final oscillator frequencies can slide microtonally. This separation keeps pattern generation deterministic and allows shruti to be treated as a timbral/intonational layer.

Raga Modulation (Scale Transition Over Bars) The engine supports raga modulation as a scheduled transition from a source raga to a target raga over B_{trans} bars beginning at B_{start} (`src/engine/raga_modulation.rs`). The The modulation state evaluates a linear progress function t for the current bar b:

$$t(b) = \text{clamp}\left(\frac{b - B_{\text{start}}}{B_{\text{trans}}},\ 0,\ 1\right)$$

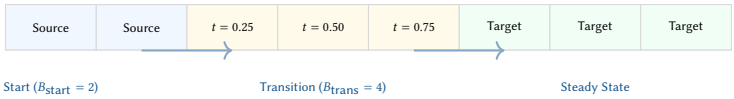


Figure 49. Raga modulation timeline. The engine calculates the progress $t(b)$ linearly over the transition duration, blending interval sets to introduce target-unique tones while phasing out source tones.

This enables mid-track colour changes without abrupt key changes or DAW-side automation.

Time-of-Day (Samay) Mood Selection The `samay` mood preset (`src/theory/mood.rs`) selects a raga and an energy curve based on local time-of-day (hour buckets). This couples two orthogonal parameters:

- **Scale choice** — biases toward ragas traditionally associated with certain times.
- **Energy architecture** — selects among curves such as `yoga_flow`, `long_journey`, and `ambient_meditation` to match the intended affect.

This feature demonstrates the intended design philosophy: music theory is encoded as deterministic selection logic over user-facing parameters, not as opaque learned behaviour.

Breath/Chakra/Arc Controls (Theory-Inspired Macro Behaviour) Several additional switches connect theory-inspired concepts to synthesis and arrangement:

- **Breath-synced LFO** — a slow modulation source that can shape brightness and wetness, particularly effective for low-energy outputs.
- **Chakra mode** — a global mode that maps sections/energy to chakra-derived colour and parameter bias.
- **Performance arc** — an Alap/Jor/Gat-inspired gating mode that controls feature activation over time.

These controls do not replace the energy model; rather, they provide interpretable, named presets that guide how energy maps onto timbre and pattern density.

User-Facing Controls in the TUI The eastern layer is directly manipulable in the terminal UI (`src/tui/app.rs` and `src/tui/ui.rs`). Notable bindings include:

- **Bol entry** — `:` enters bol input mode; on submit the bol string is applied to the next bar's tom lane.
- **Raga modulation** — a shifted key binding cycles through eastern ragas and schedules a transition over a fixed bar window.
- **Alankar** — a shifted key binding cycles the alankar type (and enables it); an alternate-modifier binding toggles alankar on/off.
- **Shruti purity** — incremental adjustments blend between 12-TET and shruti offsets.
- **Murchhana** — Shift+1 through Shift+7 selects a rotation degree; Shift+0 disables. The status header shows the active degree.
- **Samay** — a dedicated binding applies the time-of-day recommendation (mood preset swap).

Because the generator treats the configuration as the single source of truth (section 6), each toggle results in a deterministic re-render that updates both the audio output and the visible state.

Feature-to-Implementation Map Table 6 summarises the primary entry points for each implemented feature: the configuration field that carries it, the module that implements it, and the principal render-time effect.

Configuration Surface (Eastern Parameters) The eastern layer is exposed as explicit fields on `GeneratorConfig` (`src/engine/generator.rs`). This makes the feature set accessible from the CLI, from preset serialization (`serde`), and from the TUI re-render loop.

Table 7. Eastern-layer configuration fields in `GeneratorConfig`.

Field	Type	Meaning / Notes
`mood_name`	`String`	Selects a mood preset (e.g., `"eastern"`, `"meditative"`, `"devotional"`, `"samay"`); mood determines key/scale, energy curve name, and filter ranges.
`key_override`	`Option<(u8,String)>`	Optional override of the mood's root MIDI note and scale name (e.g., `(38, "bhairav")`); applied when resolving the mood.
`energy_curve_override`	`Option<String>`	Overrides the macro curve preset name (e.g., `"yoga_flow"`, `"long_journey"`, `"ambient_meditation"`); used by `MacroCurve::from_name`.
`drone_enabled`	`bool`	Enables tanpura-style drone synthesis layer; typically mixed inversely with macro energy to support sparse/meditative sections.
`breath_lfo_enabled`	`bool`	Enables breath-synced modulation (slow LFO) to shape brightness/wetness; useful for yoga/ambient curves.
`chakra_mode`	`bool`	Enables chakra-derived parameter bias and/or visual mapping in the TUI; can also be implied by yoga-style energy presets.

Continued on next page

Field	Type	Meaning / Notes
shruti_purity	f64	Microtonal blend in [0, 1]: 0.0 = 12-TET; 1.0 = full shruti offsets (when the active scale provides a tuning map).
performance_arc_mode	Option<ArcMode>	Enables Alap/Jor/Gat-inspired gating of features over time; influences when melodic density and layers become available.
modulation	Option<RagaModulation>	Scheduled source→target raga transition: start_bar, transition_bars, and progress function t used at render time for scale blending.
alankar_enabled	bool	Enables alankar-derived melodic generation as a lead-mode enhancement (typically above an energy threshold).
alankar	AlankarGenerator	Alankar configuration (type, window size, direction, speed/laya, cursor state); serialized so repeated renders continue the pattern evolution.
murchhana_degree	Option<u8>	Murchhana rotation degree (0–6); when set, an EffectiveKey wraps the rotated interval set for bass quantisation, melody/alankar/pakad degree indexing, and drone retuning. TUI: Shift+1–7 to set, Shift+0 to disable.
tani_trigger_bar	Option<u32>	Schedules a percussion-solo style event (tani avartanam) beginning at a bar index; used as an arrangement accent.
jugalbandi_mode	JugalbandiMode	Selects call-and-response behaviour between voices; interacts with melody/pattern generation rather than tuning directly.
layakari_enabled	bool	Enables rhythmic multiplication/time-warp within bars (ekgun→dugun→tigun...); affects event timing density.
layakari	LayakariState	Carries the current/target laya and transition scheduling; interpreted per bar at render time.
nadai_enabled	bool	Enables subdivision-group accent warp (tisra/khanda/misra...); applied as a velocity emphasis pattern over 16 steps.
nadai	NadaiState	Holds the current nadai selection and related transition state (if any).

The remaining configuration fields (filter/DSP, sample placement, pattern presets, per-bar overrides) are shared across western and eastern modes; the eastern layer composes with these rather than replacing them.

Table 6. Implemented eastern layer features: config fields, implementation sites, and effects.

Feature	Config / Entry Point	Implementation + Effect
Raga scales + pakad	`Key.scale` (mood / override)	`src/theory/scales.rs`: interval set + optional pakad phrases constrain pitch selection and provide phrase seeds.
Pakad melody	implicit via active scale pakad	`src/engine/pakad.rs`: phrase-bank bar generation with energy-scaled density/stride/variation.
Alankar runs	`alankar_enabled`, `alankar`	`src/engine/alankar.rs`: degree permutations emit fast raga-constrained runs; used as a high-energy lead mode.
Bol text input	TUI override workflow	`src/engine/bol_parser.rs` + `src/tui/app.rs`: parse bols to (step, vel, articulation) and write tom overrides for the next bar.
Shruti tuning	`shruti_purity`	`src/engine/generator.rs` + tuning functions: blend 12-TET to shruti offsets when available; applied at frequency computation.
Raga modulation	`modulation`	`src/engine/raga_modulation.rs`: schedule source→target transition; render-time scale blending introduces target-unique tones over bars.
Murchhana	`murchhana_degree`	`src/theory/murchhana.rs` + `src/engine/effective_key.rs`: live modal rotation; bass quantised, melody/alankar/pakad indexed, drone retuned to rotated degree. TUI: Shift+1–7/0.
Samay mood	`mood_name="samay"`	`src/theory/mood.rs`: time-of-day raga + energy curve selection; presets filter ranges accordingly.
Yoga/ambient curves	`energy_curve_override` / mood preset	`src/engine/energy.rs`: named macro curves shape arrangement density and timbral evolution.
Breath/chakra/arc	`breath_lfo_enabled`, `chakra_mode`, `performance_arc_mode`	`src/dsp/lfo.rs` + `src/theory/chakra.rs` + arc logic: interpretable macro modulation and feature gating layered atop energy.

4 Musical Lineage and Canonical Influences

Overtone's algorithms are not built in a vacuum; they are informed by specific canonical works from both the Psytrance and Indian Classical traditions. These pieces served as benchmarks during the development of the "Raga-Trance" logic.

4.1 Hindustani and Carnatic Reference Works

The engine's melodic and ornamental rules were tuned to replicate the phrasing found in these significant recordings:

- **Ustad Bismillah Khan (Raga Ahir Bhairav):** His Shehnai recordings of Ahir Bhairav influenced the engine's *Meend* (glide) logic, specifically the way the flat second (Re) is approached with extreme expressive weight.
- **Pandit Shivkumar Sharma (Raga Malkauns):** The sparse, meditative patterns of the Santoor in Malkauns served as the basis for Overtone's low-energy ambient/drone generators.

- **L. Subramaniam (Carnatic Violin):** His virtuosic use of *Gamaka* ornaments provided the technical reference for the engine's `GamakaSpec` implementation.

4.2 Psytrance Benchmarks

The rhythmic and timbral character of the engine was cross-referenced against foundational works of the genre:

- **Hallucinogen — *Twisted* (1995):** Informed the engine's acid lead synthesis and the use of extreme filter resonance modulations.
- **Astral Projection — *Trust in Trance* (1996):** Influenced the "Goa" style melodic layering and the use of Phrygian Dominant scales.
- **Ajja — *Psychogenica* (2007):** Served as a benchmark for modern "Forest" style syncopation and organic, glitchy percussion logic.

4.3 The Goa Convergence: A Brief History

The unique "Raga-Trance" hybrid implemented in Overtone traces its spiritual lineage to the beaches of Goa, India, in the late 1980s and early 1990s. This period saw a profound convergence:

- **The Arrival of Electronics:** International travellers brought early synthesizers (Roland TB-303, SH-101) and drum machines to the region.
- **Acoustic Context:** These electronic sounds were played against a backdrop of live Hindustani classical music and the persistent drone of the Tanpura.
- **The Birth of Goa Trance:** This fusion gave birth to Goa Trance, a genre that prioritised hypnotic, raga-like melodic cycles and "squelchy" acid textures over Western harmonic progressions. Overtone formalises this history into code.

4.4 Microtonal Intonation: The 22 Shrutis

While Overtone defaults to 12-Tone Equal Temperament (12-TET), its internal tuning engine is designed to support the **22 Shrutis**—the microtonal divisions used in Indian classical music.

> **Design Decision 2: The Shruti Math**
>
> In Hindustani theory, an octave is divided into 22 unequal intervals. These are often modeled as rational frequency ratios rather than logarithmic divisions. For example, the *Chatushruti Rishabh* (Major Second) might be tuned to 9/8 (~204 cents) instead of the 12-TET 200 cents. Overtone's `shruti_purity` parameter allows the user to blend between these "pure" ratios and standard Western tuning.

Table 8. Comparison of selected Shruti ratios vs. 12-TET cents.

Note (Swar)	Ratio	Just Cents	12-TET Cents
Shuddha Sa (Tonic)	1/1	0.0	0
Komal Re (Flat 2nd)	256/243	90.2	100
Shuddha Re (Nat 2nd)	9/8	203.9	200
Komal Ga (Flat 3rd)	32/27	294.1	300
Shuddha Ga (Nat 3rd)	81/64	407.8	400
Shuddha Ma (Nat 4th)	4/3	498.0	500
Pancham (Fifth)	3/2	702.0	700

Having established the theoretical frameworks—Western electronic, Indian classical, and their cross-cultural fusion—the next part characterises the nine instruments through which Overtone realises these ideas as sound.

Part III — Instruments & Sound Design

5 The Overtone Instrument Palette

Overtone's sonic identity emerges from nine distinct instruments, each occupying a specific spectral and rhythmic niche within the arrangement. This section describes each instrument's musical character, timbral qualities, and role in the mix. The corresponding synthesis implementations are detailed in section 8.

Figure 50 shows the spectral territory occupied by each instrument, and table 9 summarises the energy entry thresholds and mix interactions.

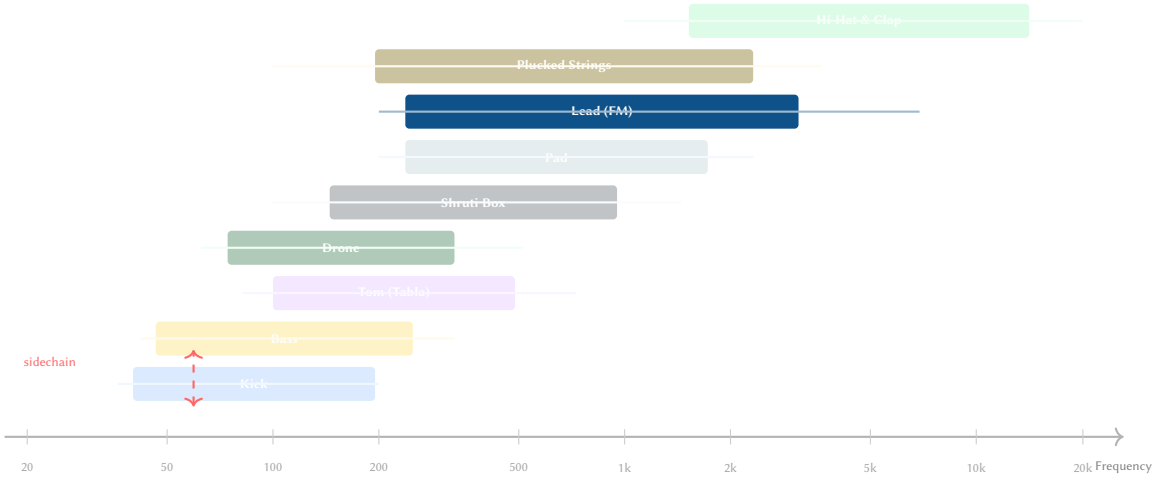


Figure 50. Spectral register map of the Overtone instrument palette. Each bar indicates the primary frequency range occupied by the instrument. The kick and bass share the sub-200 Hz region but are spectrally interleaved via sidechain compression (dashed arrow). Higher-register instruments (lead, hi-hat, plucked strings) distribute across the mid and treble bands.

Table 9. Instrument roles, energy entry thresholds, and mix interactions. The "Entry" column indicates the minimum macro energy level at which the element becomes active; "Interaction" lists key sidechain or gating relationships.

Instrument	Range	Entry	Tradition	Interaction
Kick	30–200 Hz	0.0	Western	Triggers sidechain on bass
Bass	40–400 Hz	0.05	Western	Ducked by kick sidechain
Hi-Hat	6–20 kHz	0.15	Western	Density scales with energy
Clap	1–15 kHz	0.20	Western	Syncopated accent layer
Lead (FM)	200 Hz–8 kHz	0.30	Both	Brightness $\propto$ energy
Drone	60–600 Hz	0.0	Eastern	Volume $\propto 1 - E$
Pad	200 Hz–4 kHz	0.25	Western	Harmonic glue layer
Plucked Str.	100 Hz–6 kHz	0.20	Eastern	Sympathetic resonance
Shruti Box	100 Hz–1.5 kHz	0.0	Eastern	Alternative drone voice
Tom (Tabla)	80–800 Hz	0.15	Eastern	Bol-driven patterns

5.1 Kick Drum

The kick drum is the rhythmic foundation of every psytrance track. It delivers a short, punchy impact centred in the 50–150 Hz range, combining a deep sub-bass thump with a sharp transient click that cuts through dense mixes. In classic psytrance, the kick appears on every quarter note (four-on-the-floor), establishing the relentless 140–150 BPM drive that defines the genre.

The ideal psytrance kick is tight and controlled: the body resonates just long enough to establish pitch (typically around 50 Hz) before decaying completely, leaving spectral space for the bass line. The initial transient—a brief, bright snap—provides the attack definition that makes the kick audible on small speakers and in reverberant club environments. A sub-harmonic layer extends the low-frequency sustain, adding physical weight felt more than heard on large sound systems.

In the energy model, the kick is present from the lowest energy levels, serving as the metronome around which all other elements enter and exit.

5.2 Bass

The bass line is the melodic and harmonic engine of psytrance, filling the spectral region between 40 and 400 Hz with filtered sawtooth timbres that range from deep, rolling sub-bass to bright, squelching acid lines. Unlike genres where bass provides harmonic root support, psytrance bass is an active textural element: its

character evolves continuously through filter modulation, creating the hypnotic, churning motion that defines the genre's sonic signature.

The bass operates in close symbiosis with the kick drum via sidechain compression: each kick hit momentarily ducks the bass amplitude, creating the characteristic "pumping" effect that gives the rhythm its breathing quality. This spectral interleaving ensures both elements occupy the low-frequency range without masking each other.

Two detuned oscillators produce a thick, chorus-like timbre. The "acid" character—synonymous with both Goa trance and modern psytrance—emerges from extreme filter resonance that transforms the bass from a warm foundation into a screaming, self-oscillating lead at high energy levels. Filter envelope presets (acid, slow sweep, pluck) allow rapid timbral variation across bars.

5.3 Hi-Hat and Clap

Hi-hats and claps provide the high-frequency rhythmic texture that sits above the kick-bass foundation. Together they occupy the spectral region from approximately 1 kHz upward, adding the "shimmer" and "snap" that complete the percussion palette.

Hi-Hats. Closed hi-hats produce a short, crisp tick—primarily filtered white noise with a metallic tonal component around 7.5 kHz that gives the sound its "cymbal" character. Open hi-hats have a longer, more shimmering decay. The interplay between closed and open patterns drives the 16th-note groove that distinguishes psytrance from four-on-the-floor house or techno. Pattern density scales with energy: sparse offbeat patterns at low energy expand to full 16th-note rides at peak intensity.

Claps. The clap provides a wider, more diffuse rhythmic accent, typically placed on beats 2 and 4 (or at syncopated positions in Indian-influenced patterns). Its characteristic scattered attack— multiple near-simultaneous transients—simulates the sound of several hands clapping together, creating a broader, more human-sounding hit than a single snare impact.

5.4 Lead

The lead synthesiser provides the primary melodic voice, producing tones that range from soft, bell-like purity at low energy to harsh, complex timbres at peak intensity. Its brightness and duration are directly controlled by the macro energy level: during builds and peaks, the lead becomes brighter and more sustained; during breakdowns, it recedes to sparse, delicate phrases.

In raga-informed modes, the lead becomes the vehicle for Indian classical melodic expression. It follows aroh (ascending) and avroh (descending) paths defined by the active raga, emphasises the vadi ("king note") and samvadi ("queen note"), and carries gamaka ornaments—microtonal embellishments including meend (glides between notes), kampita (oscillating vibrato), andolan (slow pitch undulation), and kan-swar (grace notes). These ornaments transform the lead from a simple pitched voice into an expressive instrument that evokes the character of a sitar or sarangi.

The frequency modulation (FM) architecture enables the lead to produce a wide palette of timbres from a single parameter set, with the modulation index controlling spectral complexity in proportion to the arrangement's energy state.

5.5 Drone (Tanpura)

The tanpura drone provides a continuous harmonic reference—a shimmering bed of sound that anchors the tonal centre of the arrangement. In Indian classical music, the tanpura is indispensable: its four strings (typically tuned Sa–Pa–Sa–Sa or Sa–Ma–Sa–Sa) establish the tonic and fifth (or fourth) from which all melodic exploration departs and to which it returns.

The tanpura's unique sonic character comes from the *jivari* (or *jawari*) bridge: a slightly curved surface against which the vibrating string grazes, producing a rich, non-linear series of overtones that creates the instrument's characteristic shimmer. This dense harmonic spectrum fills the mid-frequency range with a warm, enveloping texture that supports both drone-based meditation and active melodic performance.

In Overtone's arrangement model, the drone volume is *inversely* proportional to macro energy: it fills space during intros and breakdowns (where percussion is sparse) and recedes as the full ensemble enters during peaks. At very high energy levels on eastern scales, the drone activates a *jhala* layer—rapid, rhythmically accented strikes on the upper strings that transform the sustained drone into a driving rhythmic element, echoing the climactic phase of a Hindustani raga performance.

5.6 Pad

The pad synthesiser produces sustained, evolving chord textures that fill the mid-to-high frequency range (roughly 200 Hz to 4 kHz) with a warm, diffuse wash of sound. It voices chords—typically minor voicings

(root, minor third, fifth, minor seventh)—using multiple detuned oscillators per voice, creating a "super-saw" ensemble effect characterised by slow beating and gentle spectral movement.

The pad serves as harmonic glue between the melodic lead and the rhythmic percussion layers. Its long attack and release times (measured in seconds, not milliseconds) create smooth, ambient transitions that soften section boundaries. A stereo chorus effect widens the sound field, while a slow LFO modulates both pitch and filter cutoff to produce the gradual timbral evolution that keeps sustained pads interesting over multi-bar phrases.

In the energy model, the pad activates during mid-energy sections where harmonic richness is desired but rhythmic density has not yet reached its peak.

5.7 Plucked Strings (Sitar, Veena, Sarod, Santoor)

The plucked string engine models four Indian classical instruments, each with a distinctive timbral character:

- **Sitar** — the most recognisable timbre, characterised by long sustain, rich overtones, and a nasal "buzz" produced by the curved *jawari* bridge. Six sympathetic strings resonate in response to the main string, creating a shimmering halo of harmonics.
- **Veena** — a smoother, rounder tone with moderate sustain and four sympathetic strings. Less metallic than the sitar, the veena favours lyrical, sustained passages.
- **Sarod** — bright, clean attack with shorter sustain than the sitar. Five sympathetic strings provide resonance, but the absence of jawari gives the sound a more direct, focused quality.
- **Santoor** — the most percussive of the four, with a short, bell-like attack and rapid decay. Played with hammers rather than plucked, it produces a bright, crystalline tone without sympathetic resonance.

All four instruments share the characteristic of physical string excitation: an initial burst of broadband energy that is gradually shaped by the string's resonant properties, with higher harmonics decaying faster than the fundamental. This natural spectral evolution gives plucked strings their organic, acoustic quality.

5.8 Shruti Box

The shruti box is a bellows-driven reed instrument that provides a stable, organ-like drone. Unlike the shimmering, plucked attack of the tanpura, the shruti box produces a solid, continuous tone with a rich harmonic series shaped by the vibrating metal reeds. The characteristic pump action of the bellows creates a gentle amplitude undulation—a slow breathing quality that distinguishes it from an electronic sustain.

The instrument traditionally accompanies vocal performance, providing a fixed tonal reference in both Hindustani and Carnatic practice. In Overtone, it serves as an alternative drone voice: where the tanpura provides a shimmering, overtone-rich bed, the shruti box offers a warmer, more grounded harmonic foundation. The number of active oscillators varies by mode (Minimal, Standard, Devotional, Full), and the filter cutoff responds to the energy model, brightening the tone as intensity increases.

5.9 Tom Drum (Tabla)

The tom drum engine models tabla articulations, bringing the expressive vocabulary of Indian classical percussion into the electronic arrangement. Three distinct stroke types are available:

- **Dha** — an open, resonant stroke with a deep fundamental and long sustain. In tabla playing, Dha is a compound stroke combining the bass drum (*bayan*) and treble drum (*dayan*), producing a full, rich tone.
- **Tin** — a closed, high-pitched stroke with sharp attack and rapid decay. It provides rhythmic definition and brightness, cutting through the mix with a metallic ring.
- **Ghe** — a deep, sliding bass tone with a characteristic downward pitch sweep. It adds low-frequency weight and movement to tabla patterns.

These articulations can be programmed using traditional *bol* (syllabic) notation: the user types mnemonic sequences (e.g., "Dha Tin Tin Dha Dha Tin Dha Ghe") that are parsed into step patterns and velocity profiles, providing an intuitive rhythmic interface rooted in centuries of oral tradition. The tom responds to both the energy model (density increases with energy) and the tala accent cycle (velocity emphasis follows the rhythmic grouping structure).

The following part turns from musical character to engineering: how each instrument is synthesised, how the energy model drives the arrangement, and how the DSP chain shapes the final mix.

Part IV — Implementation

6 Architecture

Overtone is implemented as a single Rust binary (edition 2021) comprising nine top-level modules. The crate compiles to approximately 12 MB on macOS and has no runtime dependencies beyond the system audio driver (CoreAudio on macOS, ALSA on Linux).

6.1 Module Decomposition

Figure 51 illustrates the module dependency graph. Data flows left to right: the `theory` module provides musical primitives (keys, scales, moods) to the `engine`, which orchestrates pattern generation and calls into `synths` and `samples` for audio rendering. The `dsp` module provides shared signal processing primitives consumed by all synthesis code. The `output` module handles WAV export, MIDI export, and real-time playback. The `ableton` module manages Ableton integration via a TCP/JSON bridge, and the `tui` module provides the interactive terminal interface.

6.2 Key Data Structures

The system is organised around three central data structures:

- `GeneratorConfig` — the single source of truth for all generation parameters (BPM, key, mood, filter settings, energy curve, mute/solo state, pattern presets, humanisation level). Serialisable via `serde` for session persistence.
- `EnergyModel` — the hierarchical energy calculator that maps a bar number to per-element energy levels (section 7).
- `RenderResult` — the immutable output of a render pass, containing interleaved stereo samples, per-element stems, bar pattern metadata for visualisation, and MIDI export data.

> **Design Decision 3: Single Source of Truth**
>
> `GeneratorConfig` flows unmodified from CLI parsing through the TUI state into the `Generator`. Any parameter change in the TUI triggers a full re-render from the config, ensuring that the audio output always exactly reflects the visible parameter state. This design eliminates an entire class of state-synchronisation bugs at the cost of a re-render on every change—a cost that is negligible at 18 ms per 64-bar track (section 18).

6.3 The Two-Stage Pipeline: Composition vs. Generation

The system is architected as a two-stage pipeline where the **Compositional Logic** is strictly decoupled from the **Audio Generation**. This separation ensures that the engine can function as a standalone compositional front-end for DAWs (via MIDI export or the TCP/JSON bridge) even when audio rendering is bypassed.

Figure 52 illustrates this split. The pipeline first resolves the musical structure into an immutable set of `NoteEvent` sequences before branching into one or more output targets.

1. **Stage I: Composition** — The `EnergyModel` is queried to determine the arrangement density and rhythmic character for each bar. The generators then produce `NoteEvent` sequences (pitch, velocity, timing) that exist purely as metadata.
2. **Stage II: Rendering (Optional)** — The note events are dispatched to either the internal synthesis engines (Stage IIa) or pushed to external targets like Ableton Live (Stage IIb).

> **Design Decision 4: Decoupled Output Targets**
>
> Because the `RenderResult` stores the note metadata separately from the audio buffers, the user can iterate on the composition (Stage I) while monitoring the TUI visualisation, and only commit to the computationally expensive audio rendering (Stage IIa) when necessary. Conversely, the engine can act as a pure "MIDI brain" for Ableton Live, bypassing the synthesis layer entirely to save CPU cycles.

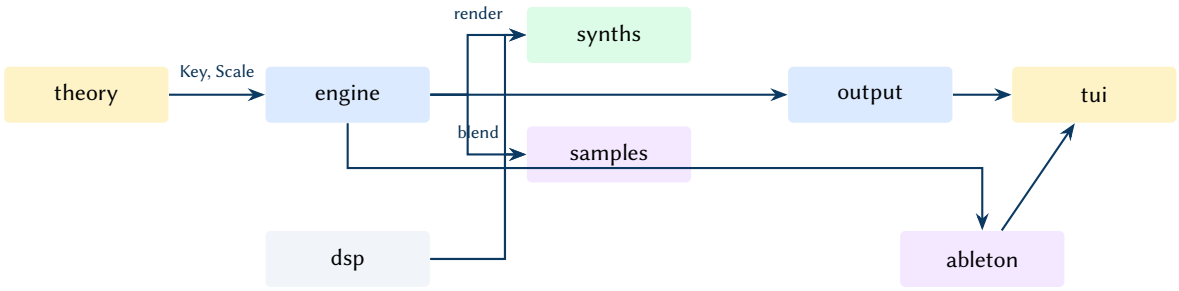


Figure 51. Module dependency graph. Colours indicate functional layers: gold for music theory, blue for core engine and output, green for synthesis, purple for integration and samples, grey for DSP primitives.

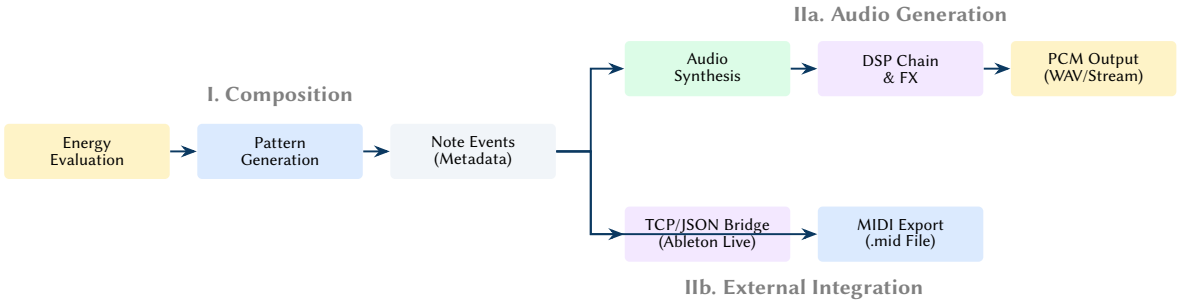


Figure 52. The decoupled pipeline architecture. The Composition stage (I) is deterministic and produces a structured event set. This set can then be rendered to high-precision audio (IIa) or exported directly to external environments (IIb).

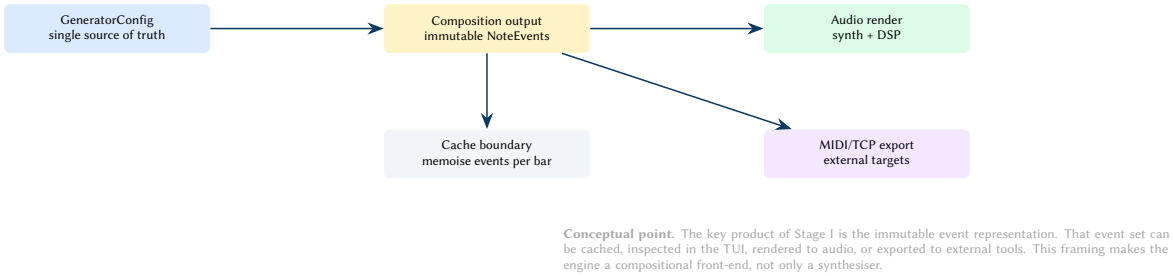


Figure 53. Data-boundary view of the two-stage pipeline. The immutable NoteEvent set is the stable interface between composition and multiple consumers (audio, MIDI, TCP/JSON bridge), and is a natural caching boundary.

7 The Energy Model

The energy model is the central organising abstraction of the generator. Rather than manually arranging patterns on a timeline, the user defines an energy curve and the model determines pattern density, filter modulation depth, sample selection, and element entry/exit timing. The model operates at four hierarchical levels.

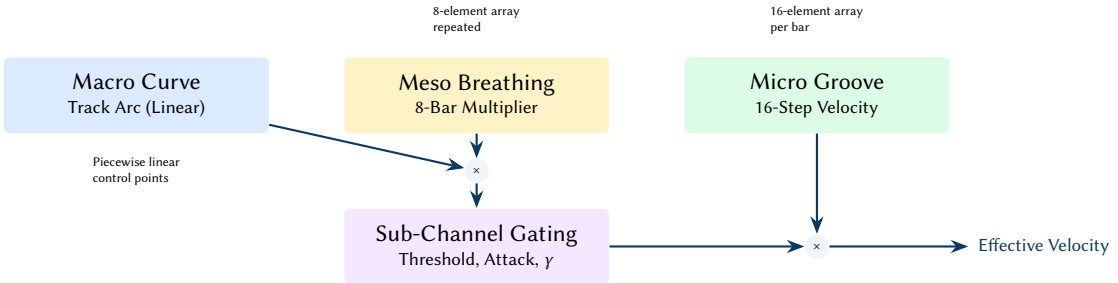


Figure 54. The four-level energy hierarchy. Macro and meso levels combine to drive the sub-channel gate, which is then modulated by the micro groove to produce the final step velocity.

7.1 Macro Curve

The macro curve defines the overall energy arc of the track as a piecewise-linear function $E_{\text{macro}}(b)$ over bar number b. It is specified as a sequence of control points $(b_i, e_i) \in [0, B] \times [0, 1]$ where B is the total number of bars.

Four named presets provide common arrangement shapes:

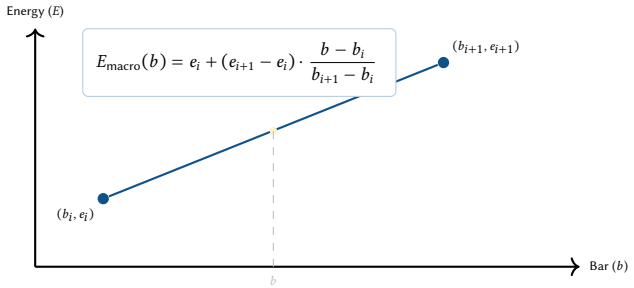


Figure 55. Macro curve linear interpolation. The energy level at bar b is computed by interpolating between the two nearest control points (b_i, e_i) and (b_{i+1}, e_{i+1}).

- **build_and_drop** — rises to 0.9 at 30%, drops to 0.3 at 35%, climbs to 1.0 at 70%, falls to 0.2 at end.
- **full_set** — monotonic rise from 0.2 to 1.0 at 75%, then a gentle decline to 0.7.
- **wave** — oscillating peaks at 0.6, 0.8, 0.9, 1.0 with valleys between, creating a rolling energy feel.
- **tension_release** — two sharp drops (0.7 → 0.2 at 30%, 0.9 → 0.3 at 60%) with recovery climbs, resolving to 1.0 at 90%.

7.2 Meso Breathing

The meso pattern imposes a phrase-level breathing rhythm as an 8-element multiplier array $M = [m_0, \ldots, m_7]$ that repeats every 8 bars. The default *breathing* pattern is:

$$M_{\text{breathing}} = [0.85,\ 0.90,\ 0.95,\ 1.0,\ 1.0,\ 0.95,\ 0.90,\ 0.85] \tag{2}$$

The combined energy at bar b is $E(b) = \text{clamp}(E_{\text{macro}}(b) \cdot M[b \bmod 8],\ 0,\ 1)$.

7.3 Micro Groove

The micro groove shapes velocity at the 16th-note level within each bar. A 16-element array $G = [g_0, \ldots, g_{15}]$ multiplies the velocity of each step, creating rhythmic accents. The *psy_groove* preset emphasises downbeats and offbeat drive:

$$G_{\text{psy}} = [1.0,\ 0.6,\ 0.75,\ 0.5,\ 0.9,\ 0.55,\ 0.8,\ 0.5,\ 0.95,\ 0.6,\ 0.75,\ 0.5,\ 0.85,\ 0.55,\ 0.8,\ 0.65] \tag{3}$$

7.4 Sub-Channel Gating

Each of the five musical elements (kick, bass, hi-hat, clap, lead) has an independent sub-channel that gates and shapes the element's response to the macro energy. A sub-channel is defined by four parameters:

Table 10. Default sub-channel parameters. The response curve exponent controls how aggressively the element responds to energy changes: < 1 gives a gentle ramp, > 1 gives a sharp threshold effect.

Element	Onset Bar	Threshold	Attack (bars)	Response γ
Kick	0	0.05	2	0.7
Bass	4	0.15	4	1.0
Hi-hat	8	0.25	4	1.2
Clap	8	0.35	2	1.5

The effective energy level for element i at bar b is calculated by the gating logic illustrated in fig. 56.

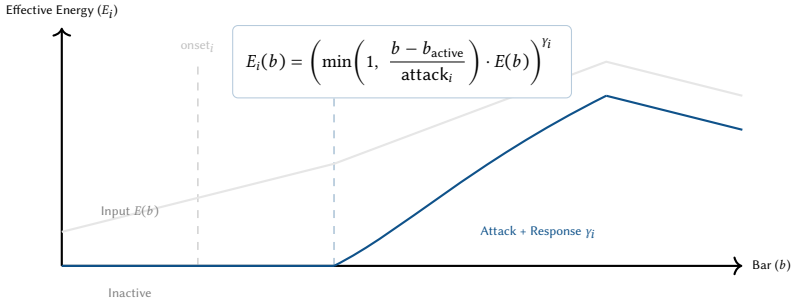


Figure 56. Sub-channel gating logic. The element remains silent ($E_i = 0$) until both the bar onset and global energy threshold are met. Once active, the energy is scaled by a linear attack ramp and a non-linear power-law response γ_i.

The sub-channel parameters (onset_i, thresh_i, attack_i, γ_i) are defined in `engine/energy.rs`.

where b_{active} is the first bar at which both conditions (onset reached and threshold exceeded) are satisfied. The attack ramp fades the element in over attack_i bars after activation, and the response exponent γ_i shapes the energy-to-density transfer function.

Design Decision 5: Staggered Entry

The sub-channel onset and threshold parameters create a natural staggered entry: kick appears first (bar 0), bass enters at bar 4 once energy exceeds 0.15, and percussion layers arrive at bar 8. This mirrors the arrangement conventions of psytrance production, where elements are introduced progressively during the intro and build sections.

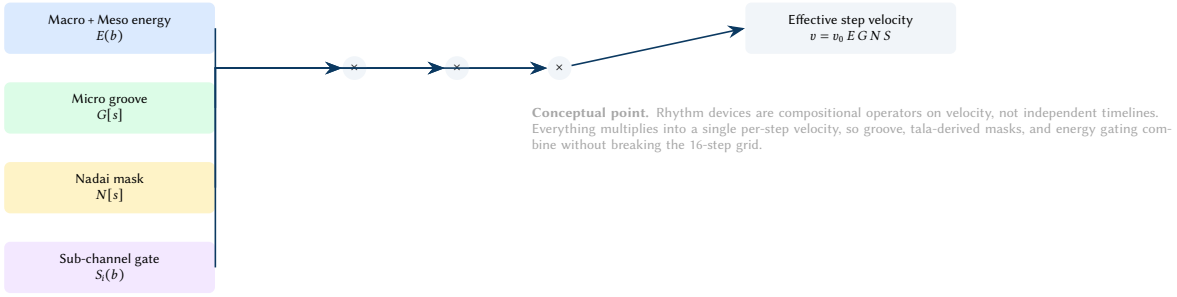


Figure 57. Velocity composition in the engine. Macro/meso energy, micro groove, nadai masks, and per-layer gates combine multiplicatively into the final step velocity used by pattern emitters.

The energy model also determines the tempo regime for eastern scales via the *laya* mapping (fig. 58): low energy maps to Vilambit (slow), mid energy to Madhya (medium), and high energy to Drut (fast).

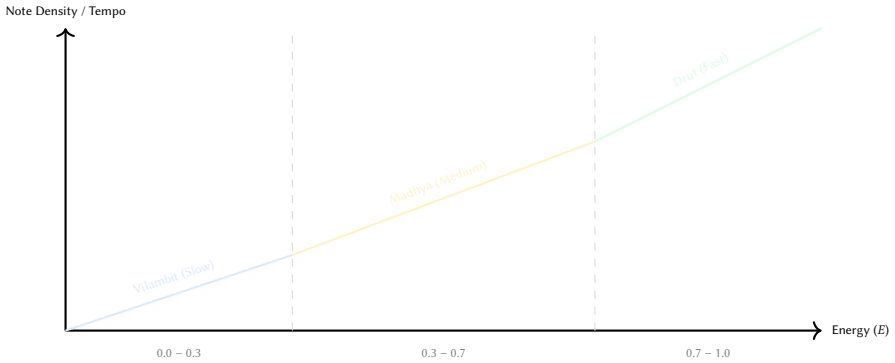


Figure 58. Laya-to-Energy mapping. The generator transitions between three speed regimes based on the macro energy state. As energy crosses the 0.3 and 0.7 thresholds, the rhythmic generators transition from sparse Vilambit phrases to dense Drut passages.

8 Synthesis Engines

The generator includes four dedicated synthesis engines, each implementing the specific timbral characteristics of its target element. All synthesis operates in `f64` precision; conversion to `f32` occurs only at the final output stage.

Musical Context

For the musical character, timbral qualities, and role of each instrument described in this section, see section 5. This section focuses exclusively on the synthesis architecture: oscillator design, envelope parameters, filter configurations, and signal flow.

8.1 Kick Drum

The kick drum synthesiser produces up to 300 ms of audio per hit by summing three layers:

1. **Main body** — a sine oscillator with an exponential pitch envelope sweeping from $f_{\text{start}} \approx 150$ Hz to $f_{\text{end}} \approx 50$ Hz. The instantaneous frequency is:

$$f(t) = f_{\text{end}} + (f_{\text{start}} - f_{\text{end}}) \cdot e^{-\alpha_p t} \tag{4}$$

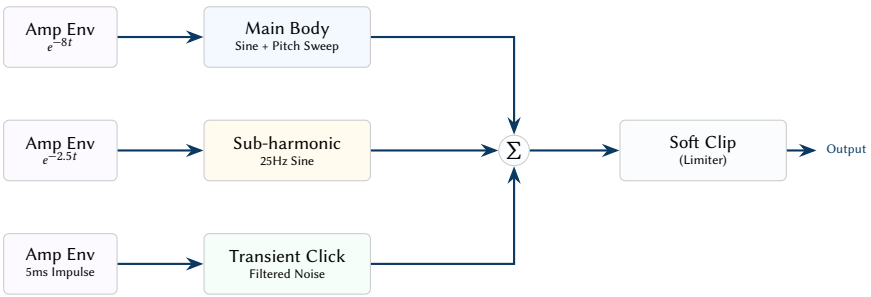


Figure 59. Three-layer kick drum synthesis architecture. The kick is synthesized by summing a pitch-swept sine body, a low-frequency sub-harmonic, and a short-duration noise click for transient definition. Each layer has an independent amplitude envelope scaled by the humanization engine.

where $\alpha_p \approx 35$ controls the pitch decay rate. The amplitude envelope is $a(t) = e^{-\alpha_a t}$ with $\alpha_a \approx 8$.

2. **Sub-harmonic** — a 25 Hz sine with a slower decay ($\alpha_{\text{sub}} \approx 2.5$) at 35% of the main body level. This extends the low-frequency sustain without muddying the transient.
3. **Click transient** — a short (~3 ms) burst of a high-frequency sine at ~3500 Hz with a linear fade-out. The click provides the initial attack snap that helps the kick cut through the mix.

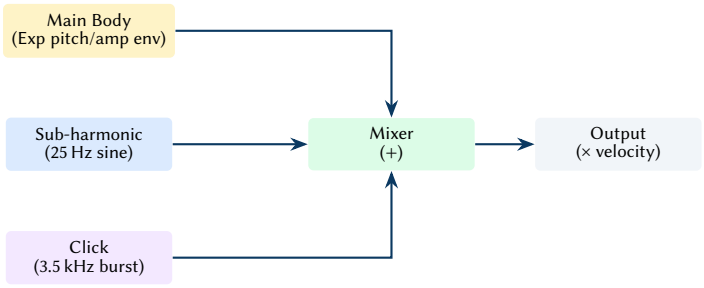


Figure 60. Kick drum synthesis model: parallel summation of main body, sub-harmonic, and click layers.

Phase Accumulation

All oscillators in the generator use phase accumulation ($\phi \mathrel{+}= f/f_s$) rather than direct computation ($\sin(2\pi f t)$). This is essential for the kick's pitch sweep: recomputing $\sin(2\pi f(t) \cdot t)$ with a time-varying $f(t)$ produces phase discontinuities that manifest as audible clicks at the sweep inflection points.

Design Decision 6: Critical Evaluation: Hardcoded Envelopes and Inflexibility

The kick synthesiser heavily relies on hardcoded constants to achieve its characteristic psytrance punch. For instance, the total duration of the kick is locked at 0.30 seconds (~13,230 samples at 44.1 kHz), and the sub-harmonic frequency is fixed to exactly 25 Hz.
While humanisation provides minor random variations, this rigid 3-layer architecture cannot accommodate drastically different kick styles, such as acoustic emulation, 808-style long decays, or hardstyle gabber kicks. The strict 0.30s cutoff abruptly zeroes out the engine, which can lead to low-frequency clicks if the envelope hasn't fully decayed. A more robust implementation would parameterize the decay curves and duration, making it a true general-purpose drum synthesiser rather than a single-trick pony.

8.1.1 Humanisation

The `humanize` parameter (0–1) scales per-hit random variation across all three layers. The humanised parameters and their variation ranges at full humanisation are:

- Start frequency: ±12 Hz
- End frequency: ±3 Hz
- Pitch decay rate: ±4
- Click frequency: ±400 Hz
- Click level: ±8%
- Sub level: ±7%

8.2 Bass Synthesiser

The bass engine renders one note at a time with a subtractive synthesis architecture: two detuned oscillators into a modulated biquad filter, followed by waveshaping and an ADSR amplitude envelope.

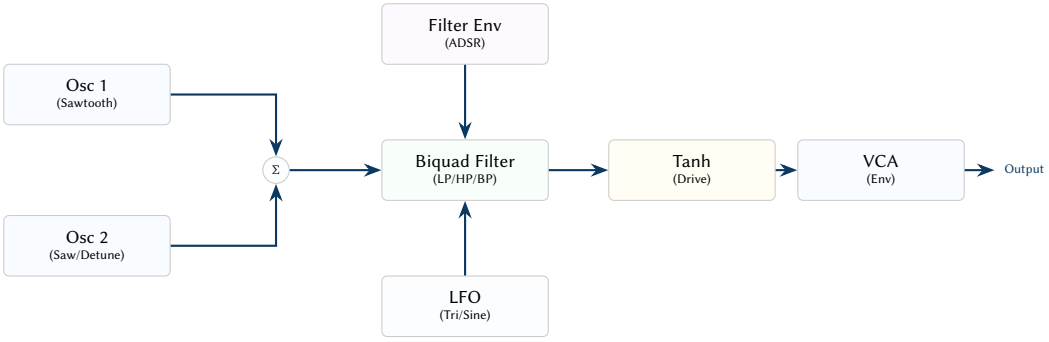


Figure 61. The bass synth signal flow. Dual sawtooth oscillators are summed and passed through a multi-mode biquad filter. Cutoff frequency is modulated by both an ADSR envelope and a free-running LFO. The signal is then saturated via a tanh function and shaped by the VCA envelope.

8.2.1 Oscillator Section

Two oscillators run simultaneously: a main oscillator at the note frequency f_0 and a detuned oscillator at $f_0 \cdot 2^{7/1200}$ (+7 cents). Each oscillator can be either a naive sawtooth (phase accumulation) or a wavetable oscillator scanning through a loaded wavetable. The mix ratio is 70% main, 30% detuned, creating a chorus-like thickening effect.

8.3 Timbral Geometries: Acid and Resonance

The "Acid" sound, characteristic of both Goa Trance and modern Psytrance, is defined by extreme, non-linear filter resonance. Overtone models this using a modified biquad filter with a resonance-dependent gain boost and soft-clipping.

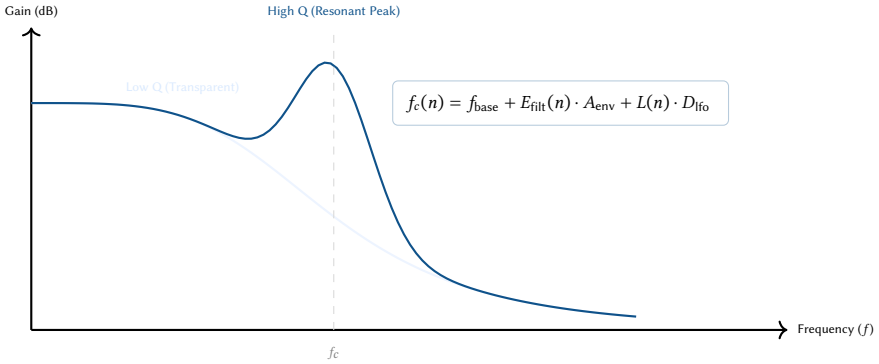


Figure 62. Filter response geometries and modulation sources. The "Acid" character is achieved by a sharp resonant peak at the cutoff frequency (f_c). This frequency is dynamically modulated per-sample by the filter envelope and LFO.

The `BiquadFilter` implements a resonance Q that can exceed 20.0, entering the region of self-oscillation. To prevent digital instability, the signal is passed through a tanh waveshaper in the filter's feedback loop, mimicking the saturation of analogue circuits.

The biquad filter cutoff is modulated per-sample by three sources: the base frequency, the filter envelope, and the LFO (see fig. 62).

where $E_{\text{filt}}(n)$ is the filter envelope output (AD shape with configurable attack, decay, sustain, and amount in Hz), and $L(n)$ is the LFO output (unipolar, 0–1) with depth D_{lfo} in Hz. The filter's `set_params(`f_c`, `Q`)` method recomputes the RBJ biquad coefficients at every sample—a deliberate design choice that trades computational cost for artifact-free modulation.

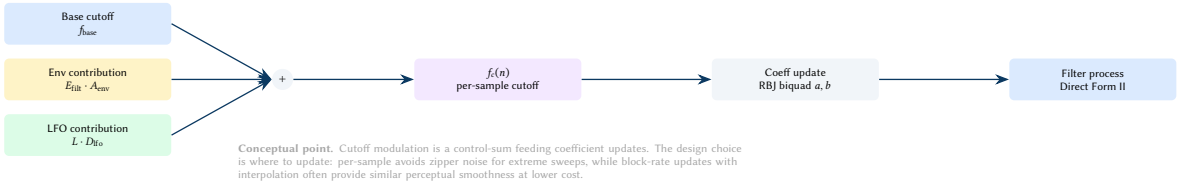


Figure 63. Filter modulation as a control-sum graph. Multiple modulation sources sum into $f_c(n)$, which drives coefficient updates and then sample processing.

Design Decision 7: Critical Evaluation: Per-Sample Coefficient Updates & Hardcoding

While updating biquad coefficients per-sample prevents audible zipper noise during extremely fast filter sweeps (e.g., acid-style 0 ms attack envelopes), it incurs a substantial and often unnecessary computational penalty. At 44.1 kHz, computing trigonometric functions for coefficients on every sample dominates the bass rendering profile. In professional audio engines, block-rate updates (every 32–64 samples) combined with parameter interpolation are standard practice and achieve identical perceptual smoothness at a fraction of the cost.
Furthermore, the amplitude envelope segments are strictly hardcoded (3 ms attack, 40 ms decay, 70% sustain, 30 ms release). While this guarantees a tight, psytrance-appropriate transient, it severely limits the engine's versatility for slower, evolving basslines in ambient or down-tempo contexts. The decision to hardcode these values rather than exposing them in the `GeneratorConfig` reflects a rigid design bias toward high-energy dance music, undermining the engine's stated goal of cross-genre adaptability.

Four filter envelope presets are provided:

Table 11. Filter envelope presets. The *amount* parameter specifies the peak cutoff offset in Hz above the base frequency.

Preset	Attack (ms)	Decay (ms)	Sustain	Amount (Hz)
Off	—	—	—	0
Acid	0	180	5%	2000
Slow Sweep	50	500	20%	1500
Pluck	0	80	0%	3000

8.3.1 Waveshaping and Envelope

The filtered signal is passed through a tanh waveshaper: $y = \tanh(1.3x)$, which adds odd harmonics and provides soft saturation. The amplitude envelope is a simple ADSR with fixed segments: 3 ms linear attack, 40 ms exponential decay to 70% sustain, sustain hold for the note duration, and 30 ms linear release.

8.4 Percussion

8.4.1 Hi-Hat

Hi-hats combine two sources: white noise filtered through a biquad HPF, and a metallic sine tone at approximately 7500 Hz. Closed hi-hats use a 7000 Hz HPF cutoff and fast amplitude decay (e^{-100t}, ~30 ms effective duration). Open hi-hats use a lower 6000 Hz cutoff and slower decay (e^{-15t}, ~150 ms).

8.4.2 Clap

The clap synthesiser produces a multi-tap noise envelope: three consecutive 10 ms bursts at 15 ms spacing (attenuating at 100%, 80%, 60%), followed by a 100 ms filtered noise tail with exponential decay. A 1000 Hz HPF removes low-frequency content. The multi-tap structure simulates the "multiple hands clapping" effect that gives the sound its characteristic scattered attack.

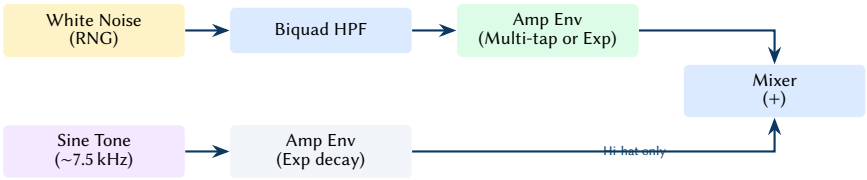


Figure 64. Percussion synthesis model. Hi-hats blend filtered noise with a metallic sine tone, while the clap relies entirely on a multi-tap noise envelope.

Design Decision 8: Critical Evaluation: Overly Simplistic Percussion Models

The percussion synthesiser reveals the most significant compromises in the engine's sound design. The hi-hat relies on a single metallic sine wave blended with filtered white noise, completely missing the complex, inharmonic interplay of multiple square wave oscillators (as seen in classic TR-808/909 architectures). This leads to a thin, somewhat static top-end.
Similarly, the clap synthesis relies on a rigidly hardcoded 3-tap delay structure (15 ms spacing). Real claps have stochastic tap spacing and density that varies per hit. By locking these parameters, the clap loses its organic feel and quickly becomes repetitive, necessitating the heavy use of the sample library (section 14.4) to mask the synthesiser's limitations.

8.5 Lead (FM Synthesis)

The lead synthesiser implements two-operator FM synthesis (see fig. 65).

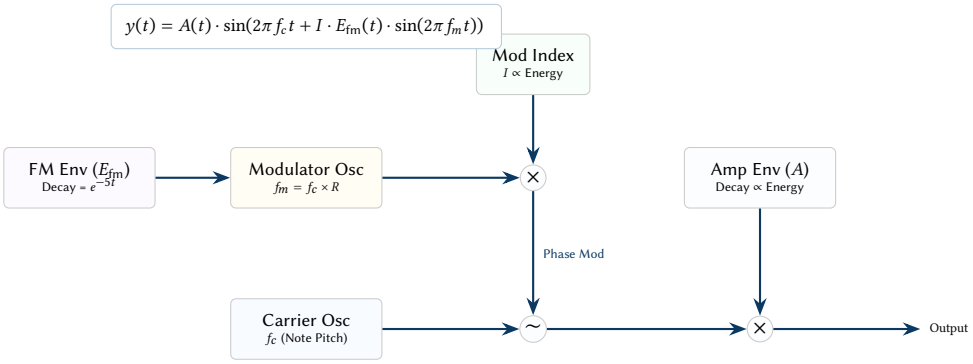


Figure 65. Two-operator FM Synthesis architecture and time-domain equation. The modulator oscillator frequency (f_m) is a ratio (R) of the carrier pitch. The modulation depth (Index I) is dynamically scaled by the macro energy level.

where f_c is the carrier frequency (note pitch), $f_m = f_c \cdot R$ is the modulator frequency (R is the FM ratio), I is the modulation index, and $E_{\text{fm}}(t) = e^{-5t}$ is a fast FM envelope that creates a time-varying timbre. The modulation index I and carrier amplitude decay are scaled by the macro energy level: higher energy produces brighter, longer lead tones.

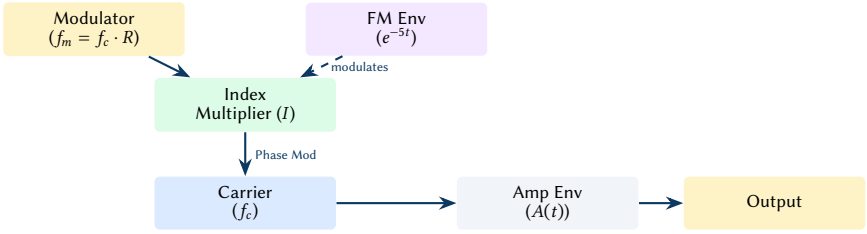


Figure 66. Lead synthesiser signal flow using 2-operator FM.

Design Decision 9: Critical Evaluation: Simplistic FM and Lack of Anti-Aliasing

The lead engine's implementation of FM synthesis is overly simplistic and suffers from significant structural limitations. Relying entirely on a hardcoded e^{-5t} decay envelope for the modulation index prevents the creation of brass-like swells or complex evolving textures that define modern FM patches. More critically, the engine applies phase modulation directly to the carrier using a naive sine accumulator without any form of bandlimiting or oversampling. As the modulation index I scales up with macro energy, the generated sidebands quickly exceed the Nyquist frequency, folding back into the audible spectrum as harsh, inharmonic aliasing. While some degree of aliasing is characteristic of early digital synths (e.g., the Yamaha DX7), the absence of even rudimentary oversampling here degrades the high-frequency fidelity during peak energy sections, leading to a brittle and fatiguing sound.

8.6 Drone (Tanpura)

The drone synthesiser models a four-string tanpura, providing a continuous harmonic reference that fills sparse, low-energy sections of the arrangement.

8.6.1 Architecture

Four independent `DroneString` instances run in parallel, each consisting of two detuned oscillators (fundamental plus a cents-offset copy), three harmonics (fundamental at 100%, 2nd at 30%, 3rd at 15%), and a bandpass filter (220 Hz, $Q = 0.9$). The strings are tuned to Sa (low), Sa (low, slightly detuned), Pa or Ma (depending on the active raga's preferred drone third), and Sa (high octave).

Each string has a *pluck cycle*: a periodic re-excitation at intervals between 0.84 s and 1.18 s (staggered across strings to prevent phase alignment). The per-string envelope is:

$$a(t) = A_{\text{attack}}(t) \cdot e^{-4t/T_{\text{decay}}} \cdot (0.5 + 0.5 \sin(2\pi f_s t)) \tag{5}$$

where A_{attack} is a 2 ms linear ramp, $T_{\text{decay}} = 3$ s, and the sinusoidal swell adds ±4% amplitude variation that simulates the characteristic tanpura "throb".

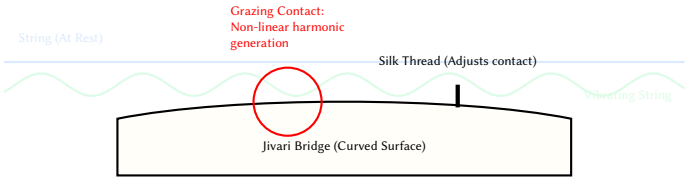


Figure 67. The Jivari bridge mechanism. The bridge has a slightly curved top surface. As the string vibrates, it grazes the bridge at varying points along its length. This periodic "buzzing" creates a rich, non-linear series of overtones. Overtone approximates this shimmer using FM synthesis with a high modulation index.

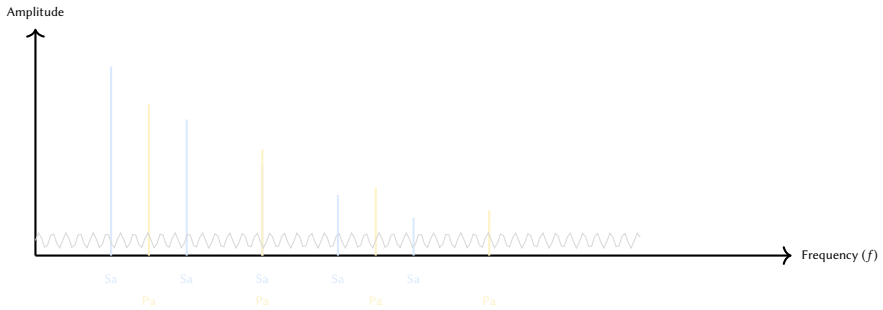


Figure 68. Harmonic spectrum of the Tanpura drone. The fundamental pitches (typically Sa and Pa) are accompanied by a dense series of upper partials that create a continuous, shimmering "bed" of sound. Overtone mimics this via multi-oscillator FM synthesis with slow phase modulations.

8.6.2 Volume and Energy

The drone volume is *inversely* proportional to the macro energy level: $v = (0.08 + 0.12 \cdot (1 - E_{\text{macro}})) \cdot (c_v / 0.15)$, where c_v is the user-configurable drone volume (default 0.15). This ensures the drone fills space during intros and breakdowns but recedes as percussion and bass dominate during peaks.

8.6.3 Jhala Layer

At high energy ($E > 0.85$) on eastern scales, the drone activates a *jhala* layer—rapid, rhythmically accented strikes on the upper strings. The jhala uses a 25% duty-cycle pulse wave with a four-hit accent cycle (strong 1.0, weak 0.55, weak 0.50, medium 0.75). The strike rate escalates with energy: eighth notes below 0.90, sixteenth notes from 0.90 to 0.95, thirty-second notes above 0.95.

8.7 Pad (Ensemble)

The pad synthesiser produces sustained, textured chords using a "super-saw" topology: three PolyBLEP sawtooth oscillators per voice, with up to four voices active simultaneously.

8.7.1 Voice Structure

Each `PadVoice` contains three oscillators at the note frequency, +1.5% detune, and −1.5% detune. Initial phases are spread at 0.0, 0.33, and 0.66 to prevent destructive cancellation on attack. A global LFO at 0.2 Hz modulates all pitches by ±0.5%, creating a slow drift.

The amplitude envelope uses long segments: 2.0 s linear attack, sustain hold, and 3.0 s linear release. The filter is a biquad LPF whose cutoff is modulated by both the LFO and the input amplitude:

$$f_c = 800 + L(t) \cdot 400 + |x_{\text{mix}}| \cdot 200 \quad \text{Hz} \tag{6}$$

8.7.2 Ensemble Chorus

A stereo chorus effect decorrelates the output using two delay taps per channel (15 ms left, 22 ms right) from 50 ms circular buffers. The mix is 70% dry, 30% wet.

When triggered from the streaming engine, the pad voices a minor chord voicing by default: root, minor third (+3 semitones), perfect fifth (+7), and minor seventh (+10).

8.8 Plucked String (Sitar/Veena/Sarod/Santoor)

The plucked string engine uses Karplus–Strong synthesis [1] with preset-specific extensions for four Indian string instruments.

8.8.1 Karplus–Strong Model

A delay line of length $N = f_s / f_0$ samples is excited with a short noise burst (lowpass-filtered), then fed back through a one-pole averaging filter:

$$y[n] = (1-d)\cdot\frac{x[n]+x[n-1]}{2} + d\cdot y[n-1] \tag{7}$$

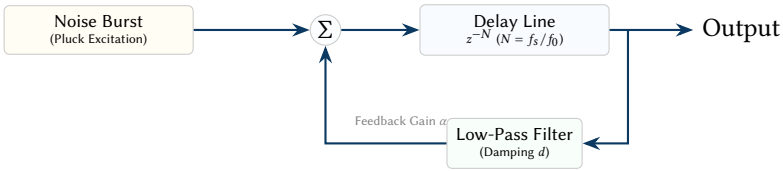


Figure 69. Karplus-Strong physical modeling architecture. A short burst of noise (the "pluck") is fed into a delay line representing the string length. The output is fed back through a low-pass filter, causing higher frequencies to decay faster than fundamental frequencies, realistically simulating a plucked acoustic string.

where d is the damping coefficient and the feedback gain controls the decay time. Table 12 lists the preset parameters.

Table 12. Plucked string preset parameters.

Preset	Feedback	Damping	Jawari	Sympathetic	Resonance
Sitar	0.997	0.12	Yes (0.85)	6 strings	0.06
Veena	0.995	0.25	No	4 strings	0.05
Sarod	0.993	0.10	No	5 strings	0.04
Santoor	0.986	0.08	No	0	—

8.8.2 Jawari (Bridge Buzz)

The sitar preset includes a *jawari* filter that models the curved bridge responsible for the instrument's characteristic nasal buzz. Below a buzz threshold, the signal is reshaped: $y = \text{sign}(x)\cdot|x|^{0.65}$, emphasising upper partials at low amplitudes. The bridge curve parameter (0.85) controls the wet/dry blend.

8.8.3 Sympathetic Strings

Up to six sympathetic strings (Karplus–Strong instances with feedback 0.998 and damping 0.28) resonate in response to the main string output, scaled by a resonance coefficient. The sympathetic halo is mixed at 25% of main amplitude.

8.9 Shruti Box (Electronic Harmonium)

The shruti box synthesiser models an Indian harmonium drone with bellows-driven amplitude modulation.

8.9.1 Oscillator Bank

Four oscillators generate harmonically rich reed tones using six-harmonic additive synthesis: $y = 0.25\sum_{k=1}^{6} h_k \sin(2\pi k f \phi)$, where the default harmonic amplitudes are $h = [1.0, 0.3, 0.7, 0.15, 0.5, 0.08]$. The number of active oscillators depends on the mode: Minimal (1), Standard (2), Devotional (2), Full (4).

8.9.2 Bellows Envelope

An asymmetric triangle LFO at 0.25 Hz simulates the pump action: $a_{\text{bellows}}(t) = 1 - d + d\cdot T(t)$, where $d = 0.15$ is the modulation depth and $T(t)$ is an asymmetric triangle (40% rise, 60% fall) that models fast push and slow pull.

The output passes through per-channel bandpass filters (520 Hz, $Q = 0.9$) with the right channel offset by +3% for stereo width. Filter cutoff modulates with energy: $f_c = 420 + 280\cdot E$ Hz.

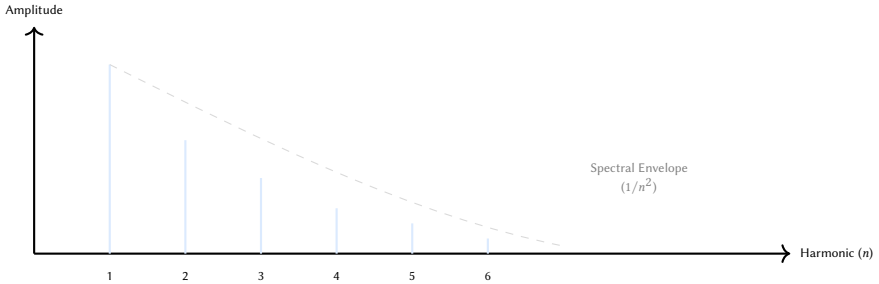


Figure 70. Additive synthesis model for the Shruti Box. The engine sums the first six harmonics of the tonic (Sa) and the fifth (Pa) with amplitudes decaying at a rate of $1/n^2$. A slow bellows-like amplitude LFO is applied to simulate the manual pumping of the instrument.

8.10 Tom Drum

The tom drum synthesiser renders deep, resonant drum hits with three distinct articulations modelled after tabla strokes:

- **Dha** (open resonant) — pitch sweep from $2.5 f_0$ to f_0 (14 ms decay), detuned second oscillator, second harmonic at −12 dB, 2 ms noise burst. 500 ms duration, LPF at $4 f_0$.
- **Tin** (closed sharp) — higher pitch ($f_0 \cdot 2^{7/12} \cdot 1.3$), fast decay (22 ms), metallic ring partial at $2.7 f_0$ (−9.5 dB), BPF at $8 f_0$. 80 ms duration.
- **Ghe** (sliding bass) — pitch sweep from $f_0 \cdot 2^{-5/12}$ to $0.6 f_0$ (10 ms decay), 1 ms noise burst, LPF at 400 Hz. 300 ms duration.

All three use exponential pitch envelopes: $f(t) = f_{\text{end}} + (f_{\text{start}} - f_{\text{end}}) \cdot e^{-\alpha_p t}$. Per-hit randomisation is applied to detune ratio, noise level, and amplitude decay rate.

9 DSP Processing Chain

The `dsp` module provides shared signal processing primitives used by all synthesis engines and the master effects chain. Figure 71 illustrates the signal flow through the effects stage.

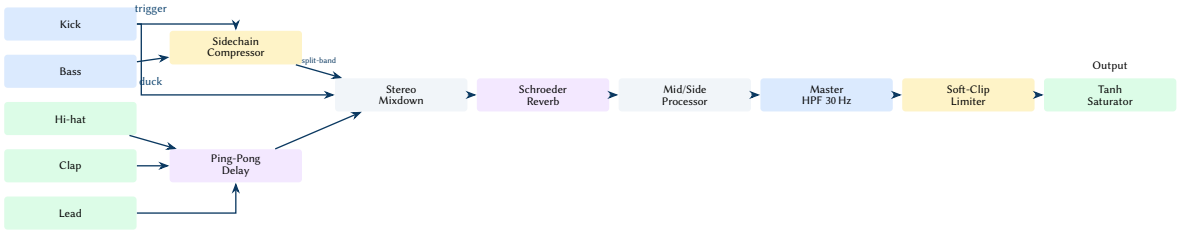
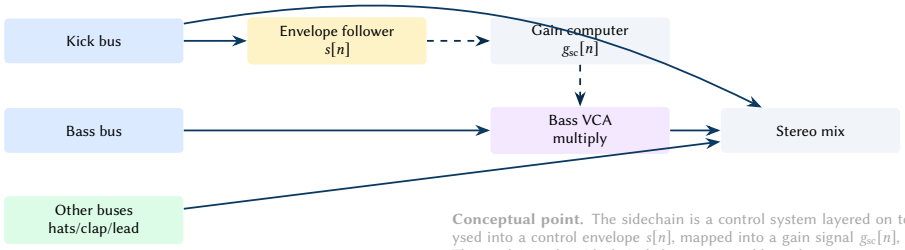


Figure 71. Master effects signal flow. The kick triggers the split-band sidechain compressor which ducks the bass low band. Hi-hat, clap, and lead pass through the ping-pong delay before joining the stereo bus. The master chain applies Schroeder reverb, Mid/Side mono-safe widening, a 30 Hz high-pass, soft-clip limiting, and a tanh saturator for bus glue.



Conceptual point. The sidechain is a control system layered on top of the audio graph: the kick is analysed into a control envelope $s[n]$, mapped into a gain signal $g_{sc}[n]$, and applied multiplicatively to the bass. This explains why sidechain behaviour is stable under mixing: control edges (dashed) are distinct from audio edges (solid).

Figure 72. Sidechain compression as a control graph. Dashed arrows carry control signals; solid arrows carry audio. This framing clarifies how the kick shapes bass dynamics without being mixed into the control path.

9.1 Biquad Filter

The `Biquad` struct implements a second-order IIR filter using the RBJ Audio EQ Cookbook [2] formulas. It supports four filter types: low-pass, high-pass, band-pass, and notch. The filter processes samples using Direct Form II Transposed as shown in fig. 73:

Stability is maintained by clamping the cutoff frequency to $[20,\ 0.45 f_s]$ Hz and the Q factor to $[0.5, 20]$. The upper cutoff clamp at $0.45 f_s$ (rather than $0.5 f_s$) prevents numerical instability as ω_0 approaches π.

9.2 Four-Pole Ladder Filter

The `LadderLp` provides a Moog-style resonant low-pass alternative to the biquad. Where the biquad offers clean, accurate frequency response in multiple filter types, the ladder excels at the aggressive 24 dB/octave slope and resonant "scream" that defines acid and psytrance bass tones. At maximum resonance the filter approaches self-oscillation, producing a pitched sine from the feedback loop alone.

The topology cascades four identical one-pole integrators with a global feedback path. Each stage implements a forward-Euler integrator:

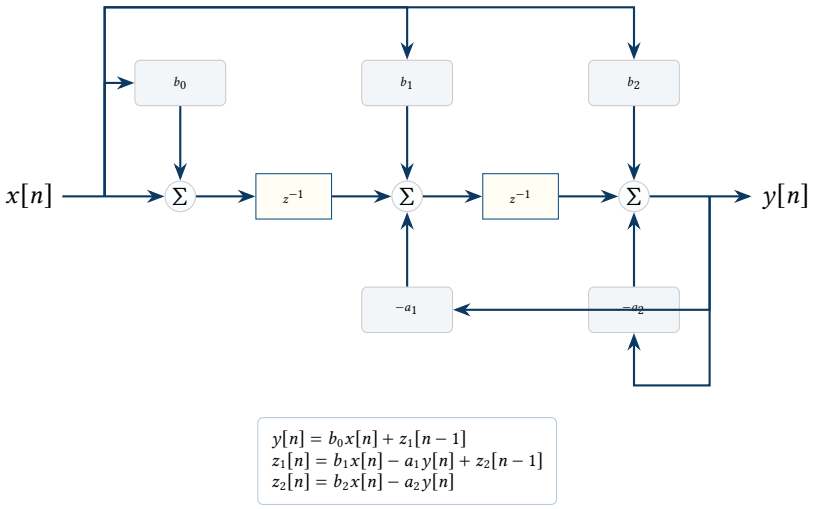


Figure 73. Biquad filter Direct Form II Transposed topology. This structure is preferred for its numerical stability and low memory footprint, requiring only two delay registers (z_1, z_2) per filter instance.

$$g = \min\left(2\pi f_c / f_s,\ 0.99\right) \tag{8}$$

$$u[n] = \tanh\left(x[n] - 4k \cdot z_4[n{-}1]\right) \tag{9}$$

$$z_i[n] = z_i[n{-}1] + g \cdot \left(z_{i-1}[n] - z_i[n{-}1]\right), \quad i = 1 \ldots 4 \tag{10}$$

where $z_0[n] \triangleq u[n]$, f_c is the cutoff frequency, $k \in [0, 1]$ is the normalized resonance, and the factor of 4 maps the unit resonance range onto the self-oscillation threshold. The tanh nonlinearity at the input soft-clips the feedback sum, guaranteeing bounded output even at full resonance.

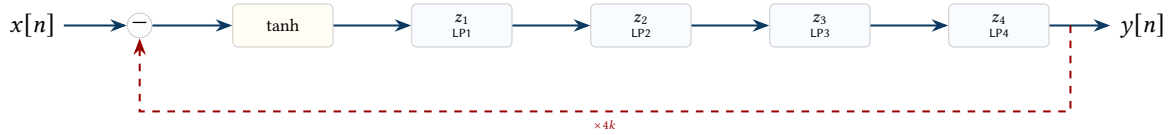


Figure 74. Four-pole ladder filter topology. Four cascaded one-pole integrators provide a 24 dB/oct slope. The feedback path (dashed) routes z_4 back to the input through a resonance gain and a tanh soft-clipper for stability.

Design Decision 10

Forward-Euler vs. ZDF. The ladder uses the simplest possible one-pole integrator (z_i += $g \cdot (\cdot)$) rather than a zero-delay feedback (ZDF/trapezoidal) topology. This keeps the per-sample cost to four multiply-adds plus one tanh, and avoids the iterative solve required by ZDF under fast modulation. The trade-off is slightly less accurate cutoff tracking at high frequencies and a frequency-dependent resonance peak. For the bass and pad voices where the ladder is deployed, this imprecision is inaudible and the characteristic warmth of the forward-Euler topology is musically desirable.

9.3 Sidechain Compression

The sidechain compressor ducks the bass signal using the kick's amplitude envelope as a control signal (see fig. 75). In the latest render path this is implemented with a dedicated `EnvelopeFollower` helper and applied as *split-band ducking*: the low band is attenuated aggressively to protect the kick/sub region, while the bass high band is preserved so upper harmonics and stereo character remain intact.

The control law is:

$$g_{\text{sc}}[n] = 1 - d \cdot \min\left(3 \cdot s[n],\ 1\right) \tag{11}$$

where $d = 0.75$ is the duck depth and $s[n]$ is the kick envelope follower with instantaneous attack and exponential release:

$$s[n] = \begin{cases} |x_{\text{kick}}[n]| & \text{if } |x_{\text{kick}}[n]| > s[n{-}1] \\ s[n{-}1] \cdot \alpha_r & \text{otherwise} \end{cases} \tag{12}$$

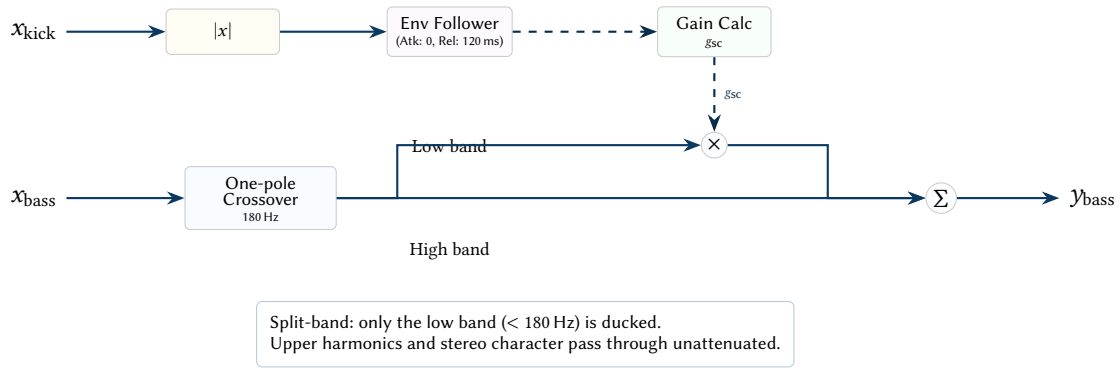


Figure 75. Split-band sidechain compression. The kick drives an envelope follower whose gain signal ducks only the bass low band (below 180 Hz), preserving upper harmonics. Dashed arrows denote control signals.

where $\alpha_r = e^{-1/(0.12 f_s)}$ gives a 120 ms release time constant. This change improves low-end separation without the audible "hole-punch" effect that full-band ducking can create on wide or harmonically rich bass presets.

9.4 Master-Bus Stereo and Saturation Stages

Recent revisions also add two lightweight mastering-oriented processes to the render bus. First, an optional `MidSide` stage converts the stereo signal into Mid/Side form and high-passes only the Side channel. This keeps sub-bass content effectively mono while preserving width in the mids and highs. The Side-path transform is:

$$m[n] = \tfrac{1}{2}(x_L[n] + x_R[n]) \tag{13}$$

$$s[n] = \mathrm{HPF}\big(\tfrac{1}{2}(x_L[n] - x_R[n])\big) \tag{14}$$

followed by the inverse decode $y_L[n] = m[n] + s[n]$, $y_R[n] = m[n] - s[n]$.

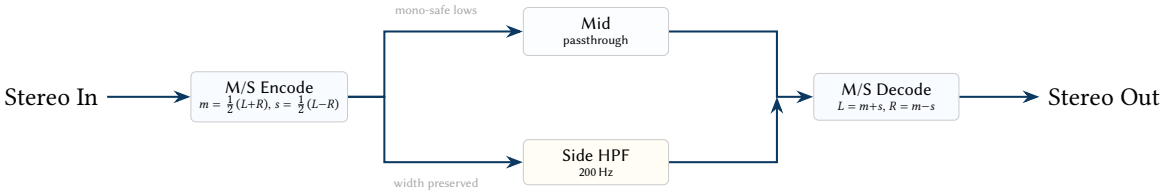


Figure 76. Mid/Side processing flow. The Side channel is high-passed at 200 Hz to collapse low-frequency content to mono while preserving stereo width in the mids and highs.

Second, the master output passes through a normalized tanh saturator used as a gentle glue stage:

$$y[n] = \frac{\tanh(d \cdot x[n])}{\tanh(d)} \tag{15}$$

where d is the drive factor. Because the denominator renormalizes the curve, moderate drive values add density and peak control without a large drop in perceived loudness. Figure 77 compares the transfer function at several drive values.

9.5 Deterministic Macro Modulator

The `MacroMod` provides ultra-slow, deterministic modulation for long-form timbral evolution. Unlike an LFO (which operates at sub-audio rates of 0.1–20 Hz), the macro modulator cycles once over tens of bars, shaping parameters across entire arrangement sections.

The period is specified in bars and converted to Hz via the current tempo:

$$f_{\text{bars}} = \frac{\text{BPM}}{60 \cdot 4} \tag{16}$$

$$f_{\text{mod}} = \frac{f_{\text{bars}}}{P_{\text{bars}}} \tag{17}$$

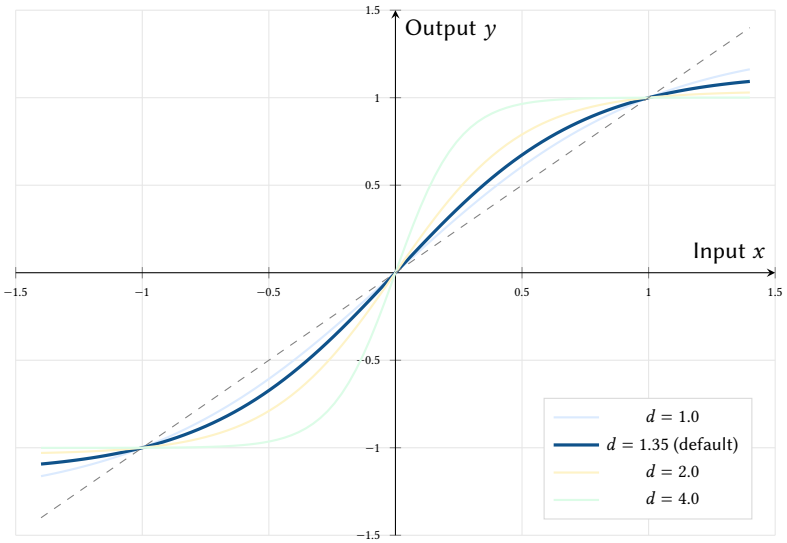


Figure 77. Normalized tanh saturator transfer curves at varying drive values. Higher drive progressively clips peaks while the $1/\tanh(d)$ normalization preserves unity gain at $x = \pm 1$. The default drive of $d = 1.35$ adds gentle density without obvious distortion.

where P_{bars} is the cycle length in bars (assuming 4/4 time). The output is a raised-cosine wave, yielding a smooth unipolar signal with zero corners:

$$v[\phi] = \tfrac{1}{2} - \tfrac{1}{2}\cos(2\pi\phi), \qquad \phi \mathrel{+}= f_{\text{mod}}/f_s \tag{18}$$

In the current render path two macro modulators are active:

- A **64-bar cycle** modulates the Schroeder reverb wet level by ±15% around its base value, adding a slow spatial "breathing" across the arrangement.
- A **48-bar cycle** modulates the bass wavetable position by ±4%, creating gradual timbral drift that is felt rather than heard.

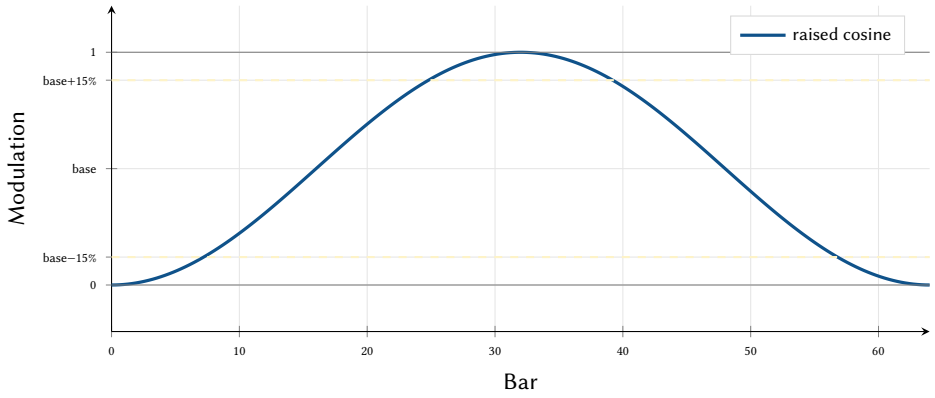


Figure 78. A 64-bar macro modulation cycle (raised cosine). The dashed lines show the reverb wet modulation range (±15% of the base value). The smooth curve avoids audible steps or glitches during long transitions.

> **Note**
>
> **Why deterministic?** The macro modulator uses a free-running phase accumulator seeded at zero rather than random noise or stochastic processes. This ensures that re-rendering the same `GeneratorConfig` with the same seed produces bit-identical audio, which is essential for session recall and A/B comparison during sound design.

9.6 Ping-Pong Delay

The `PingPongDelay` uses two circular buffers (left and right) with cross-feedback routing and a one-pole IIR low-pass filter in each feedback path for analog-style warmth:

$$\text{buf}_L[w] = x_L + \text{fb}_R \cdot k \tag{19}$$
$$\text{buf}_R[w] = x_R + \text{fb}_L \cdot k \tag{20}$$

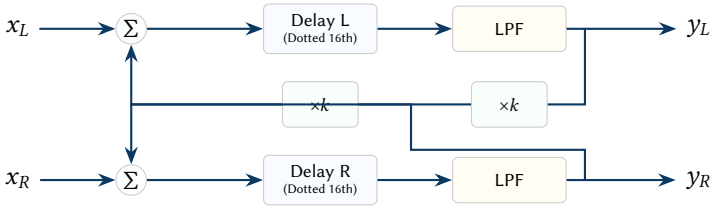


Figure 79. Ping-Pong Delay topology. Left and right channel inputs (x_L, x_R) are fed into independent delay lines. The output of each delay line is low-pass filtered (for analog-style damping), multiplied by the feedback coefficient k, and routed into the input sum of the *opposite* channel's delay line, creating a stereo bouncing effect.

where k is the feedback coefficient (clamped to 0.95 maximum) and w is the write position. The delay time is tempo-synchronised: a dotted 16th note at the current BPM, computed as $t_d = 0.375 \cdot 60/\text{BPM}$ seconds.

Default parameters: 35% feedback, 60% damping coefficient, 20% wet mix.

9.7 Schroeder Reverb

The `StereoReverb` implements the classic Schroeder topology [3] with decorrelated channels:

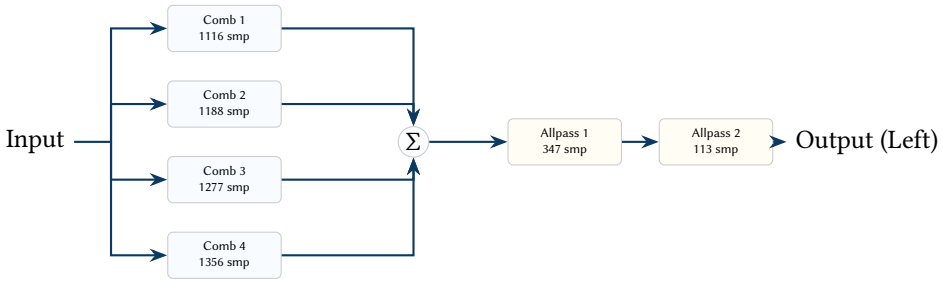


Figure 80. Schroeder Reverb topology (Left Channel). The input signal is fed in parallel to four comb filters with mutually prime delay lengths. Their outputs are summed and passed serially through two allpass filters to increase echo density without coloring the sound.

1. **Four parallel comb filters per channel** (eight total). Delay lengths at 44.1 kHz are chosen to be mutually prime to prevent flutter echo: left channel uses {1116, 1188, 1277, 1356} samples; right channel uses {1139, 1211, 1300, 1379}. A one-pole low-pass filter in each feedback path provides frequency-dependent damping.
2. **Two series allpass filters per channel.** Delays: left {556, 441}, right {579, 464}. The allpass coefficient $g = 0.5$.
3. **Wet/dry mix** at 12% wet.

Default parameters: decay coefficient 0.4, damping 0.7, giving an approximate RT60 of 0.6 s.

9.8 Master Processing

Two final processing stages are applied to the stereo mix:

- **30 Hz high-pass filter** — a second-order Butterworth (biquad HPF, $Q = 0.707$) removes subsonic energy accumulated from the kick's sub-harmonic layer and bass oscillator DC drift.
- **Soft-clip limiter** — normalises to −0.5 dBFS peak, then applies a tanh soft clipper with a knee at 0.9:

$$y = \begin{cases} x & |x| \leq 0.9 \\ \text{sign}(x) \cdot \left(0.9 + 0.1 \cdot \tanh\left(\frac{|x|-0.9}{0.1}\right)\right) & |x| > 0.9 \end{cases} \tag{21}$$

9.9 Wavetable Oscillator

The `Wavetable` structure stores multi-cycle waveforms as a flat `Vec<f32>` with a fixed cycle length (default 2048 samples). The `WavetableOscillator` scans through cycles using two parameters: `phase` (position within a single cycle, 0–1) and `cycle_pos` (morphing position across cycles, 0–1).

9.9.1 Cross-Cycle Interpolation

Sample lookup uses bilinear interpolation across both the cycle dimension and the phase dimension:

$$y(\text{cp}, \phi) = (1 - \alpha) \cdot S(c_0, \phi) + \alpha \cdot S(c_1, \phi) \tag{22}$$

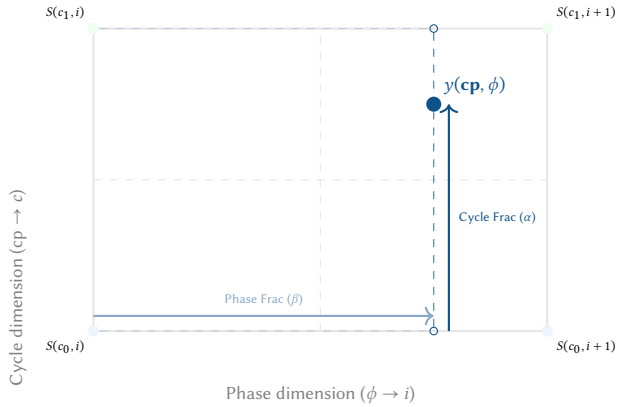


Figure 81. Bilinear cross-cycle interpolation in the Wavetable Oscillator. To prevent zipper noise when sweeping wavetables and aliasing when playing high pitches, the engine first interpolates horizontally between adjacent samples (i and $i+1$) within two adjacent wave cycles (c_0 and c_1), and then interpolates vertically between the resulting values.

where $c_0 = \lfloor \text{cp} \cdot (N_c - 1) \rfloor$, $c_1 = c_0 + 1$, α is the fractional cycle position, and $S(c, \phi)$ performs linear interpolation within a single cycle between adjacent samples. This produces smooth timbral morphing as `cycle_pos` sweeps from the first waveform shape to the last.

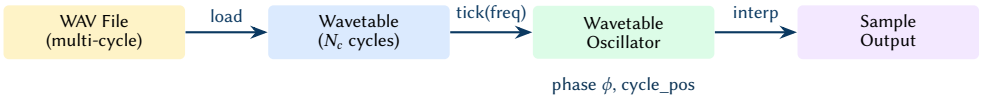


Figure 82. Wavetable oscillator data flow: WAV loading, cycle storage, and bilinear interpolation at playback.

9.9.2 Wavetable Library

The `WavetableLibrary` maintains a `HashMap` of named wavetables. Two synthetic defaults are generated at initialisation: a sawtooth (2048-sample linear ramp −1 to +1) and a square (50% duty pulse). Additional wavetables are loaded from a user-specified directory by scanning for WAV files and splitting each into 2048-sample cycles.

9.10 PolyBLEP Anti-Aliasing

The `PolyblepOscillator` reduces digital aliasing on sawtooth and square waveforms by applying a second-order polynomial correction at phase discontinuities.

9.10.1 Correction Function

The PolyBLEP residual is a piecewise quadratic applied near $t = 0$ and $t = 1$ in the phase domain:

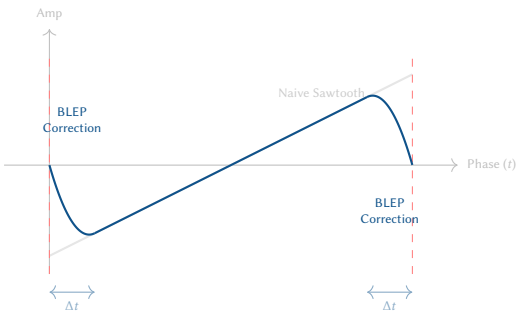


Figure 83. PolyBLEP discontinuity correction. The "naive" sawtooth waveform has an infinite slope at the wrap-around point, which creates aliases across the entire frequency spectrum. The PolyBLEP algorithm subtracts a bandlimited residual (the BLEP) within one sample period (Δt) of the discontinuity, effectively smoothing the edge and suppressing digital artifacts.

$$\text{polyblep}(t, \Delta t) = \begin{cases} \dots \frac{t}{\Delta t}\left(2 - \frac{t}{\Delta t}\right) - 1 & t < \Delta t \\ \left(\frac{t-1}{\Delta t}\right)^2 + 2\frac{t-1}{\Delta t} + 1 & t > 1 - \Delta t \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

where $\Delta t = f/f_s$ is the phase increment per sample.

9.10.2 Bandlimited Waveforms

The corrected sawtooth subtracts the PolyBLEP residual from the naive ramp: $y_{\text{saw}} = (2t-1) - \text{polyblep}(t, \Delta t)$. The square applies the correction twice—at $t = 0$ and at $t = 0.5$—to smooth both transitions:

$$y_{\text{sq}} = \text{sign}(0.5 - t) + \text{polyblep}(t, \Delta t) - \text{polyblep}((t + 0.5) \bmod 1, \Delta t) \tag{24}$$

The PolyBLEP oscillator is currently used in the pad synthesiser (section 8.7) for alias-free detuned sawtooths, and is available for future application to lead and bass oscillators.

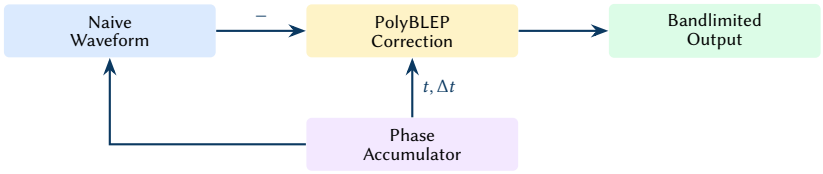


Figure 84. PolyBLEP anti-aliasing: the correction term is subtracted from the naive waveform at phase discontinuities.

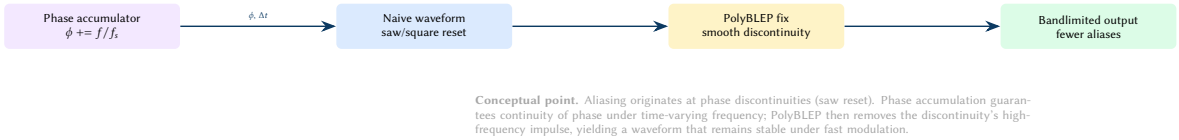


Figure 85. Oscillator correctness view: phase accumulation + bandlimiting. This unifies the time-domain source of aliasing (discontinuities) with the algorithmic fix (PolyBLEP).

10 Sample Intelligence

The sample intelligence subsystem analyses, classifies, and blends audio samples with the synthesised output. It operates in four stages: spectral analysis, rule-based classification, cached profile storage, and energy-aware placement.

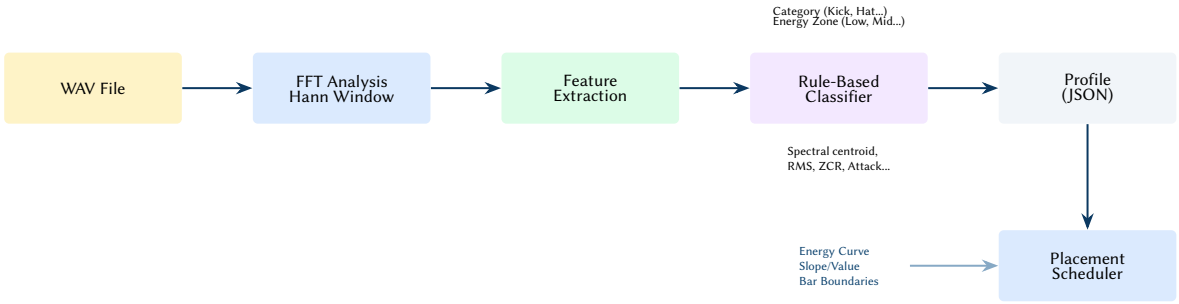


Figure 86. The sample intelligence pipeline. Raw audio is processed via spectral analysis to extract acoustic features, classified into categories and energy zones, and stored as JSON profiles. The scheduler then uses macro energy state to place samples across the arrangement.

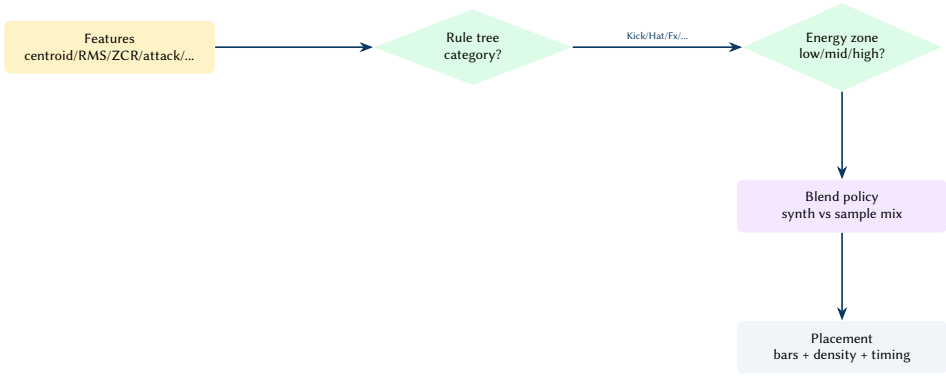
10.1 Feature Extraction

The `analyzer` module extracts seven acoustic features from each loaded WAV file using `rustfft` [4] for spectral analysis:

10.2 Classification

The `classifier` module applies a rule-based decision tree to assign each sample to one of nine categories: `Kick`, `Snare`, `Hat`, `Clap`, `Bass`, `Fx`, `Riser`, `Texture`, `Vocal`.

```
fn classify(profile: &SampleProfile, hint: Option<SampleCategory>)
    -> SampleCategory
{
    let p = profile;
    if p.spectral_centroid < 300.0
        && p.zcr < 0.05
        && p.attack_ms < 50.0
    {
        return SampleCategory::Kick;
    }
    if p.duration_ms > 2000.0 && p.amplitude_slope > 0.0 {
```



Conceptual point. Sample intelligence is a decision system, not just a pipeline: classification produces discrete labels (category, energy zone) that select a blending policy, which then determines how samples are mixed with synthesis and where they appear in the arrangement.

Figure 87. Sample intelligence as a decision DAG. Discrete classifications choose a blending policy which drives placement and mix decisions, making sample use energy-aware rather than purely random.

Table 13. Extracted acoustic features per sample.

Feature	Unit	Method
Spectral centroid	Hz	Energy-weighted mean frequency from Hann-windowed FFT ($\bar{f} = \sum f_i \lvert X_i \rvert / \sum \lvert X_i \rvert$)
RMS level	0–1	Root mean square amplitude
Zero crossing rate	0–1	Sign change count normalised by sample length
Attack time	ms	Time to reach 90% of peak RMS (5 ms analysis frames)
Tonal score	0–1	Normalised autocorrelation peak over lags 1 to $N/2$
Amplitude slope	—	Linear regression slope of per-frame RMS (positive = riser)
Brightness	0–1	Normalised centroid: $\min(1, \text{centroid}/10000)$

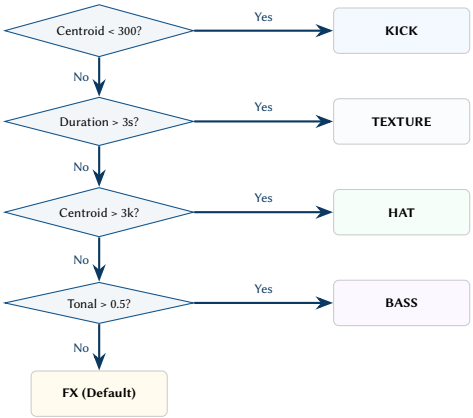


Figure 88. Rule-based sample classification decision tree. Samples are categorized using spectral and temporal heuristics. For example, low spectral centroids (< 300 Hz) combined with short attacks classify as Kicks, while high tonal scores (> 0.5) classify as Bass.

```
        return SampleCategory::Riser;
    }
    if p.spectral_centroid > 3000.0 && p.zcr > 0.15 {
        return SampleCategory::Hat;
    }
    // ... additional rules omitted for brevity
    hint.unwrap_or(SampleCategory::Fx)
}
```

When the rule-based classifier cannot determine a category, it falls back to a directory-name hint (e.g. a sample in a `kicks/` subdirectory is classified as `Kick`).

10.3 Profile Caching

Computed profiles are stored as sidecar JSON files (`<stem>.sample.json`) alongside the source WAV.

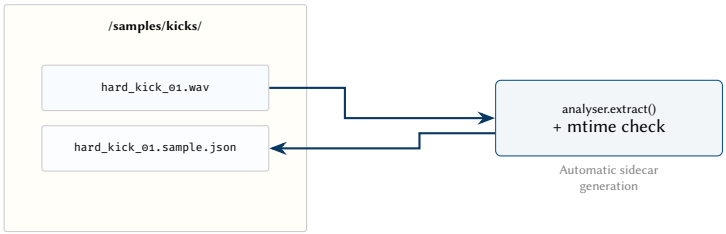


Figure 89. Sidecar caching strategy. For every WAV file in the library, Overtone generates a lightweight JSON metadata file. This prevents expensive FFT re-analysis during application startup, while the modification-time check ensures the cache remains synchronised with the source audio.

Cache validity is checked against the source file's modification timestamp ... and a schema version counter, ensuring that profiles are automatically recomputed when the analyser is updated.

10.4 Energy-Aware Placement

The `PlacementScheduler` pre-computes per-bar placement decisions for non-drum sample categories (FX hits, risers, textures, fills, vocals) based on the macro energy curve's value and slope:

- **Risers** — placed at 4-bar boundaries when the energy slope exceeds 0.15, with length chosen from $\{4, 8, 16\}$ bars to match the estimated climb duration.
- **FX hits** — at 4-bar boundaries when energy > 0.80; additionally at 2-bar boundaries when energy > 0.90.
- **Textures** — on energy plateaus ($|\text{slope}| < 0.05$) above 0.30 energy, at 4/8/16-bar intervals depending on density.
- **Fills** — at 4-bar boundaries above threshold, always at 8 and 16-bar boundaries.
- **Vocals** — every 16 bars, outside the 0.4–0.8 energy range (appearing in intros and peaks but not mid-energy builds).

10.5 Synth/Sample Blending

Each element's output can be pure synthesis, pure sample playback, or a weighted blend. The default blend ratio is computed from the macro energy level:

$$r_{\text{blend}} = \text{clamp}(0.8 \cdot E_{\text{macro}},\ 0,\ 0.8) \tag{25}$$

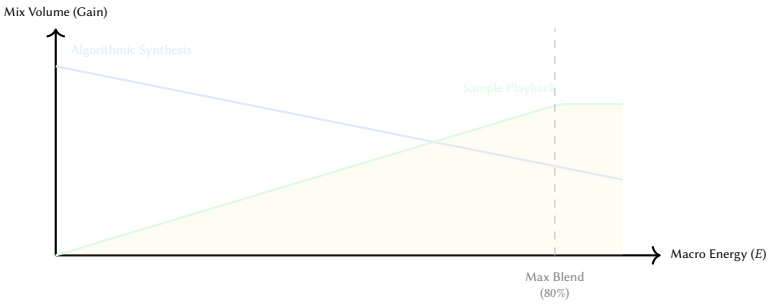


Figure 90. Synth/Sample energy blending. At low energy, the output is almost entirely driven by the internal algorithmic synthesis engines. As macro energy increases towards the peak, parsed audio samples are faded in, providing organic complexity and timbral density to the climax.

At low energy, synthesis dominates (clean, controlled sound); at high energy, samples are blended in (adding the organic variation and complexity of real recordings). Per-element blend overrides are available in the TUI.

11 Pattern Generation

All pattern generators take an `EnergyState` (the combined output of the energy model for the current bar) and produce a `BarPattern` containing a vector of `NoteEvent` structs:

```
pub struct NoteEvent {
    pub step: u8,              // 0..15 (16th note grid)
```

```
    pub velocity: f64,          // 0.0..1.0
    pub pitch_hz: f64,          // fundamental frequency
    pub duration_steps: u8,     // note length in 16ths
    pub timing_offset_samples: i32,  // micro-timing humanisation
}
```

11.1 Energy-Driven Density

Pattern density scales with the element's energy level. For bass, the number of active steps per bar is:

$$N_{\text{steps}} = \text{round}(4 + E_{\text{bass}} \cdot 12) \tag{26}$$

yielding 4 steps at zero energy (quarter notes) and 16 steps at full energy (continuous 16th notes).

11.2 Syncopation and Accent Styles

The rhythmic "feel" of each voice is dictated by a `SyncopationStyle`, which provides a 16-element array of velocity multipliers applied to the grid.

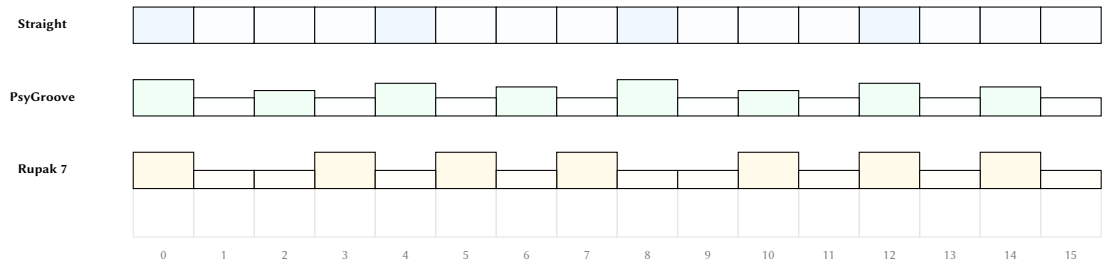


Figure 91. Syncopation style velocity multipliers. Different presets impose distinct rhythmic characters onto the 16-step grid. *PsyGroove* emphasizes offbeat 8th notes, while *Rupak7* imposes an additive 3+2+2 Indian classical feel.

11.3 Presets and Variation

Each element has multiple pattern presets that define distinct rhythmic characters. Table 14 summarises the kick presets as a representative example.

Table 14. Kick pattern presets. Each preset defines placement rules that are further modulated by the energy level.

Preset	Description
Default	Beats 1 and 3 always; beats 2 and 4 if energy > 0.3; ghost 16ths if > 0.7; stochastic pickups and flams
Minimal	Beats 1 and 3 only
Driving	All four beats plus probabilistic 16th-note fills
Breakbeat	Syncopated placement at steps $\{0, 3, 6, 10, 14\}$

11.4 Micro-Timing Humanisation

Three layers of humanisation are applied, all scaled by the global `humanize` parameter (0–1):

1. **Velocity jitter** — ±12%, with softer hits varying more: jitter $= 1 \pm 0.12 \cdot (1 - v \cdot 0.5)$.
2. **Micro-timing** — downbeats are displaced by ±1 ms; offbeats by ±3.5 ms. At 44.1 kHz, 3.5 ms is approximately 154 samples.
3. **Timbral variation** (kick only) — per-hit variation in pitch envelope, click frequency, and sub-harmonic level (section 8.1).

> **MIDI Preservation**
>
> Micro-timing offsets are preserved in MIDI export. At 960 PPQ, one tick is approximately 0.44 ms at 142 BPM, so a ±3.5 ms offset maps to ±8 MIDI ticks—sufficient resolution to reproduce the humanisation feel in a DAW.

11.5 Psytrance Pattern Logic: The Psy_Groove

The quintessential psytrance rhythm is a 16th-note subdivision where the kick drum occupies the first 16th note of each quarter-note beat (the "four-to-the-floor" foundation). The bassline interacts with this grid to create the driving momentum characteristic of the genre. Overtone implements several variations of this relationship:

- **Offbeat (1/1):** The most foundational pattern, where a single bass hit occurs on the 16th note immediately following each kick.
- **Rolling (3/1):** Three consecutive bass hits follow each kick, filling the remaining 16th-note slots in the beat. This creates a relentless, "rolling" forward motion.
- **Galloping (1/2):** A long kick hit followed by two rapid bass hits, creating a "gallop" feel (e.g., [Kick, *gap*, Bass, Bass]).
- **Syncopated (16th):** Bass hits are placed on syncopated 16th-note positions (e.g., steps 3, 6, 9, 12), creating cross-rhythmic tension against the steady kick pulse.

These patterns are implemented in `engine/patterns.rs` using a step-weighting system. At low energy, the engine prefers simple offbeat patterns; as energy increases, it transitions into rolling or syncopated variations to heighten the driving feel.

11.6 Euclidean Rhythm and Phase Preservation

The `PolyrhythmEngine` generates Euclidean rhythms using Bjorklund's algorithm [5]: given k pulses distributed as evenly as possible over n steps, the algorithm produces maximally even patterns such as the 3-over-8 tresillo or 5-over-12 West African bell pattern.

11.6.1 The Phase Problem

A naïve implementation resets the pattern index to zero at every bar boundary: step i in bar b maps to $i \bmod n$. When n does not divide 16 (the 16th-note grid length), the pattern wraps at the same offset every bar, producing identical bars and destroying the characteristic cross-bar polyrhythmic drift that makes Euclidean patterns compelling.

11.6.2 Persistent Absolute Counters

The fix is a per-element `poly_steps[i]` counter that persists across bars and advances by 16 each bar. The step index into the Euclidean pattern becomes:

$$\text{idx}(b, i) = (\texttt{poly\_steps} + i) \bmod n \tag{27}$$

so bar 1 starts at offset 16 mod n rather than zero, and the pattern evolves across bars until it cycles after $\text{lcm}(16, n)/16$ bars.

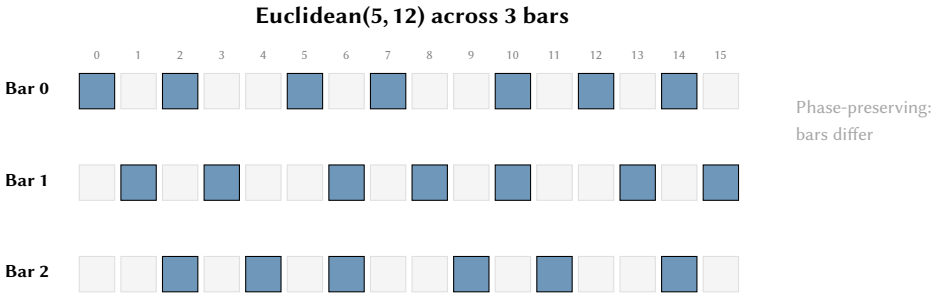


Figure 92. Euclidean phase preservation. A $(5, 12)$ pattern across three bars with persistent absolute step counters. Because $16 \bmod 12 = 4$, each bar starts at a different phase offset, producing evolving polyrhythmic accents instead of identical repetitions.

11.7 Raga-Aware Melody Generation

The `RagaMelodyGenerator` produces bar-length melodic patterns for lead synthesis when an eastern scale is active. It respects the raga's aroh (ascending path) and avroh (descending path) and weights pitch selection toward the vadi (primary important tone) and samvadi (secondary important tone).

11.7.1 Density and Direction

Note density scales with macro energy: 3 notes per bar below 45% energy, 4 notes between 45% and 75%, and 8 notes above 75%. The generator maintains a traversal direction (ascending or descending) that flips with 25% probability each bar, producing melodic contour variety. Pitch candidates are drawn from the direction-appropriate path when the scale defines explicit aroh/avroh sequences.

11.7.2 Pitch Selection

At low energy (< 0.45), pitches are drawn uniformly from the scale's stable tones (root, preferred drone third, perfect fifth). At higher energy, a weighted random selection favours the vadi (40% probability) and samvadi (25%), with the remaining probability spread across all allowed degrees. Octave selection is one octave above the root by default, with a 15% chance of jumping to two octaves above 75% energy.

11.7.3 Ornamentation

Each generated note may receive a `GamakaSpec` ornament based on the raga's per-pitch gamaka rules:

- **Andolan** — slow oscillation (0.9 Hz, 20–60 cents depth).
- **Kampita** — sharp vibrato (4.0 Hz, 8–25 cents depth).
- **Meend** — pitch glide from the previous note (90 ms slide), triggered when the interval is ≤ 4 semitones and energy exceeds 45%.
- **Kan-swar** — grace note approach (± 1 semitone, 20 ms), triggered at 15% probability above 65% energy.

Note duration decreases with energy: 3 steps (held) below 45%, 2 steps at moderate energy, 1 step (rapid-fire) above 75%. Velocity is accented on the first note of each bar (0.95) and jittered by $\pm 8\%$ on subsequent notes.

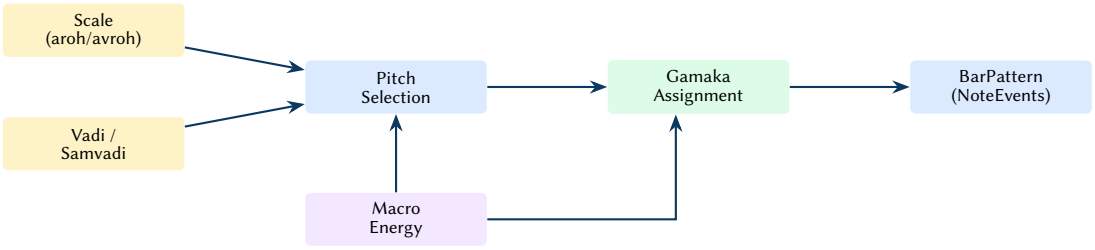


Figure 93. Raga melody generation: scale constraints and vadi/samvadi weighting feed pitch selection, energy controls density and ornamentation depth, and gamaka rules produce per-note ornaments.

11.8 Arrangement Sections

The `arrangement` module classifies each bar into one of four structural sections—Intro, Build, Peak, or Breakdown—using a heuristic decision tree over the macro energy curve's value, slope, and song progress:

Table 15. Arrangement section detection heuristics. Progress is measured as bar/total_bars.

Section	Conditions
Intro	Progress $< 12\%$ and energy < 0.45; or energy < 0.35 at any point
Build	Slope > 0.02 (rising energy); or default for mid-range energy
Peak	Energy ≥ 0.85
Breakdown	Progress $> 30\%$, slope < -0.04, energy < 0.60; or energy < 0.35 after 30% progress

These section labels are used by MIDI export, Ableton scene mapping, and TUI status display.

12 Harmonic Planning and Tension Architecture

While the energy model (section 7) controls *how much*—density, volume, activity—music also requires a parallel axis for *how tense*. A quiet passage can be harmonically unresolved (preparing a climactic resolution), and a loud passage can be harmonically simple (the euphoric payoff). The tension architecture introduces a multi-dimensional compositional layer that operates independently of energy, controlling dissonance, chord complexity, key modulation, rhythmic syncopation, register placement, and resolution planning.

12.1 The Tension Curve

The tension curve is a composite signal $T(b) \in [0, 1]$ over bar number b, decomposed into four sub-dimensions:

$$T(b) = w_h \cdot T_{\text{harm}}(b) + w_r \cdot T_{\text{rhythm}}(b) + w_t \cdot T_{\text{timbral}}(b) + w_s \cdot T_{\text{struct}}(b) \tag{28}$$

where the sub-curves are:

- T_{harm}: harmonic tension—chord complexity, dissonance level
- T_{rhythm}: rhythmic tension—syncopation density, metric displacement
- T_{timbral}: timbral tension—filter resonance boost, distortion drive
- T_{struct}: structural tension—distance from tonic key, time since last resolution

By default all weights are $\frac{1}{4}$, but the user can adjust them or edit each sub-curve independently. Chapter-level presets provide common shapes: *Resolved* ($T \approx 0.15$), *Building* ($0.2 \to 0.7$), *Arch* ($0.2 \to 0.8 \to 0.2$), *Sustained* (≈ 0.65), and *Release* ($0.7 \to 0.0$).

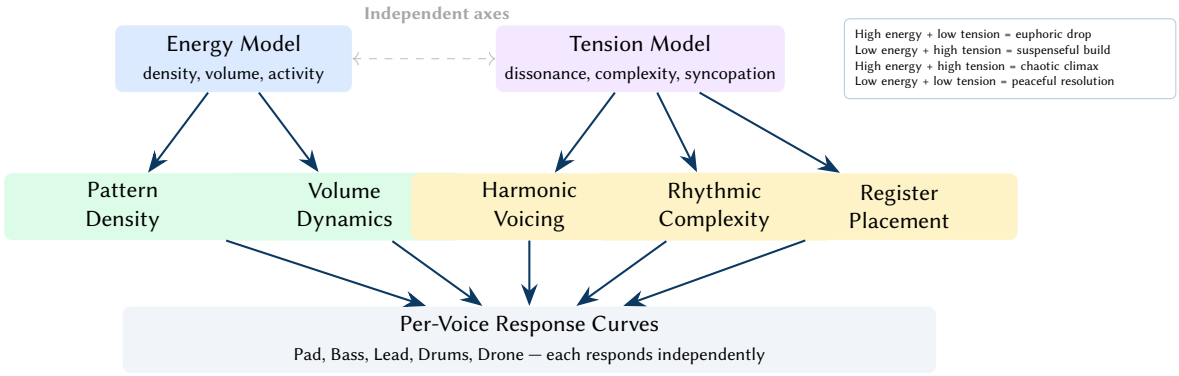


Figure 94. The tension model operates as a parallel control signal alongside the energy model. Energy controls "how much"; tension controls "how tense". Both independently drive per-voice response curves that determine the musical output at each bar.

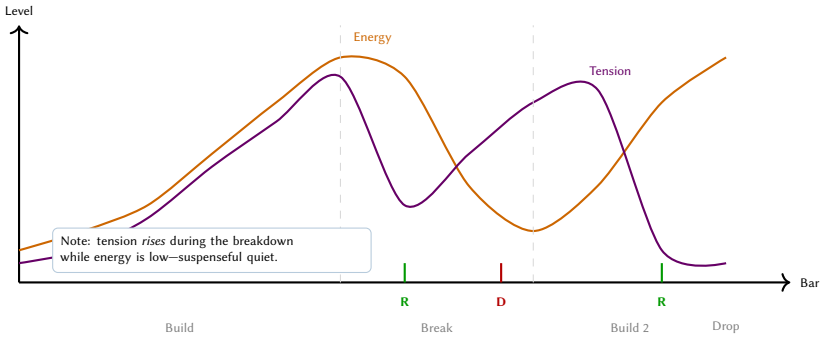


Figure 95. Energy and tension curves for a 64-bar track. Note how they move independently: the breakdown (bars 6–8) has low energy but rising tension. Resolution markers (R, green) and deceptive cadences (D, red) are placed at structural boundaries.

12.2 Chord Voicing Engine

The voicing engine maps the harmonic tension value $T_{\text{harm}}(b)$ to a specific chord vocabulary and voicing density per bar.

Table 16. Chord vocabulary by tension level. The engine selects from the appropriate tier based on T_{harm} at each bar.

Tension Range	Chord Vocabulary	Character
0.00–0.15	Root + 5th, octave doubling	Full resolution
0.15–0.30	Triads, simple inversions	Gentle motion
0.30–0.50	sus2, sus4, add9, add11, modal interchange	Ambiguous
0.50–0.70	V7, dim7, aug, secondary dominants	Dominant drive
0.70–0.85	9th/11th/13th, #9, b9, tritone subs	Chromatic
0.85–1.00	Clusters, polytonal stacks, quartal voicings	Maximum dissonance

The suspended chord range (0.30–0.50) is particularly important for electronic music. A sus4 chord (root + 4th + 5th) removes the 3rd, creating tonal ambiguity without actual dissonance—the listener cannot tell whether the music is in a major or minor context. This "floating" quality is characteristic of trance breakdowns and ambient passages.

The engine outputs a `ChordVoicing` per bar containing: a set of MIDI pitches, a voicing spread parameter, an inversion indicator, and a harmonic function label (tonic, subdominant, dominant, or chromatic).

12.3 Per-Voice Tension Response

Each synthesiser voice responds to tension via its own response curve, allowing different elements to contribute different facets of tension simultaneously.

Pad Response The pad is the primary carrier of harmonic tension. At $T = 0$, the pad plays root + 5th with a warm low-pass filter and gentle chorus. As tension rises, it progresses through full triads, suspended voicings,

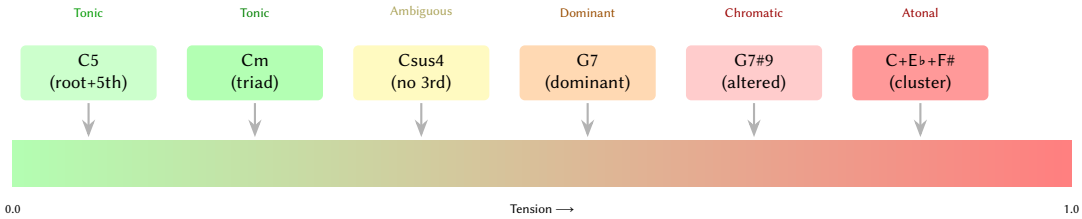


Figure 96. Chord vocabulary mapped along the tension gradient. As tension increases, the voicing engine draws from progressively more dissonant chord types. Chord function shifts from tonic (stable) through dominant (pulling) to atonal (chaotic).

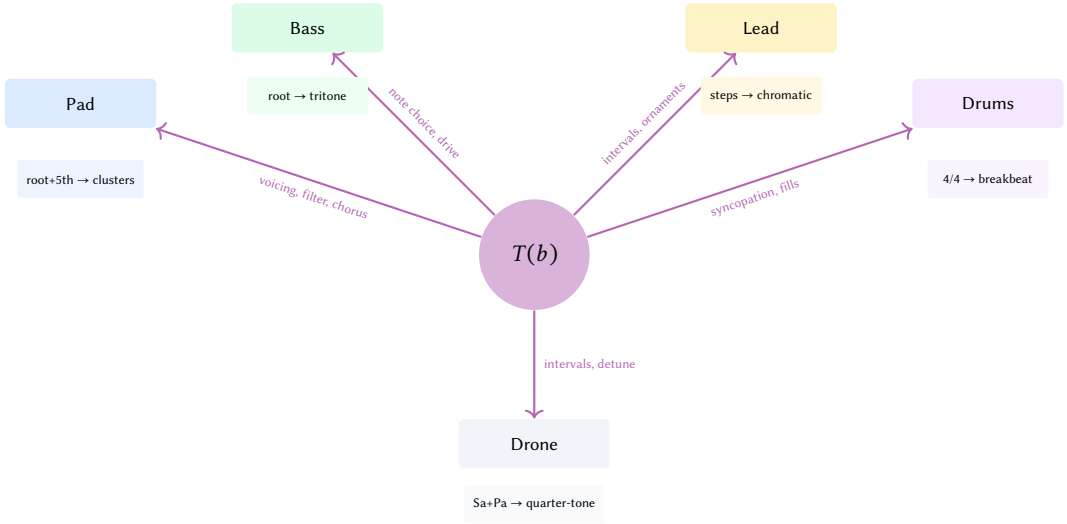


Figure 97. Per-voice tension response. Each voice receives the tension value and maps it through a voice-specific response curve that controls the relevant musical parameters. The pad carries harmonic tension through chord voicing; the bass through note choice; the drums through rhythmic complexity.

7th/9th stacks, and finally dense semitone clusters with high filter resonance. Chord inversion follows tension: low tension uses root position (stable); high tension uses 2nd inversion (weaker bass support, less grounding). Attack time also lengthens with tension—soft pads at rest, sharp stabs at peak.

Bass Response The bass transitions from a stable root-note foundation to an increasingly destabilising force. At $T < 0.2$ the bass plays root notes with a clean sub timbre, locked to the kick. At $T \approx 0.5$ it begins playing 3rds and 5ths (chord inversions), creating harmonic ambiguity. At $T > 0.7$ the bass plays the 7th (strong dominant pull) or even the tritone—the most unstable interval, creating visceral discomfort that demands resolution. A *bass pedal* technique locks the bass on a single dominant note for 4–8 bars while the harmony moves above it, building massive anticipation.

Drum and Percussion Response The drums respond primarily to T_{rhythm}. At low tension the kit plays a steady four-on-floor pattern. As tension rises, kicks begin to displace (shifting by one 16th-note step), hihats switch from straight 16ths to Euclidean polyrhythms, ghost notes fill gaps, and fills mark tension peaks. At maximum tension, the drum pattern becomes a breakbeat with no clear downbeat. A critical technique is the "pull-back": 1 bar before a resolution point, the drums thin to kick-only or silence, creating a vacuum that makes the resolution hit harder.

Lead Response The lead transitions from consonant stepwise motion to chromatic intensity. At low tension it favours scale-degree steps and returns to the root. At medium tension it introduces suspensions (holding the 4th over a chord change, then resolving to the 3rd) and passing tones. At high tension it uses diminished arpeggios, chromatic enclosure (approaching a target from both half-steps), and tritone leaps.

Drone Response The tanpura drone transitions from a stable Sa + Pa (root + 5th) anchor through Sa + Ma (root + 4th, creating a sus4 feel), then Sa + komal Ni (root + flat 7th, dominant pull), and finally Sa + tritone with quarter-tone detuning at maximum tension.

12.4 Cadential Patterns

Cadences are the primary mechanism for resolution—the moments where accumulated tension is released. The engine formalises five cadence types, each with specific per-voice behaviour:

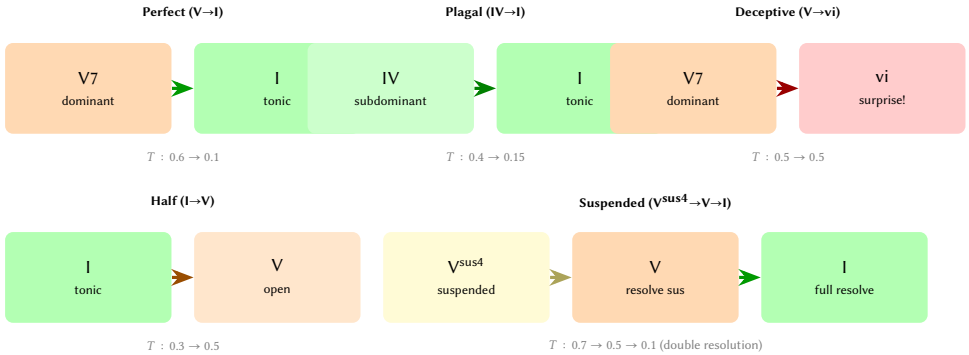


Figure 98. The five cadence types with their harmonic motion and tension delta. The perfect cadence (V→I) provides the strongest resolution; the deceptive cadence (V→vi) subverts expectation and maintains tension; the suspended resolution provides a "double gratification" through two sequential resolutions.

Cadence Placement The engine places cadences automatically at structural boundaries: light cadences (plagal or half) every 8 bars, stronger cadences (perfect or deceptive) every 16 bars, and full perfect cadences at chapter boundaries and the track end. The "almost resolution" technique places a deceptive cadence before the real resolution—the V7 chord begins to resolve but pivots to vi, spiking tension back up. This can be repeated 1–2 times to create a narrative arc where the eventual resolution feels earned.

12.5 Key Modulation Planning

Key changes are planned as part of the harmonic journey rather than applied randomly. Five modulation types are supported, each associated with a characteristic tension range:

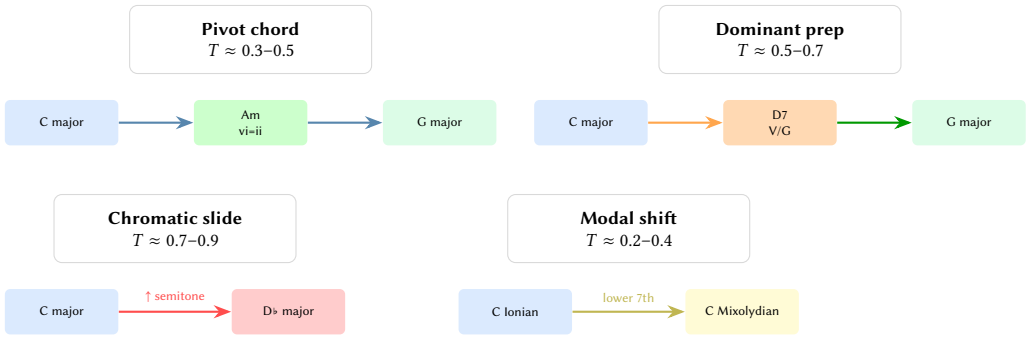


Figure 99. Key modulation types. Pivot chord modulation uses a shared chord as a bridge (smoothest). Dominant preparation inserts V7 of the target key. Chromatic slide moves by a semitone (dramatic, sudden). Modal shift changes one scale degree (subtle colour change, same root).

A default 64-bar modulation plan establishes the home key (bars 1–16), introduces a modal shift (bars 17–32), modulates via dominant preparation (bars 33–48), builds tension in the new key (bars 49–56), and returns home via dominant preparation for a circular resolution (bars 57–64).

12.6 Voice Leading Rules

Smooth chord transitions require voice leading—rules governing how individual pitches move between successive chords:

- **Minimum motion**: each voice moves to the nearest available pitch in the new chord.
- **No parallel 5ths/octaves**: parallel perfect intervals between voices are flagged and corrected (they sound hollow in polyphonic textures).
- **Common-tone retention**: if a pitch appears in both chords, at least one voice holds it.
- **Contrary outer motion**: when possible, the highest and lowest voices move in opposite directions.
- **Tendency tone resolution**: leading tones resolve up to the tonic; chord 7ths resolve down by step; tritones resolve by converging or expanding.

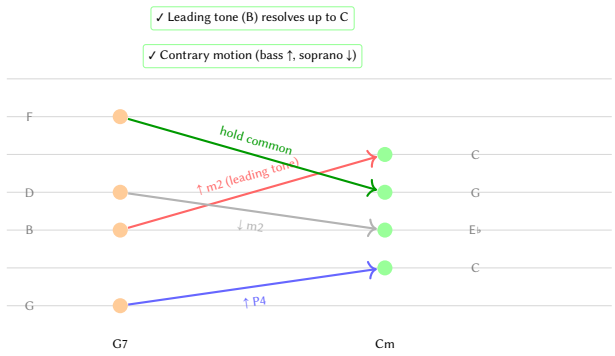


Figure 100. Voice leading from G7 to Cm (perfect cadence in C minor). Each voice moves by the smallest possible interval. The leading tone (B) resolves up to C; contrary motion between bass (ascending 4th) and soprano (descending); common tones are retained where possible.

12.7 Syncopation–Tension Coupling

Rhythmic complexity is driven by T_{rhythm} rather than by energy alone. This enables the powerful combination of low energy with high rhythmic tension (a quiet passage with complex cross-rhythms) or high energy with low rhythmic tension (a loud, driving, straight four-on-floor section).

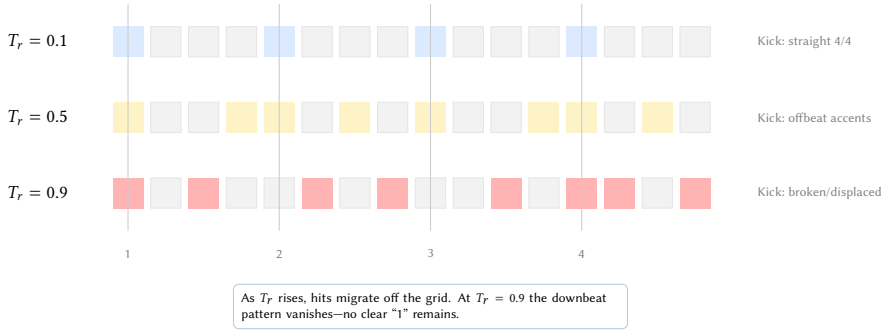


Figure 101. Kick pattern evolution under rhythmic tension. At $T_r = 0.1$ the kick plays a steady four-on-floor. At $T_r = 0.5$ offbeat accents create syncopation. At $T_r = 0.9$ the pattern becomes broken, with no clear downbeat—creating urgent demand for resolution.

Drum–Bass Interaction At low tension ($T_r < 0.3$), the bass and kick are rhythmically locked: every bass note onset coincides with a kick hit. As tension rises, the bass begins to syncopate independently, creating a rhythmic counterpoint. At high tension ($T_r > 0.7$), the kick itself displaces, and bass and kick actively play off each other in a call-and-response pattern. At maximum tension, the two voices are fully independent—the loss of rhythmic foundation creates powerful demand for resolution.

12.8 Resolution Architecture

Resolutions are organised into a four-level hierarchy, each with different scope and impact:

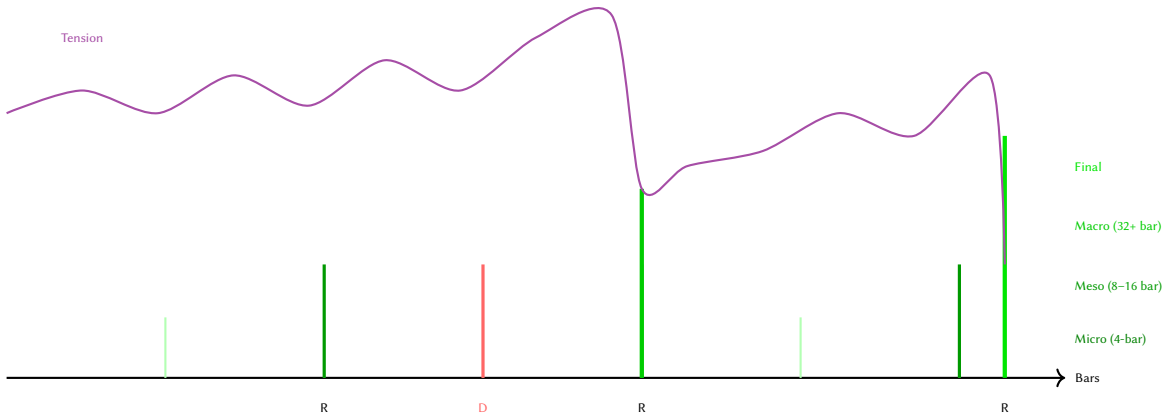


Figure 102. Resolution hierarchy across a 64-bar track. Micro-resolutions provide subtle breathing room every 4 bars. Meso-resolutions mark section boundaries. The macro-resolution ("the drop") is the climactic tension release. The final resolution brings the track home. A deceptive cadence (D) at bar 24 denies resolution, making the eventual macro-resolution more satisfying.

Note

The "Almost Resolution" The deceptive cadence (V→vi) is the engine's primary tool for narrative arc. By beginning a perfect cadence and then pivoting to vi at the last moment, the engine denies the expected resolution. Tension spikes back up, and the listener's desire for resolution intensifies. This technique can be applied 1–2 times before the "real" resolution at a macro boundary—creating the "will they / won't they" narrative arc that makes the eventual resolution feel earned. This maps directly to the *build → fake-out → build → drop* structure common in electronic music.

12.9 Harmonic Rhythm

How fast the chords change is itself a tension device, independent of which chords are playing. The harmonic rhythm parameter H_r maps chord change rate to bars:

Table 17. Harmonic rhythm by tension level.

Tension	Chord Rate	Effect
0.0–0.2	1 chord / 4 bars	Stable, meditative
0.2–0.4	1 chord / 2 bars	Gentle motion
0.4–0.6	1 chord / bar	Active, colourful
0.6–0.8	2 chords / bar	Urgent, restless
0.8–1.0	4 chords / bar	Frantic, overwhelming

Design Decision 11

Pre-Cadence Harmonic Acceleration The most powerful harmonic rhythm technique: during the 4 bars before a macro-resolution, the chord change rate accelerates rapidly. Bar −4: 1 chord/bar. Bar −3: 1 chord/bar. Bar −2: 2 chords/bar. Bar −1: 4 chords/bar (IV → V → V^{sus4} → V7, all in one bar). Bar 0: I chord, *held for 4 bars*. This "harmonic gear-shift" is one of the most viscerally satisfying devices in music—chords pile up faster and faster until the resolution explodes.

12.10 Register and Tessitura as Tension

Which octave a voice plays in creates tension independently of harmony or rhythm. Low register sounds grounded; high register sounds exposed and tense.

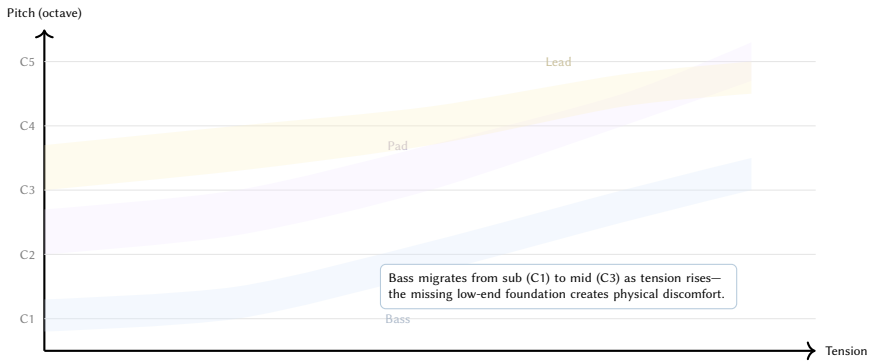


Figure 103. Voice register bands as a function of tension. All voices migrate upward with increasing tension. The bass's migration from sub frequencies into mid range is particularly disorienting—the missing low-end foundation creates a visceral need for resolution.

Inverted Pedal A particularly effective register technique: the lead sustains a high tonic note (e.g., C6) while the pad harmony drifts chromatically below. This creates a "rubber band" effect—the further the harmony wanders from tonic, the more the sustained high note pulls the listener back. When the harmony finally returns to tonic, the held note transforms from dissonant to consonant *without moving*—one of the most elegant tension-resolution devices in tonal music.

12.11 Textural Density and Rhythmic Unison

Textural density measures how many *independent rhythmic streams* are active—distinct from energy (which measures volume and hit count). One voice playing a rhythm is monolithic; seven voices each on their own pattern is chaotic.

Table 18. Textural density by tension level.

Tension	Independent Streams	Character
0.0–0.2	2–3 (kick+bass locked, pad sustained)	Monolithic, simple
0.2–0.4	4 (hihat gains independence)	Groove emerges
0.4–0.6	5 (lead enters with own rhythm)	Polyphonic interest
0.6–0.8	6 (clap/tom add cross-rhythms)	Dense, complex
0.8–1.0	7 (all independent, drone pulses)	Chaotic, overwhelming

Rhythmic Unison Moments The inverse of textural density: moments where *every* voice hits the same rhythm simultaneously. At resolution points, all voices land on beat 1 together (monumental arrival). At tension peaks, all voices hit a syncopated accent together (collective scream). "The Break"—1 beat of total silence across all voices, then unison re-entry—is the single most attention-grabbing moment in any track. These moments are rare (1–2 per 16 bars) because overuse destroys their impact.

12.12 Metric Displacement: Anticipation and Delay

Placing harmonic events slightly before or after their expected metrical position creates a micro-level tension tool:

- **Anticipation** ($T > 0.3$): bass arrives 1 step before the chord change—creates forward momentum.
- **Chord anticipation** ($T > 0.5$): the chord change itself lands on beat 4 instead of beat 1 of the next bar—creates urgency.
- **Delayed resolution** ($T > 0.7$): the resolution chord lands on beat 2 instead of beat 1—"where's the 1?"
- **The delayed drop**: at the climactic macro-resolution, all voices cut to silence on bar N beat 4. Beat 1 of bar N+1 is *silent*. The full resolution hits on beat 2—the 1-beat delay makes the audience hold their breath, doubling the impact when it lands.

12.13 Pedal Tones

Pedal tones are sustained notes that create tension by refusing to move while the harmony changes around them.

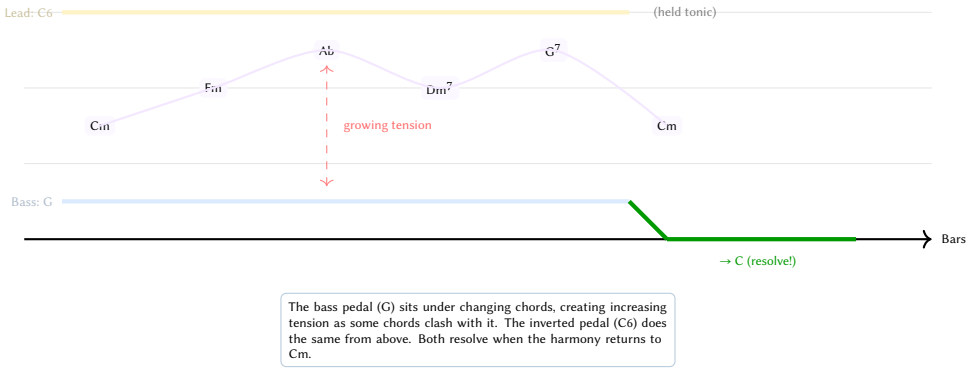


Figure 104. Pedal tones in action. The bass locks on G (dominant pedal) for 8 bars while the pad harmony moves through Cm → Fm → Ab → Dm^7 → G7. Simultaneously, the lead holds a high C6 (inverted pedal). Both pedals create tension that resolves dramatically when the bass drops to C and the harmony arrives at Cm.

Three pedal types are supported:

- **Bass pedal** ($T \approx 0.5$–0.8): the bass locks on the dominant (5th degree) for 4–8 bars. As chords move above it, some consonances become dissonances, building anticipation. When the bass finally drops to the root, the foundation returns—a massive release.
- **Inverted pedal** ($T \approx 0.6$–0.9): a high sustained note acts as a tether. No matter how far the harmony drifts, the held note reminds the listener where home is.
- **Inner pedal** ($T \approx 0.7$–0.9): a middle voice in the pad holds one note while the outer voices move around it, creating a shimmering instability as the held note alternates between consonance and dissonance.

13 Composition Studio: GUI Workflow

The native GUI provides a *Composition Studio* mode—a phrase-level compositional environment where the user manipulates musical ideas (phrases, progressions, tension arcs, rhythmic patterns) rather than individual notes. The design philosophy is **"smart defaults, gestural overrides"**: the engine composes intelligent

tension arcs automatically; every aspect can be reshaped through intuitive visual interactions with immediate audio preview.

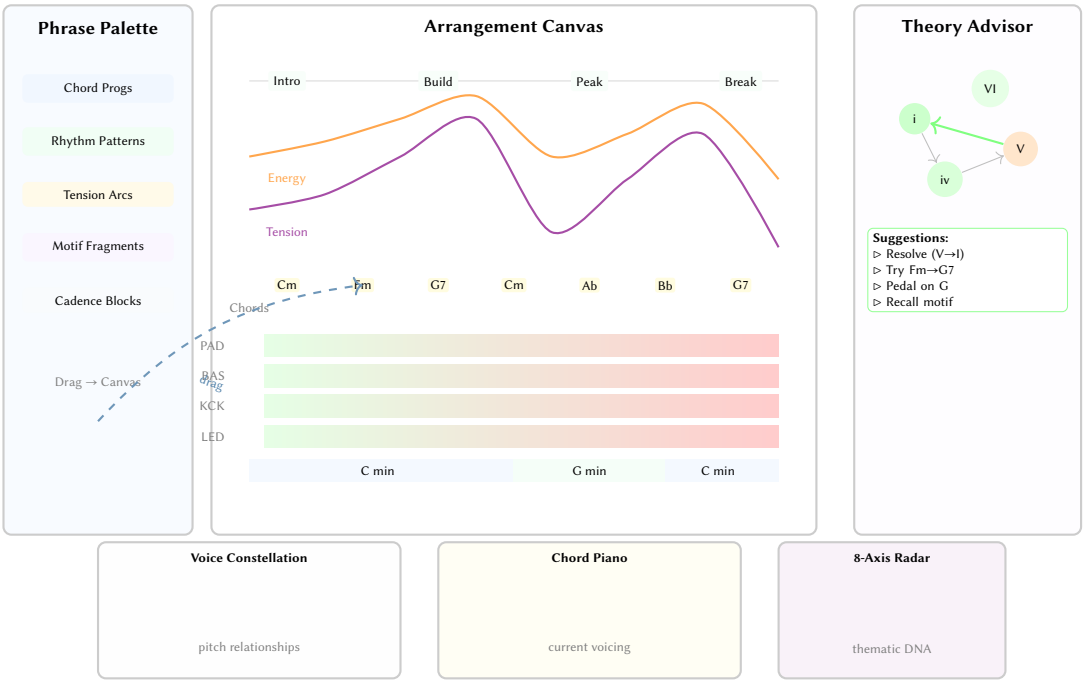


Figure 105. The Composition Studio layout. The **Phrase Palette** (left) contains draggable musical ideas. The **Arrangement Canvas** (centre) shows the multi-lane timeline with energy, tension, chords, voice activity, and key regions. The **Theory Advisor** (right) provides context-aware suggestions and an interactive chord map. Bottom panels show real-time voice relationships, current voicing, and the 8-axis thematic radar.

13.1 The Phrase Palette

The key innovation of the Composition Studio is that users work at the *phrase level*—not placing individual notes, but dragging complete musical ideas onto the arrangement canvas.

- **Chord progressions**: pre-built sequences labelled by emotional character ("Home → Away → Home" = I–IV–V–I; "Endless climb" = I–V–vi–IV; "Dark descent" = Andalusian cadence). Each shows a mini tension-profile sparkline. Dropped onto the harmonic strip, they replace chords and auto-adjust the tension curve.
- **Rhythmic patterns**: drum and bass patterns labelled by feel ("Four-on-floor", "Breakbeat", "Tala: Teental", "Poly 3:4"). Each shows a step-grid preview. Dropped onto a voice lane, they auto-transpose and density-adjust.
- **Tension arcs**: multi-bar tension curves ("Slow build", "Build–deny–resolve", "Catharsis"). Dropped onto the tension lane, they replace the curve for that region.
- **Motif fragments**: captured melodic phrases from Phase 29 (section 12.8), shown as contour sketches. Dropped onto a voice lane, the engine transposes to the current key and chord.
- **Cadence blocks**: complete resolution patterns ("Perfect V→I", "Double deny + resolve"). Dropped onto any bar, they insert full multi-voice cadence behaviour.

13.2 The Theory Advisor

A context-aware music theory guide that watches the current compositional state and generates ranked suggestions.

The advisor generates ranked suggestions based on arrangement position, current tension level, time since last cadence, time in the current key, and recently used chords. Each suggestion includes a human-readable explanation, a **[Preview]** button (2–4 second audio snippet), an **[Apply]** button, and a tension delta indicator.

> **Note**
>
> "Teach Me" Mode When enabled, every auto-generated harmonic decision gets an explanatory annotation in the arrangement canvas. For example: "Pad plays G^{sus4} here because tension is 0.45—suspended chords create ambiguity without dissonance. The sus4 shares two tones with the previous Cm chord (C and

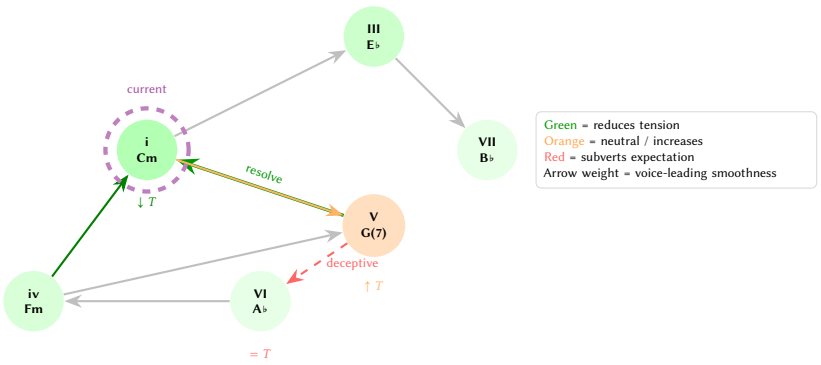


Figure 106. The interactive chord map for C minor. The current chord (Cm, circled) pulses in the UI. Arrows show voice-leading motions: thick arrows indicate smooth transitions. Colour indicates tension effect: green reduces tension, orange increases it, red (deceptive) subverts expectation. Clicking any chord previews a 2-second audio snippet.

> G), maintaining voice-leading smoothness." This turns the composition process into a music theory lesson—users absorb theory naturally through contextual explanations of each engine decision.

13.3 The Voice Constellation

A real-time visualisation showing how all voices relate at the current playback position—the "x-ray view" of the composition.

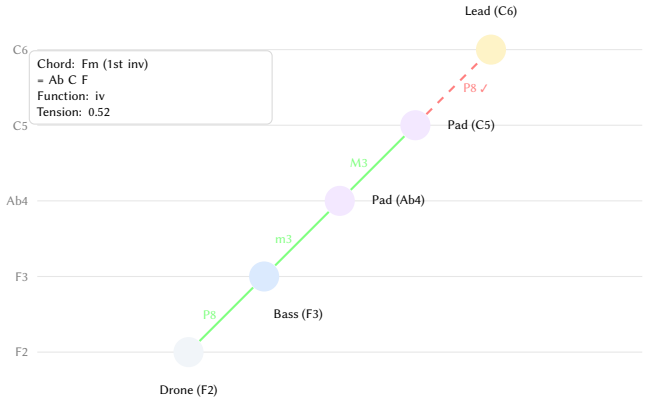


Figure 107. The Voice Constellation at bar 24. Each sounding pitch is a node in a vertical pitch space. Solid lines show consonant intervals; dashed lines show dissonant intervals. The panel updates in real-time during playback, showing the harmonic relationships unfold moment to moment.

13.4 Live Jam Mode

The Composition Studio includes a *Live Jam Mode* that mirrors the real-time improvisational workflow of Ableton's Session View. The user performs compositional gestures in real time—adjusting tension, triggering cadences, modulating keys, muting voices—and every gesture is recorded into a timeline that can be refined after the fact.

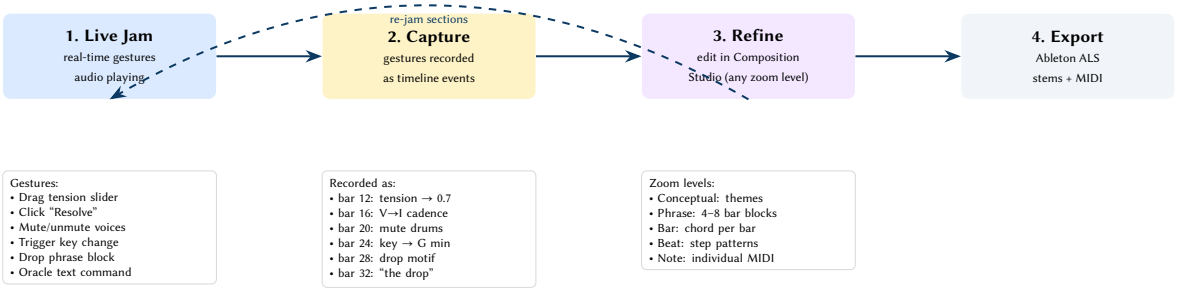


Figure 108. The Live Jam workflow. The user performs compositional gestures in real time (phase 1). Every gesture is captured as a timestamped event (phase 2). The timeline can then be refined at any zoom level in the Composition Studio (phase 3) and exported to Ableton or as stems (phase 4). The user can re-jam specific sections to iterate.

Gesture Recording During a jam session, the engine plays audio continuously (via the streaming engine) while the user interacts with the GUI controls. Every interaction is recorded as a timestamped `JamEvent`:

- Tension slider movements → tension curve control points
- Resolution/cadence button clicks → cadence markers at the current bar
- Voice mute/unmute toggles → arrangement mute automation
- Key modulation triggers → modulation events in the key plan
- Phrase drops from the palette → phrase insertions at the current bar
- Oracle text commands → intent events with bar timestamps

After the jam, these events form a complete compositional timeline that can be played back, edited, and refined.

Multi-Level Zoom The recorded timeline supports five zoom levels, allowing the user to move seamlessly between conceptual composition and note-level detail:

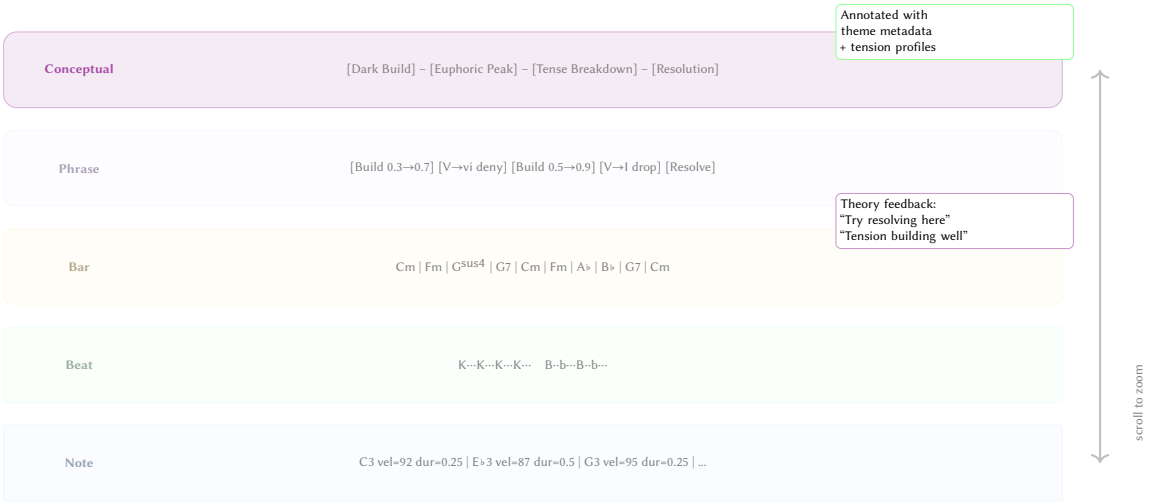


Figure 109. Five zoom levels for timeline editing. Scroll-wheel zooms seamlessly from the conceptual level (themes and emotional arcs) through phrases, bars, and beats down to individual MIDI notes. Each level shows appropriate annotations: theme metadata at the top, theory feedback at phrase level, chord symbols at bar level, step patterns at beat level, and MIDI parameters at note level.

At the highest zoom level, chapters are labelled with their thematic identity ("Dark Build", "Euphoric Peak") and annotated with metadata vectors. At the phrase level, tension arcs and cadence patterns are visible. At bar level, individual chord symbols appear. At beat level, the 16-step grid shows per-voice patterns. At note level, individual MIDI events with velocity, duration, and pitch are editable—the full piano-roll experience when needed.

Compositional Feedback At the conceptual and phrase zoom levels, the Theory Advisor provides structural feedback:

- "You've been building tension for 16 bars without resolution— consider adding a cadence at bar 32."
- "The tension dropped from 0.7 to 0.2 here but energy stayed high—this is a great euphoric release moment."
- "This key modulation (Cm → Gm) uses no preparation— a dominant preparation (D7 for 2 bars) would smooth the transition."
- "The opening motif hasn't returned since bar 4— restating it here would create thematic closure."
- "Three consecutive deceptive cadences may fatigue the listener— consider resolving on the next attempt."

This feedback is generated by rule-based analysis of the arrangement state (time since last cadence, tension curve shape, motif recurrence) and could, in future work, be enhanced with an LLM-based compositional advisor that understands broader musical narrative and style.

Export to Ableton The completed timeline exports as:

- **Ableton ALS project**: an XML project file with all stems as tracks, tempo automation, and clip names matching the chapter structure.

- **Labeled stems**: `kick.wav`, `bass.wav`, `pad.wav`, etc., with a `stems.json` metadata sidecar containing BPM, key, tension curve, and chord progression per bar.
- **MIDI**: per-voice MIDI files preserving the chord symbols and tension annotations as MIDI CC automation or marker text events.

The user can freely iterate between jamming, refining in the Composition Studio, and pushing to Ableton for further production work.

13.5 Phrase-Level Editing

When the user selects a region across bars on any voice lane in the arrangement canvas, a phrase-level toolbar appears offering musical transformations:

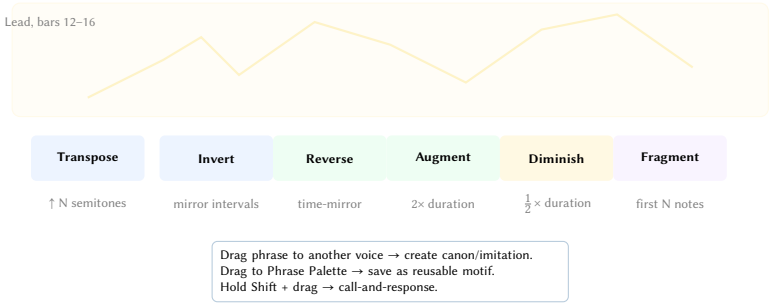


Figure 110. Phrase-level editing. Selecting a region on a voice lane shows the melodic contour and a toolbar of musical transformations. Each transform operates on the entire phrase as a unit—the user never needs to manipulate individual notes. Dragging the phrase to another voice creates polyphonic textures (canon, imitation, call-and-response).

Each transformation operates on the complete phrase as a musical unit: **Transpose** shifts all pitches by N semitones; **Invert** mirrors intervals around the centre pitch; **Reverse** plays notes in reverse order; **Augment** doubles durations (half-speed reprise); **Diminish** halves durations (double-speed intensification); **Fragment** keeps only the first 2–4 notes (teaser or foreshadowing).

Multi-voice selection (Ctrl+click across voice lanes) enables texture-level transforms: "thin out" (reduce to 2 voices), "full ensemble" (all voices active), "unison rhythm" (all voices play the same pattern), and "rhythmic scatter" (give each voice independent timing).

13.6 Sound Design Workflow

The Composition Studio governs *what* the engine plays; the Sound Design Workflow governs *how* it sounds. Rather than exposing the hundreds of low-level parameters that characterise each synthesiser (section 8), the workflow presents a reduced set of *macro knobs*, a preset inheritance tree for iterative patch development, A/B comparison, bounded randomisation, automation capture, and global timbral coherence controls.

13.6.1 Macro-Knob Interface

Each synthesiser exposes between four and eight *macro knobs*—high-level descriptors such as "Brightness", "Aggression", "Space", or "Warmth"— that map a single $[0, 1]$ scalar to a weighted combination of internal parameters. The mapping is encoded in a `MacroMapping` struct:

Listing 1. MacroMapping definition.

```
pub struct MacroTarget {
    pub param: &'static str,  // e.g. "filter.cutoff_hz"
    pub lo:    f64,
    pub hi:    f64,
    pub curve: f64,           // 1.0 = linear, 2.0 = squared
}

pub struct MacroMapping {
    pub name:    &'static str,
    pub targets: &'static [MacroTarget],
}
```

At render time, each path is resolved and interpolated: $v = \text{lo} + (\text{hi} - \text{lo}) \cdot m^{\text{curve}}$, where $m \in [0, 1]$ is the macro value.

13.6.2 Preset Inheritance

Presets form an acyclic inheritance tree rather than a flat list. Every preset stores only the *delta* from its parent: a sparse map of `(param_path, value)` pairs. The engine resolves the full parameter set by walking the ancestor chain from root to leaf and applying overrides in order. Saving a "variant" means writing a new node whose `parent_id` points to the current preset—the parent is never mutated.

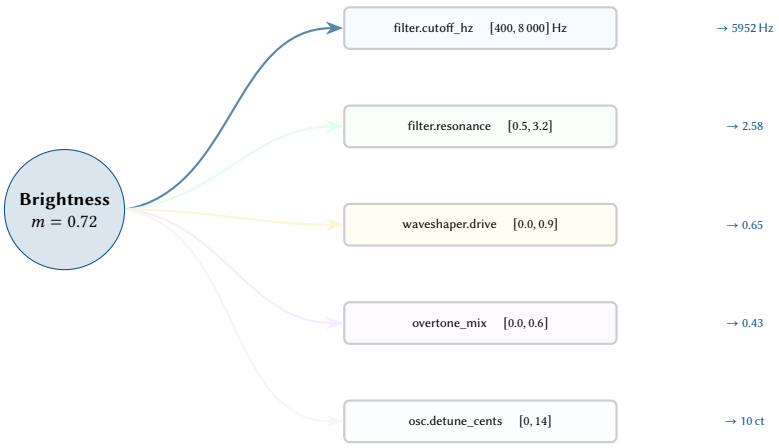


Figure 111. Macro-knob fan-out for the "Brightness" control. One scalar $m = 0.72$ resolves to five coordinated internal parameter values.

Table 19. Standard macro knobs for bass and lead synthesisers.

Macro	Primary targets	Secondary targets	Default
Brightness	`filter.cutoff_hz`, `filter.resonance`	`waveshaper.drive`, `overtone_mix`	0.45
Aggression	`waveshaper.drive`, `adsr.attack_ms`	`filter.resonance`, `lfo.depth`	0.30
Space	`reverb.wet`, `delay.feedback`	`pan_spread`, `chorus.depth`	0.55
Warmth	`osc.detune_cents`, `filter.cutoff_hz`	`adsr.release_ms`, `lfo.rate_hz`	0.60

13.6.3 A/B Patch Comparison

Two `GeneratorConfig` snapshots (A and B) can be loaded simultaneously. A dedicated toggle key swaps the active slot without re-rendering the full arrangement; only the affected voice is re-synthesised for the current preview bar.

13.6.4 Randomise-Within-Constraints

The "Surprise me" function generates a new parameter set guaranteed to lie within a named *constraint space*. Built-in spaces include `"acid_bass"` (fixes filter type to LowPass, constrains resonance to $[2.5, 4.0]$), `"ambient_pad"`, `"dark_drone"`, `"tabla_texture"`, and `"sitar_classical"`.

13.6.5 Sound Design Recording

Any parameter touched in the Sound Design panel is appended to a time-series buffer, enabling full automation replay. Each event is a tuple `(bar, beat_frac, param, value)`; the buffer is stored in the session alongside the preset tree and intersected with the render at each bar boundary.

13.6.6 Cross-Voice Timbral Coherence

Global timbral controls shift all voices simultaneously via a `GlobalTimbralState` that acts as a secondary modulation layer applied after per-voice macros. `brightness_offset` translates into a per-voice filter cutoff delta (in log-frequency space) and a proportional waveshaper drive reduction, so "Darken all" consistently dulls every element.

Table 20. Global timbral control routing across primary voices.

Control	Kick	Bass	Pad	Lead
Brightness↑	Click +3 dB	HPF ×1.4	Chorus +20%	Overtone +0.15
Brightness↓	Click −3 dB	LPF ×0.7	Reverb +8%	Cutoff −15%
Space↑	Room +6%	Delay +10%	Reverb +15%	Delay +0.12
Saturation↑	Sub +0.1	Tanh −0.05	Bus sat +0.2	FM fb +0.08

Design Decision 12: Coherence vs. Autonomy

The tension between per-voice customisation and global cohesion is resolved by the layering order: individual macros first (giving each voice character), then global offsets (imposing ensemble identity). Neither layer can drive a parameter outside its safety bounds—the biquad cutoff clamp of $0.45\,f_s$ and the resonance cap of $Q = 20$ remain enforced.

14 Streaming Engine

The batch pre-render pipeline described in section 6 produces a complete track in a single pass. The `streaming` module provides an alternative: a sample-accurate, event-driven engine that processes audio in blocks, enabling real-time playback with lower latency and incremental state updates.

14.1 Architecture

The `StreamingEngine` pre-sorts all note events by absolute sample position at construction time, then processes them via a linear cursor scan during block rendering.

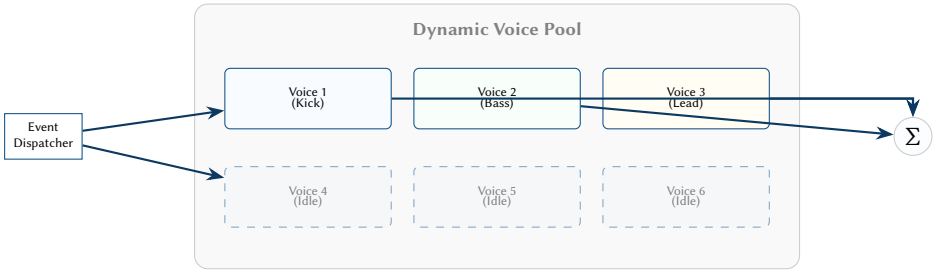


Figure 112. Dynamic voice pool management. The engine maintains a pool of pre-allocated DSP voices. When an event is dispatched, an idle voice is activated and assigned a synthesis model. Once the amplitude envelope reaches silence, the voice returns to the idle state, minimizing real-time allocations.

The engine maintains per-element synth instances (kick, bass, hi-hat, clap, lead, pad, tom, drone), an effects chain (sidechain compressor, ping-pong delay, Schroeder reverb, master HPF), and an optional breath LFO for ambient modulation.

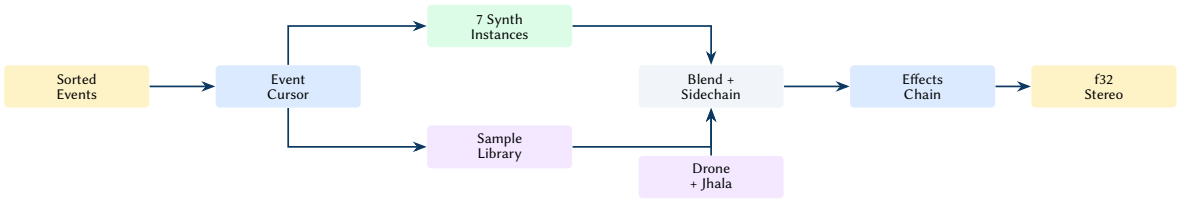


Figure 113. Streaming engine architecture: pre-sorted events are dispatched sample-accurately, blended with energy-aware sample playback, and processed through the effects chain.

14.2 Block Processing

For each sample frame in a block, the engine:

1. Computes the current bar index from the global sample clock and triggers bar-level sample placements (FX, risers, textures, vocals) at bar boundaries.
2. Dispatches all scheduled events at the current sample position, triggering the appropriate synth with energy-aware blend ratios ($r = \min(0.8 \cdot E_{\text{macro}}, 0.8)$).
3. Renders all active synth voices and sample playback instances, applying sidechain compression to the bass.
4. Optionally modulates volume (±15%), filter cutoff (±30%), and effects wet level (±20%) via the breath LFO.
5. Adds the drone and jhala layers (stereo).
6. Processes the stereo bus through the delay, reverb, and master HPF chain.
7. Outputs interleaved `f32` stereo samples.

14.3 Synth/Sample Blending

Each element's blend ratio is computed from the macro energy at the current bar. The blending strategy varies by element:

- **Kick** — synth velocity scaled by $(1 - r_{\text{blend}})$; sample played at full velocity.
- **Bass** — at blend ≥ 0.8, sample-only mode (synth bypassed entirely).
- **Hi-hat/Clap** — at blend ≥ 0.5, sample preferred; synth used as fallback.
- **Lead** — pure synthesis (no sample blending).
- **Pad** — voiced as a minor chord (root, +3, +7, +10 semitones).

14.4 Sample Library

The `SampleLibrary` indexes loaded samples by (SampleCategory, EnergyZone) pairs, with round-robin counters per pair to prevent repetition. Samples are classified into nine categories (Kick, Snare, Hat, Clap, Bass, FX, Riser, Texture, Vocal) and three energy zones (Low, Mid, High). Zone assignment uses RMS level thresholds:

Low below 0.08, Mid between 0.08 and 0.22, High above 0.22. For energy-aware selection during playback, zones are alternatively assigned from the macro energy curve: Low below 0.35, Mid between 0.35 and 0.70, High above 0.70.

When a category has samples in only one energy zone, the library automatically redistributes them into Low, Mid, and High zones by sorting on RMS level and dividing into equal thirds. Queries fall back through zones in priority order (requested → Mid → remaining) to maximise sample utilisation.

14.5 Playback and Live Parameters

The `PlaybackHandle` wraps a `cpal` audio stream with lock-free parameter control via `AtomicU64` (f64 bit patterns) and `AtomicUsize` for filter type. Four live parameters are exposed: cutoff frequency, resonance, gain, and filter type.

The audio callback checks for parameter changes each block using threshold guards ($|\Delta f_c| > 0.1\,\text{Hz}$ or $|\Delta Q| > 0.01$) and recomputes biquad coefficients only when needed. Two playback modes are supported:

- **Buffer playback** — pre-rendered samples are read sequentially with per-sample live filtering and gain.
- **Streaming playback** — the `StreamingEngine` is locked per block, live parameters are injected into its config, and `process_block()` renders directly into the output buffer.

The audio callback has no external synchronisation dependencies; tempo and transport state are communicated asynchronously via the TCP/JSON bridge events (section 17.3.5), which update the `GeneratorConfig` in the TUI thread rather than the audio callback.

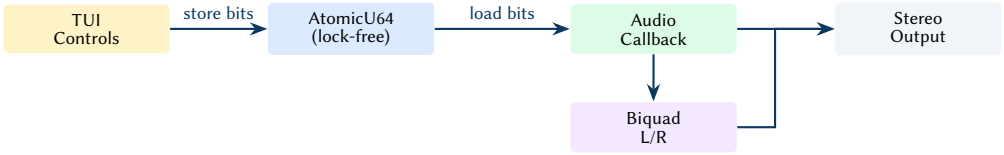


Figure 114. Lock-free live parameter path: the TUI stores f64 bit patterns into atomics; the audio callback loads them each block and updates the biquad filter only when thresholds are exceeded.

15 Session and Preset Management

15.1 Sessions

A `Session` captures the complete generator state (name, UTC timestamp, full `GeneratorConfig`, and an optional `ImportedMidiSnapshot`) as a JSON file. Sessions are named `{name}_{YYYYMMDD_HHMMSS}.json` and stored in a user-specified directory. The `list_sessions()` function scans the directory and returns all sessions sorted newest-first.

15.2 Presets

A `Preset` snapshots the audio-relevant subset of the generator configuration—mood, energy curve, filter settings, delay/reverb wet levels, pattern presets for all elements, sample blend overrides, and humanisation level—tagged with a name, timestamp, and search tags. Unlike sessions, presets omit runtime state (mute/solo, playback position) and focus on reproducible sound design.

The `from_config()` constructor extracts preset parameters from a live `GeneratorConfig`, and `apply_to()` writes them back, enabling rapid A/B comparison and preset morphing workflows.

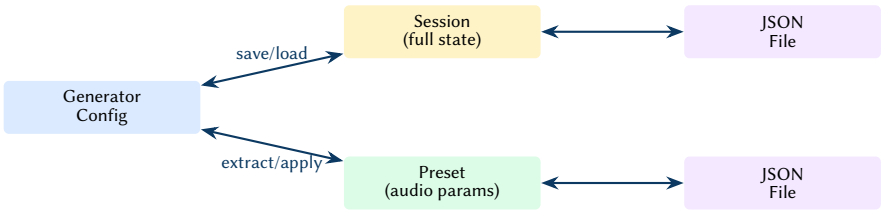


Figure 115. Session and preset persistence: sessions capture full state, presets capture audio parameters only. Both serialise to JSON.

With the synthesis pipeline and supporting subsystems in place, the final part describes the user-facing workflow: the terminal interface, Ableton integration, and evaluation of the complete system.

Part V — Workflow & Integration

16 Terminal User Interface

The TUI is implemented with `ratatui` [6] and `crossterm`, running at a 50 ms frame cadence. It provides a state machine with six screens: Launch, Playback, StepEdit, SessionList, PresetList, and EnergyEditor.

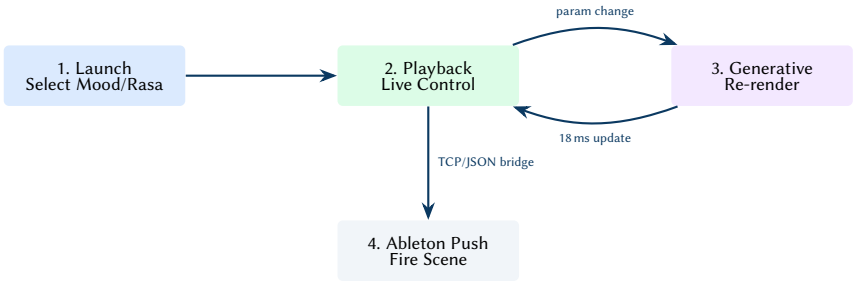


Figure 116. The user workflow journey. Selecting a mood configures the global state. Adjustments during playback trigger an instantaneous generative re-render, with structural updates pushed seamlessly to Ableton Live.

16.1 Screen State Machine

The interface logic is structured as a finite state machine, where each screen handles input events and updates the global `AppState`.

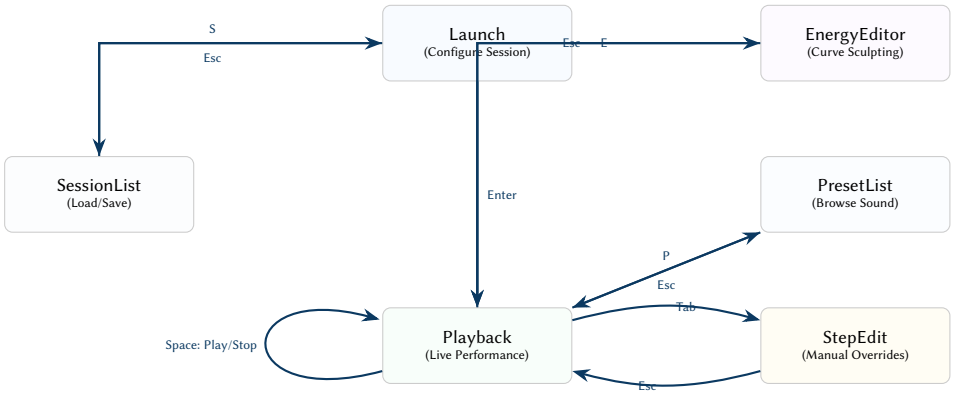


Figure 117. The TUI screen state machine. Users navigate between configuration, performance, and manual editing screens via hotkeys. The Playback screen acts as the hub for real-time control, while specialized editors provide deep access to the generative and synthesis models.

- **Launch** — parameter configuration: mood selection, ... BPM/bars adjustment, theme browsing, key/scale overrides, filter/LFO/arp settings, sample directory selection.
- **Playback** — live visualisation of pattern grids (4-bar view with per-step markers), energy curve sparkline, waveform display, element mute/solo/density controls, Ableton connection status indicator, and real-time parameter knobs. Recent revisions also expose the `humanize` amount, live filter-mode switching (biquad vs. ladder), chapter-local syncopation overrides, and master toggles for macro modulation and Mid/Side processing.
- **StepEdit** — per-bar step editing with toggle-able 16th-note grid cells. Overridden patterns bypass the generative algorithm for the edited bar.
- **SessionList / PresetList** — browsable lists with timestamps and metadata, supporting load, save, and delete operations.
- **EnergyEditor** — direct manipulation of macro curve control points.

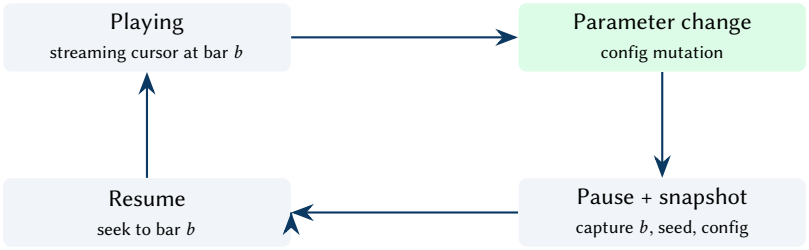
16.2 Re-Render Protocol

Parameter changes trigger a full re-render: the TUI stops playback (recording the current bar position), constructs a new `Generator`, calls `render()`, and resumes playback from the same bar offset. At 18 ms per render (section 18), this produces an imperceptible gap in audio output.

16.3 Theme System

Thirty-four theme presets are defined as a static array, each bundling a mood, BPM, key, scale, energy curve, DSP parameters, syncopation style pairings, and a five-dimensional metadata vector into a coherent sonic identity. Themes span six broad categories:

- **Classic psy:** Dark Phrygian, Acid Machine, Goa Sunrise, Goa Matrix
- **Uplifting:** Euphoric Rise, Solar Flare, Morning Glory, Astral Projection



Conceptual point. The TUI loop is a deterministic state machine: config is the single source of truth, and every change triggers a replayable render step. The pause/snapshot boundary ensures the UI can restart playback at the same musical time coordinate (bar b) after regeneration.

Figure 118. Re-render-on-change as a state machine. The engine treats rendering as a pure function of config (and seed), enabling deterministic regeneration during live playback.

- **Hard / aggressive:** Hitech Assault, Aggressive Drive, Quantum Drop, Temple of Bass
- **Moody / atmospheric:** Midnight Tears, Lunar Eclipse, Mariana Trench, Deep Forest
- **Experimental:** Alien Signal, Neural Link, Zenon Swamp, Suomi Sprites
- **Eastern / tala-influenced:** Ganges Dawn, Kashi Nights, Temple Pulse, Savasana, Desert Night

Applying a theme overwrites all relevant fields in the `GeneratorConfig` via `apply_to()` and seeds a single-chapter timeline with explicit chapter metadata and per-chapter bass/hi-hat syncopation defaults. Each theme's `ThemeMetadata` carries five perceptual axes—darkness, euphoria, tension, groundedness, and complexity—that flow into the chapter's metadata and can be queried by the narrative oracle and arrangement heuristics.

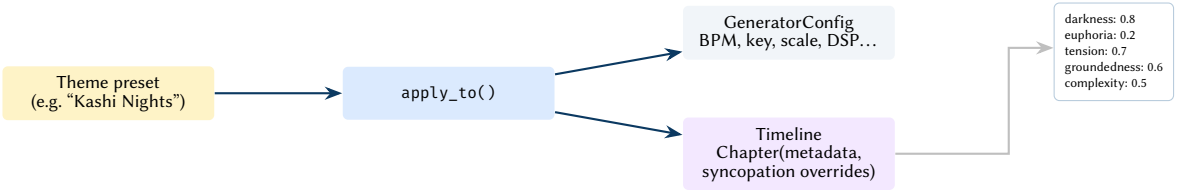


Figure 119. Theme application flow. A theme preset populates the `GeneratorConfig` and creates a single-chapter timeline with metadata and syncopation overrides.

Table 21. Representative themes with syncopation style pairings.

Theme	Bass Syncopation	Hi-hat Syncopation	Category
Dark Phrygian	PsyGroove	PsyGroove	Classic psy
Acid Machine	Syncopated	Shuffle	Classic psy
Kashi Nights	PsyGroove	Straight	Eastern
Hitech Assault	Straight	PsyGroove	Hard
Midnight Tears	Shuffle	Straight	Moody
Alien Signal	Syncopated	Syncopated	Experimental

16.4 Filter Mode Selection

The `FilterMode` enum selects between two filter implementations for the bass voice at runtime:

- **Biquad** (default) — the RBJ biquad (section 9.1), supporting low-pass, high-pass, band-pass, and notch modes.
- **Ladder** — the four-pole ladder (section 9.2), low-pass only with resonant self-oscillation character.

When the ladder is selected but the voice requests a non-LP filter type, the engine falls back silently to the biquad. The mode is stored in `GeneratorConfig` and flows through the render pipeline into `BassFilterParams`, where the per-sample `process()` call dispatches to the correct implementation.

Live switching is available via `Shift+F` in the TUI and radio buttons in the native GUI. The playback path supports auditioning the filter mode change during output processing without a full re-render.

16.5 Tala Circle Visualisation

The `tala_circle` module renders a circular beat indicator in the TUI, displaying the current tala cycle as a ring of glyphs. Each beat position is mapped to polar coordinates ($\theta = 2\pi \cdot b/B - \pi/2$, starting from top-centre) and rendered with type-specific glyphs:

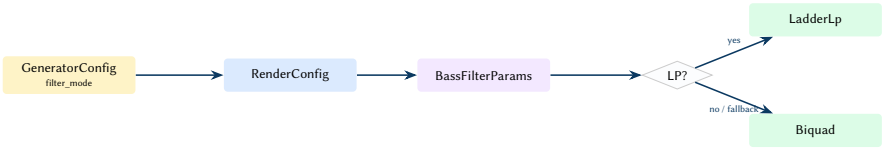


Figure 120. Filter mode dispatch. The selected `FilterMode` flows from config to per-sample processing. Non-LP filter types fall back to the biquad regardless of mode selection.

- **Sam** (downbeat) — red filled circle, yellow arrow when active.
- **Khali** (empty beat) — blue double circle.
- **Normal** — grey open circle.

The visualisation respects the tala's vibhag structure (groupings of beats with assigned accent types), providing a musically meaningful display of rhythmic position within the cycle.

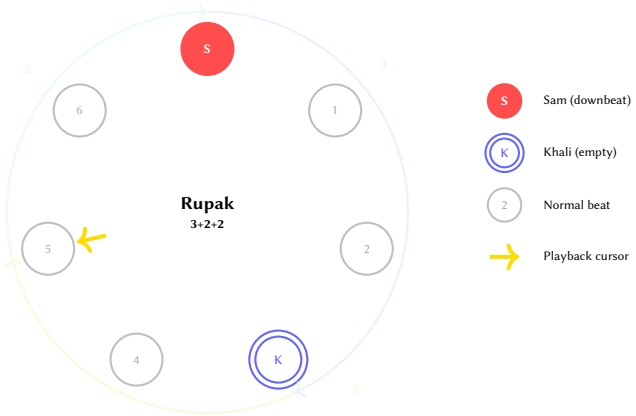


Figure 121. Tala circle visualisation as rendered in the TUI. Beats are arranged clockwise from the top (Sam). Vibhag groupings (3+2+2 for Rupak) are shown as arcs. The playback cursor (yellow arrow) indicates the current position within the cycle.

16.6 Energy Colour Mapping

The TUI uses a four-tier colour gradient to represent energy levels throughout the interface (pattern grid cells, energy sparkline, status indicators):

Table 22. Energy-to-colour mapping in the TUI.

Energy Range	Colour	Meaning
< 0.33	Blue	Low energy (intro, breakdown)
0.33–0.66	Green	Moderate energy (build)
0.66–0.85	Yellow	High energy (drive)
≥ 0.85	Red	Peak energy (climax)

17 Ableton Integration

The generator integrates with Ableton Live through two complementary mechanisms: MIDI file interchange (export and import) and a real-time TCP/JSON bridge to a custom Python Control Surface running inside Ableton. The bridge carries commands, acknowledgements, and event notifications over a single TCP connection, replacing the previous dual-channel architecture (UDP/OSC + Ableton Link) with a unified, reliable protocol.

17.1 Design Rationale

The original integration relied on two independent communication channels: (1) the third-party AbletonOSC remote script over UDP ports 11000/11001, using the `rosc` crate for OSC encoding; and (2) Ableton Link via `rusty_link`—a Rust FFI binding to Ableton's GPL-2.0 C++ Link library—for tempo and phase synchronisation. This dual-channel architecture introduced several problems:

- **UDP unreliability.** OSC messages are fire-and-forget. Pushing a 7-element clip set required empirical sleep delays (20–100 ms) between commands; even with retries, clips occasionally arrived empty.

- **Timing races.** The per-element delete → create → add_notes sequence could not guarantee clip allocation between UDP packets. A create arriving before the previous delete completed would silently fail.
- **Dual-channel complexity.** Tempo from Link callbacks and track state from OSC required two polling loops, two connection states, and complex coordination in the TUI event loop.
- **GPL-2.0 contamination.** `rusty_link` forced the entire binary into GPL compliance, restricting distribution.
- **Excessive surface area.** AbletonOSC exposes hundreds of OSC addresses; Overtone uses approximately 15 commands.

Design Decision 13: Single-Channel TCP/JSON Bridge

The replacement collapses both channels into a single TCP connection to a purpose-built Python Control Surface (`OvertoneRemote`). Every command receives a JSON acknowledgement with a matching `id`, and compound operations like `push_clips` execute atomically inside Ableton's main thread—eliminating timing races, sleep delays, dropped messages, and the GPL dependency in a single architectural change.

17.2 Architecture Overview

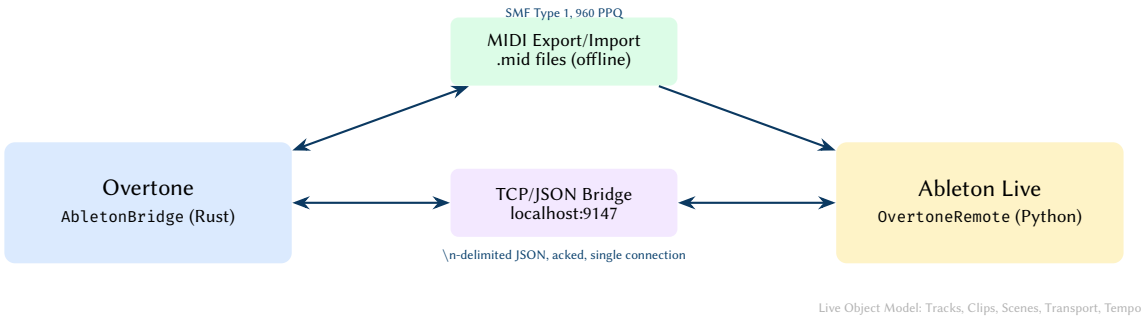


Figure 122. Ableton Live integration architecture. Overtone communicates via Standard MIDI Files for offline interchange and a single TCP/JSON connection for real-time clip pushing, scene control, and tempo/transport synchronisation. The custom `OvertoneRemote` Control Surface replaces both the third-party AbletonOSC script and the Ableton Link library.

Key properties of the new architecture:

- **Single channel.** One TCP connection carries commands, responses, and event notifications. No separate Link session.
- **Acked protocol.** Every command receives a JSON response with a matching `id` field, confirming success or reporting an error.
- **Compound operations.** The `push_clips` command sends all clip data in a single message; Python executes the full delete → create → name → add_notes sequence synchronously inside Ableton's main thread.
- **Track resolution in Python.** Rust sends track names (now typically "Kick – Mood" or "Bass – Mood"), not numeric indices. Python resolves names against Live's track list and auto-creates missing tracks.
- **Richer status reporting.** The bridge and UI now surface connection state, discovered track counts, and missing-track diagnostics so the performer can see whether a Live set is ready to accept clip pushes before triggering them.

17.3 Protocol Specification

17.3.1 Transport Layer

The protocol operates over TCP on `localhost:9147` using newline-delimited JSON (one JSON object per `\n`-terminated line, UTF-8 throughout). Only a single client connection is accepted; new connections replace the existing one.

17.3.2 Command/Response Flow

Every command from Rust includes a monotonically increasing `id` (u64). Every response from Python includes the matching `id` and an `ok` boolean:

Listing 2. Command/response exchange examples.

```
Rust -> Python:  {"id":1, "cmd":"get_tempo"}
Python -> Rust:  {"id":1, "ok":true, "tempo":142.0}
```

```
Rust -> Python:  {"id":2, "cmd":"fire_clip", "track":99, "slot":0}
Python -> Rust:  {"id":2, "ok":false, "error":"list index out of range"}
```

17.3.3 Command Reference

Table 23 lists the complete command set.

Table 23. TCP/JSON command reference. All commands carry an `id` field (omitted for brevity). Response fields beyond `ok` are listed in the rightmost column.

Command	Parameters	Response
`get_tempo`	—	`tempo`
`get_track_names`	—	`tracks`
`create_midi_track`	`index`	—
`set_track_name`	`track`, `name`	—
`set_track_color`	`track`, `color`	—
`create_clip`	`track`, `slot`, `length`	—
`delete_clip`	`track`, `slot`	—
`set_clip_name`	`track`, `slot`, `name`	—
`add_notes`	`track`, `slot`, `notes`	—
`fire_clip`	`track`, `slot`	—
`fire_scene`	`scene`	—
`start_playing`	—	—
`stop_playing`	—	—
`set_tempo`	`bpm`	—
`create_scene`	`index`	—
`set_scene_name`	`scene`, `name`	—
`get_num_scenes`	—	`num_scenes`
`get_scene_name`	`scene`	`name`
`subscribe_beat`	—	—
`unsubscribe_beat`	—	—
`push_clips`	`clips`: array of clip objects	`pushed`

17.3.4 The Compound `push_clips` Command

This command is the central reliability improvement. A single message contains all clip data for all elements. Recent revisions also qualify the target track name with the active mood so multiple stylistic track groups can coexist in one Live set without name collisions:

Listing 3. Compound `push_clips` payload.

```
{"id":23, "cmd":"push_clips", "clips":[
  {"track":"Kick - Dark Phrygian", "slot":0, "length":128.0,
   "name":"Kick - Dark Phrygian",
   "notes":[[36,0.0,0.25,100],[36,4.0,0.25,100]]},
  {"track":"Bass - Dark Phrygian", "slot":0, "length":128.0,
   "name":"Bass - Dark Phrygian",
   "notes":[[40,0.0,1.0,90],[42,4.0,0.5,85]]}
]}
```

Python executes the following for each clip entry, synchronously within Ableton's main thread:

1. Resolve the track by name (case-insensitive, then substring match).
2. If no match exists, create a new MIDI track and set its name/colour.
3. Ensure enough scenes exist for the requested slot index.
4. Delete any existing clip in the target slot.
5. Create a new clip with the specified length.
6. Set the clip name and add all notes via `clip.add_new_notes()`.

The response confirms the count: `{"id":23, "ok":true, "pushed":7}`.

17.3.5 Unsolicited Events

Python pushes state-change notifications without a command `id`:

Listing 4. Unsolicited event examples.

```
{"event":"tempo", "bpm":145.0}
```

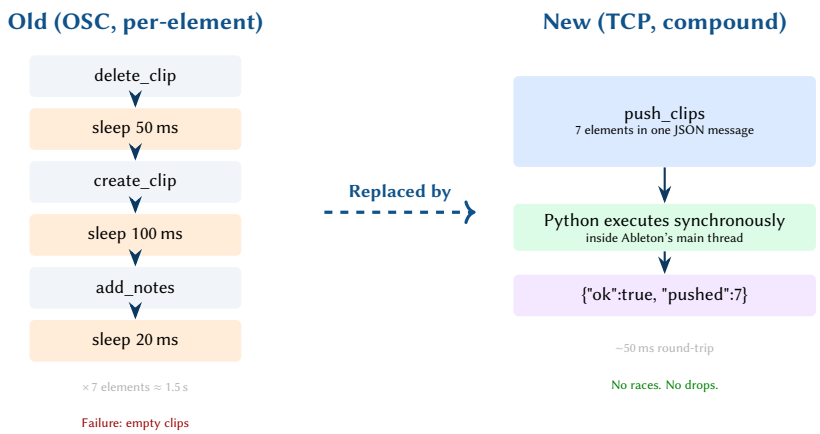


Figure 123. Comparison of clip push strategies. The old per-element OSC approach required seven sequential command bursts with empirical sleep delays (~1.5 s total, prone to empty clips). The new compound `push_clips` sends a single TCP message; Python executes all operations synchronously (~50 ms round-trip, zero failures).

```
{"event":"transport", "playing":true}
{"event":"beat", "beat":3}
{"event":"tracks_changed",
 "tracks":["Kick","Bass","Hihat","Clap","Lead","Pad","Tom"]}
```

Table 24. Unsolicited event types pushed from the Python Control Surface to the Rust client. These replace both the OSC listener callbacks and the Ableton Link tempo/transport callbacks from the previous architecture.

Event	Fields	Trigger
`tempo`	`bpm`: f64	Live tempo changes
`transport`	`playing`: bool	Play/stop state changes
`beat`	`beat`: i32	Each beat tick (when subscribed)
`tracks_changed`	`tracks`: [str]	Track added/removed/renamed

17.4 MIDI Export

The `midi` module generates Standard MIDI Files (SMF Type 1, 960 PPQ) using the `midly` crate. Each non-muted element is written as a separate track. Drum elements use General MIDI standard note numbers (kick = 36/C1, closed hi-hat = 42/F#1, clap = 39/D#1). Bass notes are converted from frequency to MIDI pitch via $p = 69 + 12\log_2(f/440)$.

17.5 MIDI Import Strategy: Generative Overriding

The `midi_import` module parses Standard MIDI Files and classifies tracks into generator lanes (Kick, Bass, Hihat, Clap, Other) using a two-stage mapping strategy:

1. **Track name matching** — case-insensitive keyword search for "kick", "bass", "hihat"/"hat", "clap"/"snare" in the MIDI track name meta event.
2. **General MIDI drum note** — fallback to GM note numbers: 36 → Kick, 42/46 → Hihat, 38–40 → Clap.

Imported MIDI acts as an *override* layer. When the `Generator` or `StreamingEngine` encounters an element with an imported pattern in its `GeneratorConfig`, it bypasses the generative algorithms and uses the imported notes directly.

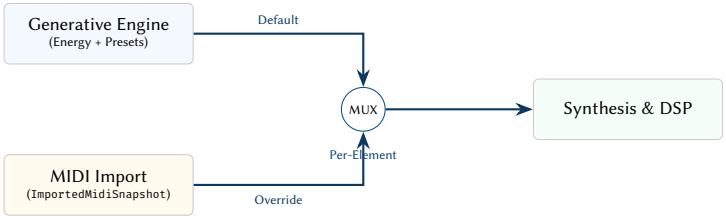


Figure 124. MIDI Import overriding architecture. Users can selectively replace the generative output of specific elements (e.g., locking in a custom bassline) while allowing the engine to continue generating the rest of the arrangement (drums, leads, drones).

The user can select the classification mode: TrackNameThenGm (default, tries names first), TrackNameOnly, or GmOnly. The parser maintains a `HashMap<(channel, key), (start_tick, vel)>` of active notes, closing

them on NoteOff or zero-velocity NoteOn. Imported notes are sorted per lane by tick and stored in an `ImportedMidiSnapshot` alongside the session for recall.

17.6 Python Control Surface Architecture

The Control Surface is installed into Ableton's MIDI Remote Scripts directory and selected in Preferences → Link, Tempo & MIDI → Control Surface. It consists of two files:

- `__init__.py` — the `OvertoneRemote` Control Surface class (~280 lines): command dispatch, compound `push_clips` handler, event listeners.
- `tcp_server.py` — non-blocking TCP server using `select.select()` (~110 lines): connection management, line-buffered JSON parsing.

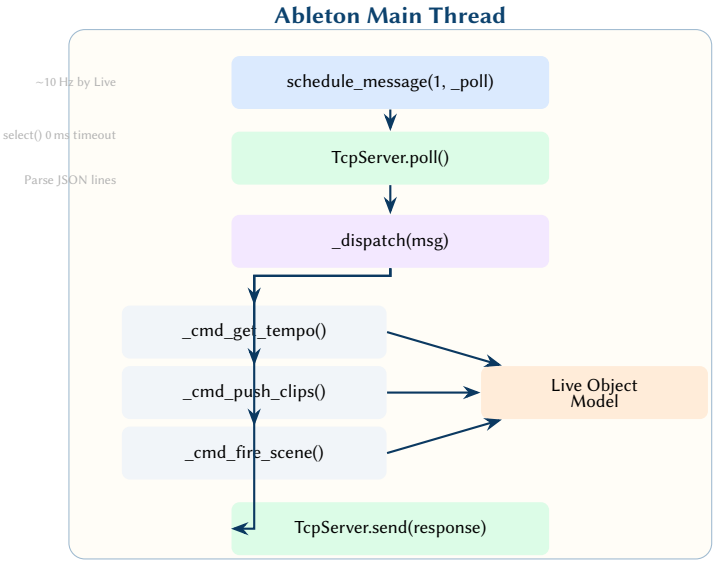


Figure 125. Python Control Surface execution model. Ableton's main thread calls `_poll` approximately 10 times per second via `schedule_message`. Each invocation polls the non-blocking TCP server, dispatches parsed JSON commands to handler methods that access the Live Object Model directly, and sends JSON responses back over TCP.

17.6.1 Non-Blocking I/O

Ableton's Python runtime is single-threaded and does not support blocking I/O in Control Surface callbacks. The `TcpServer` class uses `select.select()` with zero timeout to poll for new connections and incoming data:

Listing 5. Non-blocking TCP polling in Ableton's timer callback.

```
def _poll(self):
    # Re-schedule before any work (ensures poll continues on error)
    self.schedule_message(1, self._poll)
    msgs = self._server.poll()  # select() with 0ms timeout
    for msg in msgs:
        self._dispatch(msg)
```

17.6.2 Event Listeners

The Control Surface registers listeners on Ableton's Song object:

- `add_tempo_listener` → pushes `{"event":"tempo", "bpm":...}`
- `add_is_playing_listener` → pushes `{"event":"transport", "playing":...}`
- `add_tracks_listener` → pushes `{"event":"tracks_changed", "tracks":[...]}`
- `add_current_song_time_listener` → pushes `{"event":"beat", "beat":...}` (when subscribed)

17.6.3 Track Auto-Creation

When `push_clips` references a track name that does not exist, the Python handler creates a new MIDI track:

17.7 Rust Client Architecture

17.7.1 Module Structure

```
src/ableton/
  mod.rs          # AbletonBridge, job system, note conversion
  protocol.rs     # Serde types for commands, responses, events
```

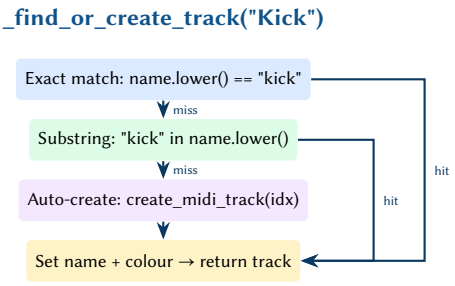


Figure 126. Track name resolution in Python. The handler tries exact case-insensitive match, then substring match, then auto-creates a new MIDI track. Element colours are assigned from a predefined palette matching Ableton's colour index system.

Table 25. Element colour assignments for auto-created Ableton tracks.

Element	Colour Index	Visual
Kick	13	Red
Bass	69	Deep blue
Hihat	26	Yellow
Clap	5	Orange
Lead	48	Cyan
Pad	37	Purple
Tom	61	Olive

17.7.2 Connection Model

The bridge maintains **two TCP connections** to the Python server:

1. **Main connection** — used by the TUI thread for interactive commands (`get_tempo`, `fire_scene`). A background reader thread parses incoming lines and forwards them through an `mpsc` channel, drained by `poll()` each frame.
2. **Worker connection** — used by the background job thread for long-running `push_clips` operations. Blocking reads with a 5-second timeout isolate the TUI from clip push latency.

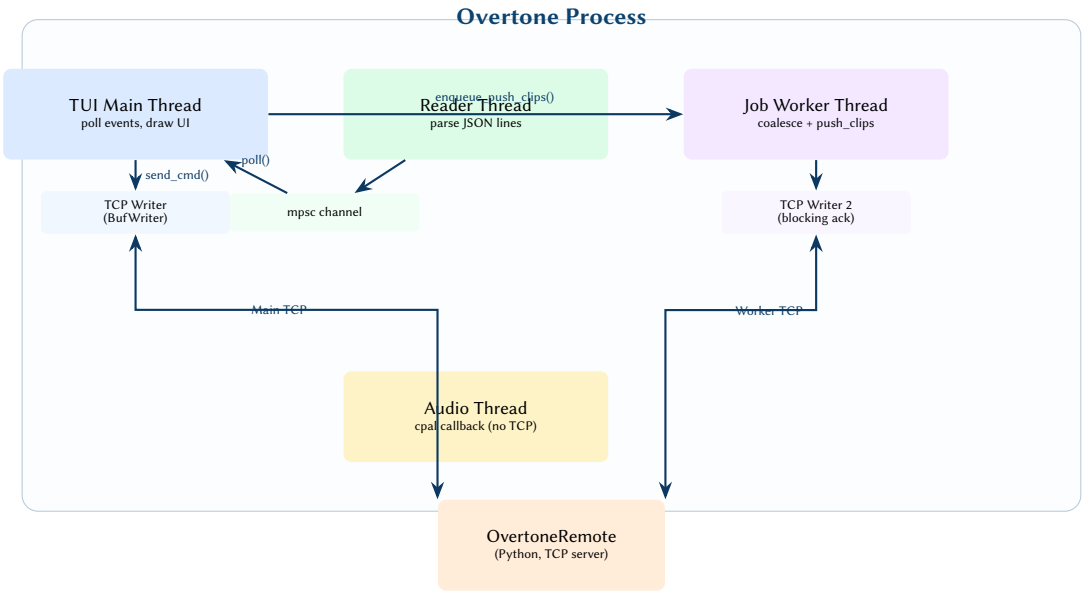


Figure 127. Thread architecture and TCP connection model. The TUI thread sends interactive commands via the main TCP connection and receives responses/events through an `mpsc` channel fed by the reader thread. The job worker thread uses a separate blocking TCP connection for `push_clips` operations, isolating the UI from clip push latency. The audio thread has no TCP or synchronisation dependencies.

17.7.3 Protocol Types

Listing 6. Core protocol types for the TCP/JSON bridge (`protocol.rs`).

```
/// Clip data for the compound push_clips command
pub struct ClipData {
```

```
    pub track: String,        // Track name (resolved by Python)
    pub slot: i32,            // Scene slot index
    pub length: f32,          // Clip length in beats
    pub name: String,         // Display name
    pub notes: Vec<NoteData>, // [pitch, start, duration, velocity]
}

/// Parsed incoming message
pub enum IncomingMessage {
    Response(Response),       // Acked command response
    Event(AbletonEvent),      // Unsolicited push notification
}

/// Unsolicited events (serde tagged enum)
pub enum AbletonEvent {
    Tempo { bpm: f64 },
    Beat { beat: i32 },
    Transport { playing: bool },
    TracksChanged { tracks: Vec<String> },
}
```

17.7.4 Connection Status Model

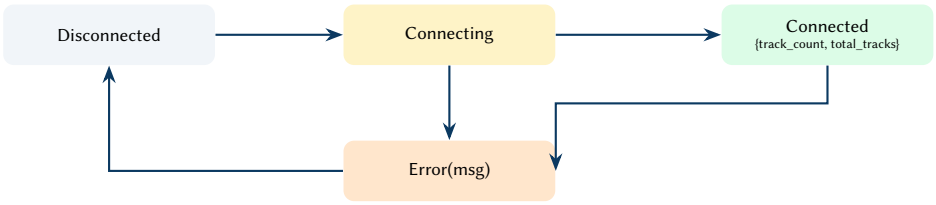


Figure 128. Connection status state machine. The `Connected` state reports how many of the 7 expected element tracks were found in Ableton's track list, displayed in the TUI as "Ableton: 7/7" or "Ableton: 4/7".

Track matching uses a multi-pass algorithm: (1) exact case-insensitive match, (2) alias match ("bd" → Kick, "hh" → Hihat), (3) substring match ("1-Kick" → Kick).

17.7.5 Job Coalescing

Background push jobs use an `mpsc` channel with job-type coalescing. When the user triggers rapid re-renders (e.g., cycling through themes), only the most recent clip set is pushed:

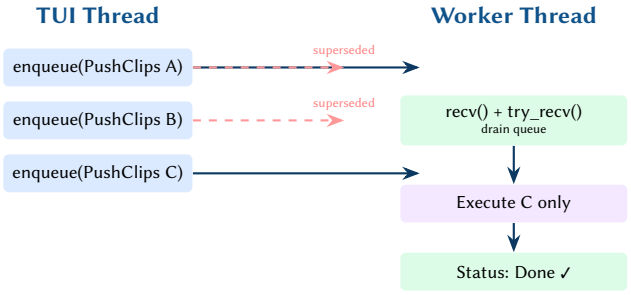


Figure 129. Job coalescing in the worker thread. When multiple push jobs arrive during a busy period, the worker drains the channel and keeps only the latest job per type, avoiding a queue of stale clip data.

17.8 Sequence Diagrams

17.8.1 Initial Connection

17.8.2 Clip Push

17.8.3 Tempo Sync Event

17.9 Comparison with Previous Architecture

17.10 Reliability Analysis

17.10.1 TCP vs UDP for DAW Control

For localhost DAW control (not network-wide performance), TCP's properties are strictly superior to UDP:

- **Ordering.** In-order delivery guaranteed.
- **Reliability.** No silent packet loss; connection drops are detected.
- **Flow control.** Backpressure from Ableton automatically throttles Rust.

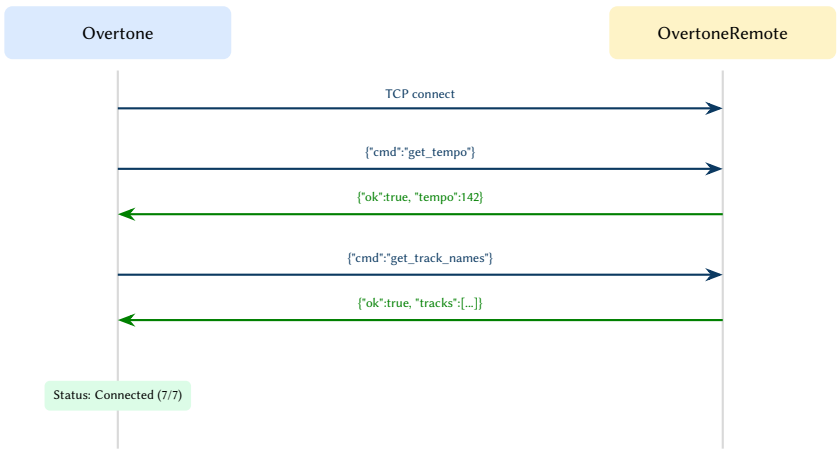


Figure 130. Initial connection handshake. On TCP connect, Overtone queries tempo and track names to populate the connection status and verify track availability.

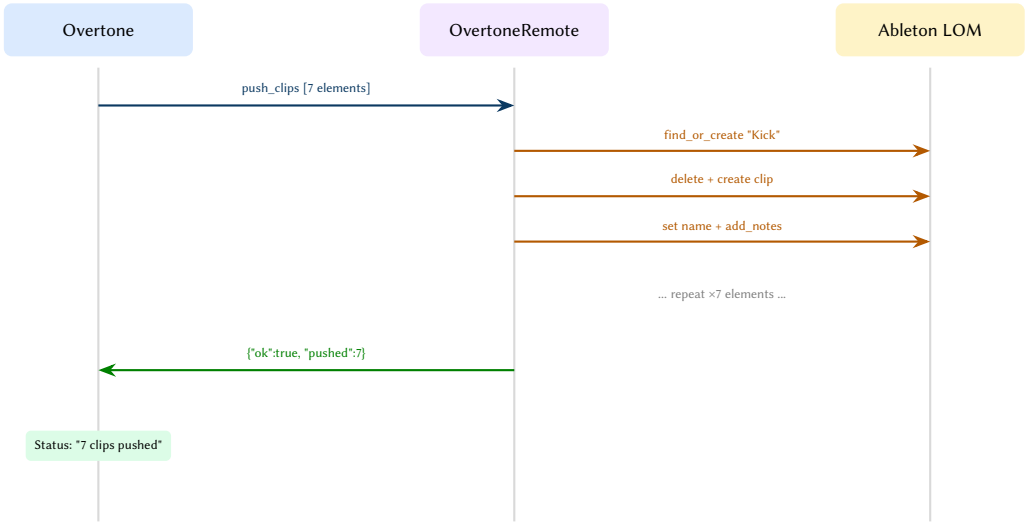


Figure 131. Compound clip push sequence. A single TCP message triggers synchronous execution of all track resolution, clip creation, and note insertion within Ableton's main thread. The ack confirms the exact count of clips pushed.

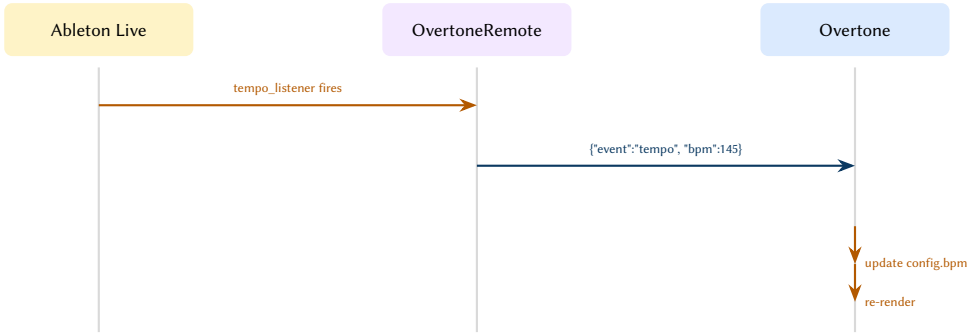


Figure 132. Tempo synchronisation via unsolicited event. When Ableton's tempo changes, the Python listener pushes a `tempo` event; Overtone updates its `GeneratorConfig` and triggers a re-render.

- **No MTU limits.** A `push_clips` payload with 7 elements and 500 notes each is ~50 KB—well beyond UDP's ~1500-byte MTU, which previously required chunked note batches with inter-batch delays.

17.10.2 Connection Lifecycle

17.11 Note Conversion Pipeline

The note conversion from Overtone's internal `NoteEvent` representation to Ableton's MIDI note format produces `ClipData` structs:

$$p_{\text{MIDI}} = 69 + 12 \log_2 \left(\frac{f}{440} \right) \tag{29}$$

Table 26. Architectural comparison between the old (OSC + Link) and new (TCP/JSON) Ableton integration.

Aspect	Old (OSC + Link)	New (TCP/JSON)
Transport	UDP (fire-and-forget)	TCP (reliable, ordered)
Delivery	None (empirical sleeps)	Ack per command
Clip push	7×(delete+sleep+notes)	Single `push_clips`, atomic
Timing races	Frequent (empty clips)	Impossible (synchronous)
Track resolution	Rust-side index math	Python-side name resolution
Missing tracks	Manual TUI wizard	Auto-created by Python
Tempo sync	Link callbacks	TCP event push
Channels	2 (OSC UDP + Link network)	1 (TCP localhost)
Dependencies	`rosc`, `rusty_link`	`serde_json` (already present)
License impact	GPL-2.0 (rusty_link)	None (MIT-compatible)
Push latency	~1.5 s (7×200 ms sleep)	~50 ms (single round-trip)
Rust LoC	~1000 (osc/ + link/)	~680 (ableton/)
Python LoC	0 (third-party AbletonOSC)	~380 (purpose-built)

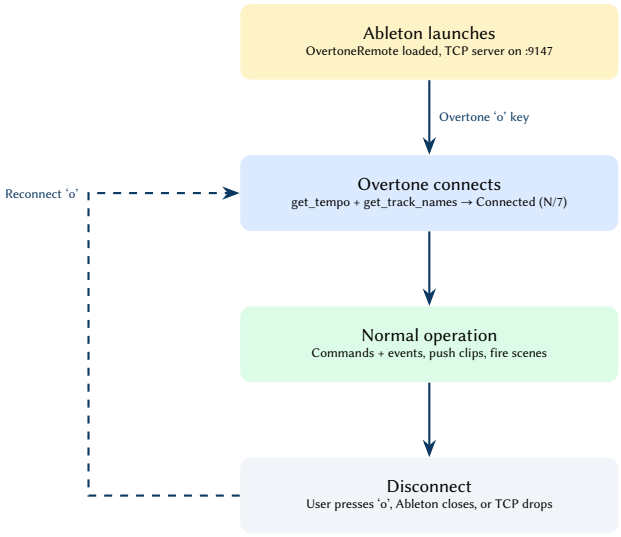


Figure 133. Connection lifecycle. The Python TCP server starts with Ableton and waits for connections. Overtone connects on user request, performs a handshake, and enters normal bidirectional operation. Disconnection is detected automatically; the user can reconnect with the 'o' key.

Table 27. Element-specific MIDI pitch mapping for Ableton clip notes.

Element	Pitch Source
Kick	Fixed: MIDI 36 (GM Bass Drum)
Bass	`freq_to_midi(pitch_hz)`: $69 + 12\log_2(f/440)$
Hihat	Sentinel: $> 0.5 \rightarrow$ MIDI 46 (Open HH), else 42 (Closed HH)
Clap	Fixed: MIDI 39 (GM Hand Clap)
Lead	`freq_to_midi(pitch_hz)`
Pad	`freq_to_midi(pitch_hz)`
Tom	Sentinel: $\leq 0 \rightarrow$ 41 (Dha), $< 0.5 \rightarrow$ 43 (Tin), else 45 (Ghe)

17.12 Workflow: Session View and Templates

Overtone's integration is explicitly designed for Ableton's **Session View** (the non-linear, vertical grid) rather than the horizontal Arrangement timeline. This mirrors how live performers use Ableton to "jam" with structural blocks of a song.

- **Live Performance:** Overtone generates "Scenes" (Intro, Build, Peak, Breakdown, Bridge) representing discrete energy states. By pushing these to the Session View, the user can trigger them dynamically.
- **API Capabilities:** Ableton's Python Remote Script API is built around matrix-style clip launching. Programmatically drawing clips along the horizontal timeline is largely unsupported via the remote API.

A known limitation of Ableton's Remote API is that it **cannot programmatically load VSTs or devices from the browser onto tracks**. Overtone can automatically create empty MIDI tracks and fill them with generated note patterns, but it cannot instantiate a software synthesizer.

To solve this, the standard workflow uses an **Ableton Template**:

1. **Initial Setup:** Sync Overtone to an empty Ableton project to generate the named tracks (`Kick`, `Bass`, `Lead`, etc.).
2. **Sound Design:** Drag desired instruments and effect chains onto those tracks.
3. **Clear Grid:** Delete all generated MIDI clips from the Session View, leaving only the populated tracks.
4. **Save Template:** In Ableton, go to `File → Save Live Set as Default Template`.

Now, whenever Ableton opens, the instruments are already loaded on tracks with the correct names. When Overtone pushes clips, its `_find_or_create_track` function will find the existing tracks by name and inject the new MIDI patterns directly into the pre-configured instruments.

17.13 Installation (Quick Start)

1. Copy `ableton_remote/OvertoneRemote/` to Ableton's MIDI Remote Scripts directory.
2. In Ableton Preferences → Link, Tempo & MIDI → Control Surface, select "OvertoneRemote".
3. No MIDI input/output ports are required.
4. Launch Overtone and press `o` to connect.

> **Installation Paths**
>
> **macOS:** `/Applications/Ableton Live*/Contents/App-Resources/MIDI Remote Scripts/`
> **Windows:** `\ProgramData\Ableton\Live*\Resources\MIDI Remote Scripts\`

17.14 Scene Control

The energy model's arrangement sections (Intro, Build, Peak, Breakdown) are mapped to Ableton scenes. When `follow_energy` mode is enabled, the generator fires the appropriate scene automatically as the bar cursor advances through section boundaries, driven by beat events from the TCP bridge.

18 Evaluation

We evaluate the generator on two axes: rendering throughput and output quality characteristics. All benchmarks were performed on a workstation with an Apple M2 processor and 16 GB of RAM, running Rust 1.82 on macOS 15.

18.1 Rendering Performance

Table 28 reports rendering times for tracks of increasing length. All renders use the *dark* mood preset at 142 BPM with default DSP parameters and no sample blending (pure synthesis).

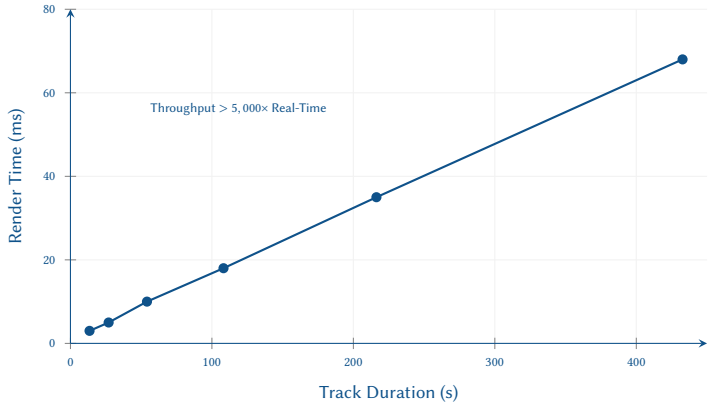


Figure 134. Rendering throughput. The relationship between track length and render time is linear, indicating a constant per-sample processing overhead and zero allocation-driven performance decay for tracks up to 7 minutes long.

The sub-linear scaling (real-time ratio *improves* with track length) reflects the fixed overhead of energy model initialisation and DSP state allocation, which is amortised over longer renders.

...

Table 28. Rendering performance by track length (pure synthesis, 44.1 kHz stereo). The real-time ratio is computed as track duration / render time.

Bars	Duration (s)	Render Time (ms)	Real-Time Ratio
8	13.5	3	4,500×
16	27.0	5	5,400×
32	54.1	10	5,410×
64	108.2	18	6,011×
128	216.3	35	6,180×
256	432.7	68	6,363×

Sample Blending Overhead

When sample blending is active, render times increase by approximately 15–25% due to the additional PCM resampling and gain staging. A 64-bar track with full sample blending renders in approximately 22 ms—still well within the 50 ms TUI frame budget.

18.2 Memory Footprint

The system maintains a low memory profile. The core engine and DSP state allocate statically, requiring less than 50 MB. The `SampleLibrary` manages memory by eagerly extracting `rustfft` profiles during directory scans but caching them in sidecar JSON files (`*.sample.json`). During playback, only the necessary `f32` audio buffers for scheduled samples are loaded into RAM, keeping the footprint tightly bounded even with extensive sample directories.

18.3 Concurrency Analysis

Real-time audio demands non-blocking, wait-free coordination between the TUI thread and the `cpal` audio callback. The engine achieves this via `AtomicU64` types that store `f64` bit patterns (`f64::to_bits()`). When the user adjusts a live parameter (e.g., filter cutoff) in the TUI, it is written atomically. The audio block processor loads these values continuously without acquiring mutexes. This lock-free design ensures the audio thread is never preempted by UI updates, avoiding xruns (buffer underruns) and ensuring a glitch-free 50 ms re-render cadence.

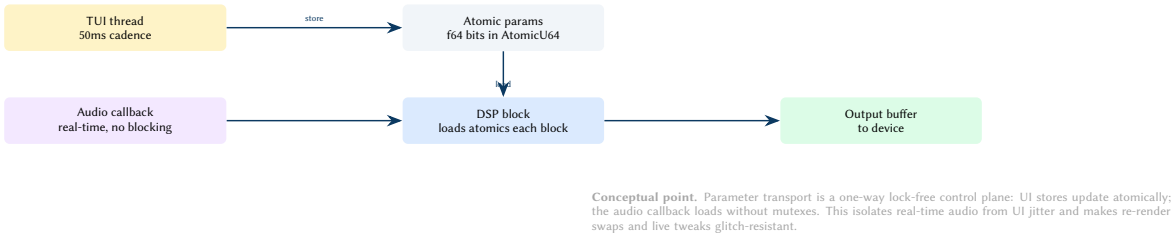


Figure 135. Lock-free control plane between UI and audio threads. Atomics carry parameters across threads without blocking the audio callback.

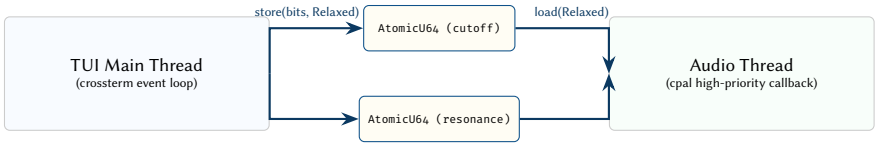


Figure 136. Lock-free concurrency model. The UI thread communicates with the real-time audio thread strictly through atomic variables, preventing priority inversion and guaranteeing that the audio callback never blocks waiting for a mutex.

18.4 Signal Characteristics

The generator produces output with the following signal properties at default settings:

- **Peak level**: −0.5 dBFS (after normalisation and limiting).
- **RMS level**: approximately −12 to −8 dBFS during peak energy sections, consistent with mastered electronic music.
- **Frequency range**: 30 Hz to 20 kHz, bounded by the master HPF and the Nyquist frequency.
- **Stereo field**: kick and bass centre-panned; hi-hat slightly right (pan 0.55); clap slightly left (pan 0.45).
- **Internal precision**: 64-bit floating-point throughout the signal path; 16-bit integer at WAV export.

19 Visualisation Architecture

Overtone produces a continuous stream of DSP telemetry—per-voice amplitudes, spectral data, harmonic ratios, rhythmic phase offsets, tension coefficients, and energy levels—that can drive multiple visualisation modes simultaneously. This section defines the design principles, catalogues the visual modes, and describes how they share a unified render pipeline.

19.1 Design Principles

19.1.1 Information Density vs. Clarity

Music is inherently multi-dimensional. Overtone follows a *layered disclosure* model: a default view presents the three most salient dimensions for the currently active mode, while additional dimensions are accessible through hover, zoom, or panel toggle. Colour encodes at most one scalar per visual element; shape encodes category; size encodes magnitude.

19.1.2 Real-Time Update Budget

The browser render loop targets 60 fps (frame budget ≈16 ms). The `telemetry_buffer` module maintains a ring-buffer of the 64 most recent ticks and exposes a throttled selector that fires at most every 16 ms. Expensive visualisers—particularly the FFT waterfall and the 3-D energy landscape—are rendered entirely in a `requestAnimationFrame` loop via React Three Fiber, reading directly from a shared `Float32Array` typed-array bus.

19.1.3 Colour Semantics

All visualisers share a common colour vocabulary:

Table 29. Shared colour semantics across all visualisation modes.

Token	Hue	Semantic role
`layerA`	Deep teal	Kick / rhythmic foundation
`layerB`	Warm amber	Bass / low-frequency content
`layerC`	Violet	Melodic lead / pitch activity
`layerD`	Sage green	Harmonic / pad layer
`layerE`	Coral pink	Ornament / microtonal detail
`skAccent`	Bright cyan	Cursor, selection, highlight

Tension maps to *saturation*; energy maps to *brightness*.

19.2 Catalogue of Visual Modes

19.2.1 Spectral Waterfall

A scrolling time–frequency display for each active voice. The engine computes a 512-point FFT per voice per render block; the resulting magnitude spectrum is transmitted in the `fft_bins` field of the `StateUpdate` message.

19.2.2 Mood Space Navigator

Each `MoodPreset` is annotated with a 5-dimensional metadata vector $\mathbf{m} = (\text{energy}, \text{darkness}, \text{euphoria}, \text{tension}, \text{groundedness}) \in [0,1]^5$. The mood space navigator projects this via PCA into 2-D. Genre clusters emerge naturally; the active preset is shown as a pulsing `skAccent` dot with a 32-bar fading trail.

19.2.3 Harmonic Series Tree

Visualises overtone relationships between simultaneously sounding voices. The root is the lowest sounding pitch; child nodes are added for every other voice whose f_0 is within 8 cents of an integer multiple of the root.

19.2.4 Rhythmic Phase Wheel

Extends the tala circle (section 16) into a multi-ring polyrhythm display. When dots on different rings align, they are connected by a bright `skAccent` spoke, making polyrhythmic convergence points visible.

19.2.5 Motif Lineage Graph

A timeline-spanning arc diagram. Each motif occurrence is a dot on the baseline; arcs connect source occurrences to derived variants, colour-coded by transformation type: inversion (`layerA`), retrograde (`layerB`), intervallic expansion (`layerC`), energy scaling (`layerD`), and verbatim recurrence (`layerE`).

19.2.6 Tension Heatmap

A matrix with bars as columns and voices as rows. Cell brightness encodes the tension contribution of that voice at that bar, as computed by the `TensionCurve` (section 12).

Figure 137. Mood space 2-D scatter. Each dot represents a `MoodPreset` projected via PCA onto the first two principal components. Genre clusters emerge naturally from the 5-D metadata vectors.

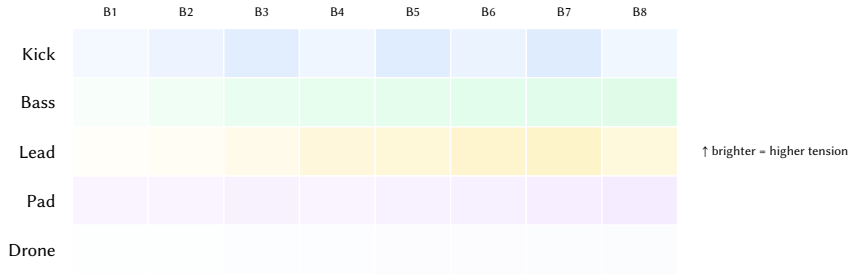


Figure 138. Tension heatmap mockup for an 8-bar build section. Each row is one voice; each column is one bar. Cell brightness encodes tension contribution.

19.2.7 Energy Landscape

A 3-D terrain mesh where x=time (bars), y=voice index, and z=effective energy value after all four levels of the energy model (section 7) have been applied, rendered via Three.js `PlaneGeometry`.

19.3 Existing Visualisations Consolidated

Several visualisations predate the unified architecture:

- **Tala circle** (section 16) — the reference implementation for the phase wheel.
- **Voice constellation** (section 13.3) — a 2-D projection of the harmonic series tree.
- **Energy curve** (section 7) — the sparkline that the energy landscape extends into 3-D.

19.4 Render Pipeline

All visualisers share a single telemetry path rooted in the DSP engine. At the end of each render block (128 samples at 44.1 kHz ≈ 2.9 ms), the engine collects per-voice metrics and sends them to the Axum broadcast channel (section 21.2.2).

The React frontend maintains a `telemetry_buffer` singleton that receives raw `StateUpdate` messages, writes FFT and energy arrays directly into pre-allocated `Float32Array` slots (zero-copy path for WebGL), and throttles React state updates to 60 fps.

19.5 Accessibility and Inclusive Design

The visualisation layer is designed around the principle that musical creativity should not be gated by sensory or motor ability.

19.5.1 Colour-Blind Safe Palettes

Every visualisation ships with four palette variants: the default *standard* palette and three colour-blind safe alternatives targeting deuteranopia, protanopia, and tritanopia.

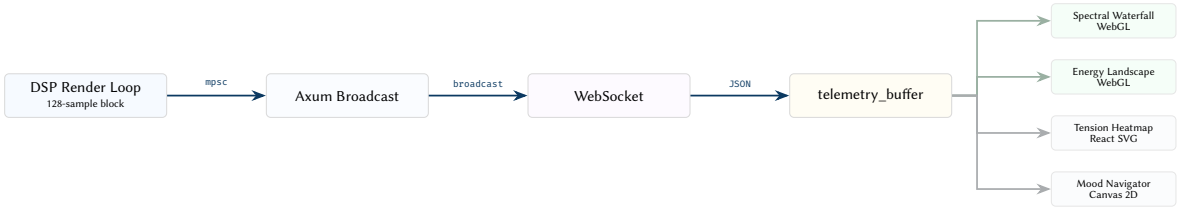


Figure 139. Visualisation render pipeline. WebGL visualisers read directly from typed-array slots (zero-copy); React-based visualisers are throttled to 60 fps.

Table 30. Standard palette tokens mapped to colour-blind safe alternatives.

Token	Role	Standard	Deuter.	Protan.	Tritan.
`layerA`	Kick	`#E05C4B`	`#0072B2`	`#0072B2`	`#D55E00`
`layerB`	Bass	`#4BAE8A`	`#E69F00`	`#E69F00`	`#009E73`
`layerC`	Melody	`#7B68EE`	`#56B4E9`	`#56B4E9`	`#CC79A7`
`layerD`	Harmony	`#F0A500`	`#F0E442`	`#F0E442`	`#56B4E9`
`layerE`	Drone	`#9DD1D1`	`#CC79A7`	`#CC79A7`	`#E69F00`

Figure 140. Standard (left) vs. deuteranopia-safe (right) colour palette.

19.5.2 Sonification of Visual Data

For screen reader users, the engine maps the three primary visualisation streams to audio parameters on a dedicated monitoring bus:

- **Tension curve → pitch contour** (110–220 Hz sine).
- **Energy level → monitoring volume** (−24 dB to −6 dB).
- **Chord function → timbre**: tonic = pure sine; subdominant adds $2f$ (−6 dB); dominant further adds $3f$ (−12 dB).

19.5.3 Reduced-Motion Mode

When `reduced_motion` is enabled: particle animations are replaced by static heatmap tiles refreshing at ≤ 4 Hz; spinning indicators become determinate progress bars; waveform scrolling is replaced by a fixed bar-level cursor; and transition cross-fades are disabled.

19.5.4 Keyboard-First Navigation

The TUI (section 16) is inherently keyboard-only; the GUI composition studio extends this discipline. Every action reachable by mouse has a keyboard binding; modal commands are grouped under single prefix keys.

19.5.5 One-Handed and Switch-Accessible Control

Three features reduce physical demand: configurable key mappings via TOML keymap files (with a `left-hand.toml` preset), sticky modifiers that latch for the next key press, and dwell-click for drag operations via the `DragHandler` trait.

19.5.6 Cognitive Load Management

Progressive disclosure partitions controls into three tiers: *Beginner* (BPM, key, scale, energy preset, themes), *Intermediate* (pattern presets, density, mute/solo, DSP macros), and *Expert* (murchhana, rasa biasing, layakari,

nadai, voicing policy). Contextual help on any widget via `?` shows description, valid range, and a cross-reference to this report.

20 UX Workflow Patterns

Overtone supports a spectrum of creative entry points. Rather than a single fixed workflow, the engine exposes six distinct *workflow modes* that share common infrastructure while offering different creative affordances.

> **Design Philosophy**
>
> Every workflow mode is a *view into the same generative engine*. State persists across mode transitions; switching modes re-contextualises the same data without discarding anything.

20.1 Top-Down Workflow: Oracle Prompt to Bar-Level Detail

The top-down workflow begins with a high-level Oracle prompt (section 21.1). The user supplies a sentence such as `"melancholic Bhairavi sunrise, 10 minutes, builds to cathartic peak at bar 64"` and the engine resolves this into a full `GeneratorConfig`. The user then progressively zooms into finer structure: chapter level, phrase palette (section 13.1), and step editor (section 16).

20.2 Bottom-Up Workflow: Jam a Loop and Grow Outward

The user starts in *Loop Focus* with a 4-bar region and iterates until satisfied. *Outward growth* is triggered by pressing **Grow**: the current loop is promoted to an anchor section, the engine proposes an energy curve around it, and a full `GeneratorConfig` is synthesised for the surrounding arrangement while the anchor's parameters are locked via `has_override` flags.

20.3 Hybrid Workflow: MIDI Import with Generative Fill

For producers with an existing sketch, the engine imports MIDI via the MIDI import pipeline, matches tracks to Overtone voices, converts detected notes to `BarPatternOverrides`, and generates all absent voices to complement the import.

20.4 Performative Workflow: Live Jam, Capture, Refine, Export

Building on Live Jam Mode (section 13.4), the performative workflow treats Overtone as a live instrument. Gestures are time-stamped into a `GestureTimeline`, captured into a deterministic render, refined in the Composition Studio, and exported.

20.5 Teaching and Learning Workflow: Teach Me Mode

Teach Me mode accompanies every generative decision with a structured *rationale annotation* displayed in a side panel. Annotations are generated at three levels:

Table 31. Teach Me annotation levels.

Level	Trigger	Example
Section	New arrangement section	"This is a *build*: energy rises from 0.4 to 0.85 over 16 bars."
Phrase	New 4-bar phrase	"Tension peaks here: the chord shifts to a tritone substitute."
Event	Individual note/hit	"This gamaka is *Meend*: a glide from Ga to Pa."

Additional features include Pause and Explain (full analytical breakdown of the current bar), counterfactual comparison (deliberate theory violation for contrast), and glossary links to section 2.

20.6 Iterative Refinement: A/B Comparison, Undo Tree, and Snapshots

A/B Comparison. Two `GeneratorConfig` snapshots can be loaded simultaneously and toggled with a keyboard shortcut. The comparison view highlights differing fields.

Undo Tree. A branching undo tree of `GeneratorConfig` snapshots is persisted in the session JSON. Branching occurs when the user edits from a non-leaf node.

Version Snapshots. Named version snapshots (e.g. `v2-dark-peak`) can be tagged at any node and shared as standalone preset files.

20.7 Workflow State Machine

The six workflow modes form a directed graph with lossless transitions.

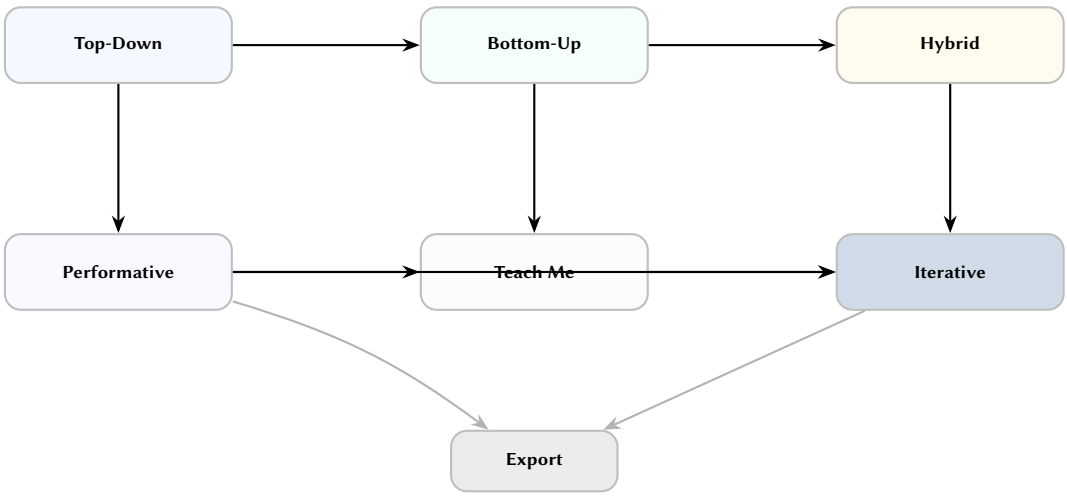


Figure 141. Workflow state machine. All modes funnel into the shared Export stage.

21 Future Directions

This section surveys planned extensions across seven areas: the narrative arc builder for multi-track set generation, a web-based visualisation interface, integration with live performance ecosystems, expansion into global ethnomusicological traditions, genre DNA and production archetypes (section 21.5), composition archetypes (section 21.6), multi-track set building UX (section 21.7), and a comprehensive export and integration ecosystem (section 21.8).

21.1 Narrative Arc Builder

While Overtone's current architecture excels at generating single-track arrangements based on static configurations, the ultimate goal is to evolve the engine into a macro-level *Set Experience Generator*. This involves abstracting the generative process from a single `GeneratorConfig` to a continuous `Timeline` populated by thematic `Chapters`.

21.1.1 The Deck and Tarot Metaphor

To make the construction of a 2-hour set intuitive, the proposed Web UI (Phase 21) abandons the traditional horizontal multi-track DAW layout in favor of a "Node-Based Flow-Graph" metaphor.

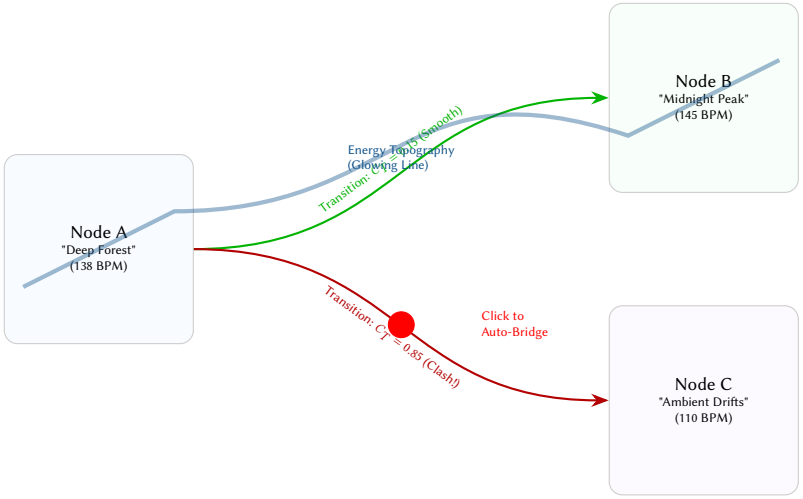


Figure 142. The Node-Based Flow-Graph metaphor. The timeline is constructed by connecting thematic nodes with algorithmic splines. Overlaid on the canvas is a glowing "Energy Topography" line that represents the global macro energy arc across the journey.

Users drag *Thematic Nodes* onto an infinite canvas. Each node renders as a unique geometric mandala pulsing at its relative BPM. Connecting two nodes with a spline creates the temporal flow.

21.1.2 Intelligent Recommendation and Transition Cost (C_T)

Overtone acts as a collaborative co-pilot, preventing jarring clashes while maintaining narrative flow. The engine features a newly implemented `TransitionEvaluator` that calculates a *Transition Cost* (C_T) between adjacent nodes as shown in fig. 143. This cost evaluates musical compatibility based on weighted components and is visually represented as an "Aura" around the connecting spline: a **Green Spline** indicates a low-cost, natural transition, while a **Red Spline** signals a jarring clash that requires structural mitigation.

- **Tempo Cost (C_{bpm}):** Calculated by comparing the absolute difference between tempos. Transitions with $|\Delta\text{BPM}| \leq 2$ yield a low cost (0.1), whereas jumps ≥ 10 BPM linearly approach a high cost (0.9), demanding a structural breakdown.
- **Cultural Cost (C_{raga}):** Modulating between parallel modes is cheap. Transitioning from a meditative midnight raga (Malkauns) to an aggressive morning raga (Bhairav) is expensive ($C_{\text{raga}} \to 1.0$), prompting the engine to suggest a bridging *Tani Avartanam* (percussion solo) to cleanse the harmonic palette.
- **Energy Continuity (C_{eng}):** Evaluates continuity between the end energy of the source and the start energy of the destination. A continuous high-energy flow (Peak → Peak) or a valid drop (Peak → Low) yields a low cost. A jarring mid-build jump (Peak → Mid-Build) is structurally awkward, raising the cost (0.8) and triggering UI warnings.

21.1.3 Auto-Bridge and Multi-Genre Thematic Fading

When a user intentionally places two high-cost cards next to each other (e.g., a 150 BPM Forest Psy chapter followed by a 170 BPM Drum & Bass chapter), the engine provides an *Auto-Bridge* feature to mathematically "solve" the transition.

$$C_T = w_1 C_{\text{bpm}} + w_2 C_{\text{raga}} + w_3 C_{\text{eng}}$$

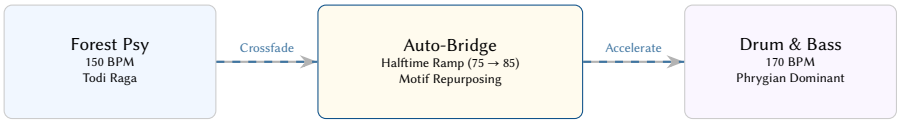


Figure 143. The Auto-Bridge mechanism and Transition Cost evaluation. The engine dynamically inserts transitional chapters based on the calculated cost C_T, which evaluates BPM differences, cultural clashes, and energy continuity.

The Auto-Bridge executes a *Multi-Genre Thematic Fade*:

1. **Rhythmic Morphing:** Dropping the beat into halftime, introducing broken-beat syncopations, and gradually accelerating.
2. **Motif Persistence:** Capturing a lead melody motif from the outgoing chapter and repurposing it as a bassline or atmospheric echo in the incoming chapter, ensuring thematic continuity.
3. **Timbral Crossfading:** Interpolating aggressive FM synthesis parameters into deep, sub-heavy textures.

21.1.4 Anchor Drop-Ins and Backward-Solving

Users are not forced to generate everything from scratch. They can place floating "Anchor Points" on the timeline—such as dropping a massive vocal sample or even importing an entire 6-minute pre-rendered track at exactly minute 45.

The Recommendation Engine treats this anchor as immutable and backward-solves the preceding chapters. If the Anchor track is 170 BPM in F-Minor, the engine calculates the transition costs backward to ensure the preceding chapter ramps seamlessly into 170 BPM. Furthermore, it will automatically *foreshadow* the anchor's motifs by playing heavily low-passed snippets of the imported audio or applying its rhythmic cadence to early hi-hat patterns.

21.1.5 Social Collaboration and The Generative Git-Tree

Because the underlying 2-hour set is simply a lightweight `Timeline` data structure, it can be version-controlled and shared instantly.

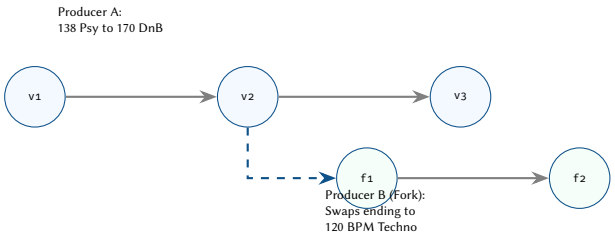


Figure 144. The Generative Git-Tree. Sets can be forked and collaboratively edited. When Producer B swaps the ending chapter, the engine automatically recalculates the transition bridges without requiring the transfer of massive audio stems.

This architecture enables asynchronous "B2B" (Back-to-Back) sets. Producer A can build the first 30 minutes and push the state. Producer B imports it, and the Recommendation Engine immediately suggests paths that logically follow Producer A's closing chapter.

21.1.6 Semantic Zoom (Macro-to-Micro UX)

Managing a continuous 2-hour timeline requires a UI that gracefully handles different levels of abstraction. The Web UI will implement *Semantic Zoom*, dynamically changing the represented data based on the current scale.

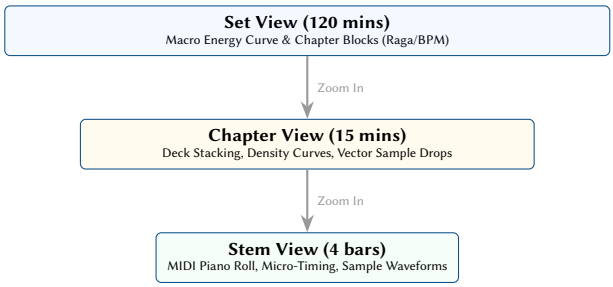


Figure 145. The Semantic Zoom interface. At the macro level, users arrange the global energetic flow. At the micro level, they edit precise MIDI syncopation. Edits made at the stem level automatically recalculate the structural tension displayed at the set level.

21.1.7 Natural Language Prompting (The Oracle)

Instead of manually dragging dozens of cards to initialize a set, users can start with an *Oracle Prompt*—a natural language interface. A prompt such as *"Build a 90-minute set that starts with deep ambient dub, transitions through heavy tribal percussion, and peaks at 150 BPM Goa trance at sunrise"* is algorithmically translated into a `Timeline` struct.

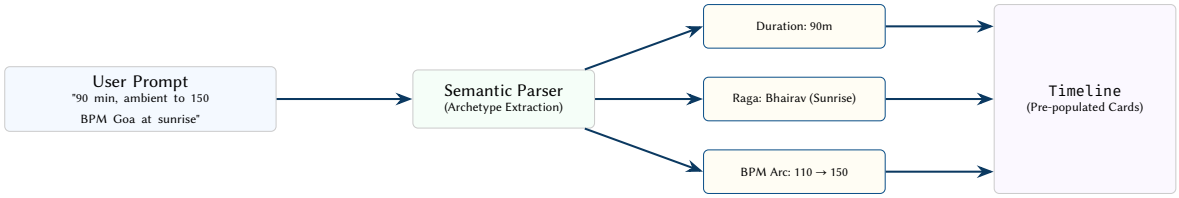


Figure 146. The Oracle Prompt workflow. Natural language is parsed into structural parameters (duration, raga mapping, tempo curves) to algorithmically pre-populate the timeline with thematic cards.

The engine automatically selects appropriate ragas (e.g., matching "sunrise" to Bhairav), calculates BPM ramps, and populates the timeline with thematic cards, providing a fully playable starting point for manual tweaking.

21.1.8 Semantic Sample Stitching (Vector Database)

In Phase 22, audio samples will evolve from static one-shots into elastic semantic objects. The engine will use `rustfft` to analyze Ableton sample packs (extracting pitch, transient markers, brightness, and semantic tags) and store them in a local Vector Database.

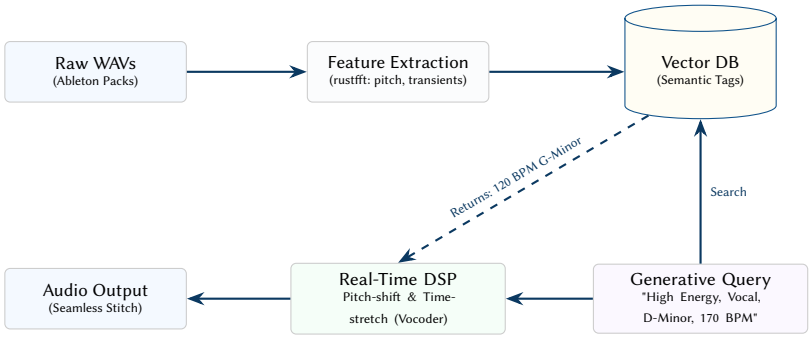


Figure 147. The Semantic Sample Stitching pipeline. During ingestion, samples are analyzed and stored in a vector database. During generation, the engine queries the DB based on the current thematic context, retrieves the best match, and dynamically time-stretches and pitch-shifts the audio to fit the active pocket.

When the engine requires a "high-energy melodic vocal motif" during a 170 BPM D-Minor chapter, it queries the database. It might retrieve a 120 BPM vocal in G-Minor, pull it into memory, and use real-time phase vocoding (or granular synthesis) to warp the sample into the D-Minor 170 BPM pocket before seamlessly rendering it into the track.

Advanced Audio Manipulation Beyond simple pitch-shifting, the engine treats these semantic objects as mutable musical primitives:

- **Transient-Locked Swing:** Because the analyzer extracts exact millisecond markers for transients, the engine can "un-quantize" and "re-quantize" static audio loops natively. A rigidly straight 16th-note techno top-loop can be sliced at its transients and delayed to lock perfectly into a sluggish Dub swing chapter.
- **Algorithmic Breakbeat Chopping:** For Drum & Bass chapters, the analyzer identifies kicks and snares within a loop based on their spectral centroids. The generative engine can then "play" the sample out of order, triggering the kick slice on beat 1 and the snare slice on beat 2.5, synthesizing original breakbeats from static loops.
- **Latent Space Motif Interpolation:** The Vector DB allows the engine to mathematically bridge two disparate samples.

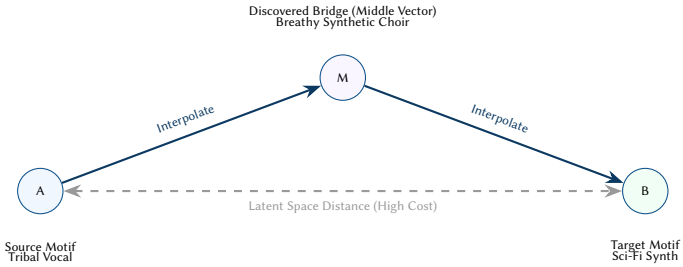


Figure 148. Latent space motif interpolation. When bridging two structurally distant chapters, the engine calculates the latent vector between their respective samples (A and B). It queries the database for a sample (M) that sits mathematically between them, creating a sonically logical bridge that a human might never have searched for.

21.1.9 Macro-Driven Micro-Timing (Deep Humanization)

The macro-narrative will directly dictate the lowest-level microscopic timing and dynamics of the DSP engine (Phase 20).

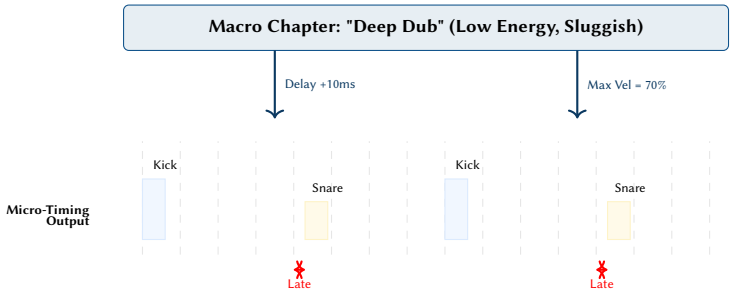


Figure 149. Macro-driven micro-timing. The global thematic state cascades down to the lowest level of the engine, algorithmically pushing notes off the rigid 16th-note grid and scaling absolute MIDI velocities to match the emotional intent of the chapter.

- **Thematic Swing:** A driving Goa chapter is locked strictly to a 0% swing 16th-note grid. As the set crossfades into a Deep Dub chapter, the engine automatically ramps up the swing and delays the snare hit by 5–10 ms, algorithmically creating a "sluggish" pocket.
- **Velocity Capping:** The engine scales the absolute MIDI strike force of the drums based on the macro-chapter. The kick drum might be hard-capped at 70% velocity during intro chapters, physically reserving the maximum 100% strike force for the exact moment the "Peak" chapter begins.

21.1.10 3D Audiovisual Representation

Because working strictly at the narrative level separates the user from the immediate "piano roll," the Web UI must provide high-fidelity sensory feedback.

Overtone will feature a real-time *Three.js/WebGL Particle Engine* directly patched into the DSP's output.

When hovering over a polyrhythmic chapter (e.g., a 3-over-4 *Khanda* syncopation), the visualizer will render rotating geometric rings interlocking perfectly. When the Semantic Sample engine pitch-shifts a bright vocal, the 3D space will flash with high-frequency particle bursts synchronized precisely to the transient markers analyzed by `rustfft`. This ensures the user can "see" the groove without reading MIDI data.

21.1.11 Environmental and Gestural Generative Seeds

To push Overtone from a studio compositional tool into a truly live performance instrument, the generative model will react to the physical environment.

The `Timeline` will not be entirely fixed. The engine will support:

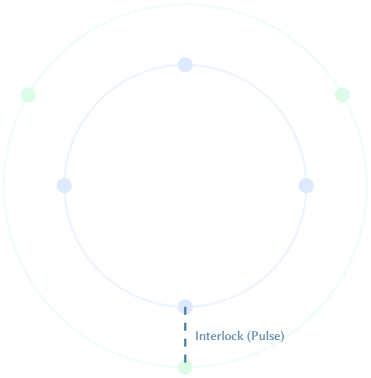


Figure 150. Geometric polyrhythm visualization. The 3D UI represents complex syncopations (e.g., 3-over-4) as interlocking rotating rings. When beats align, the particles flash, allowing the user to "see" the groove's tension and resolution without reading a MIDI grid.

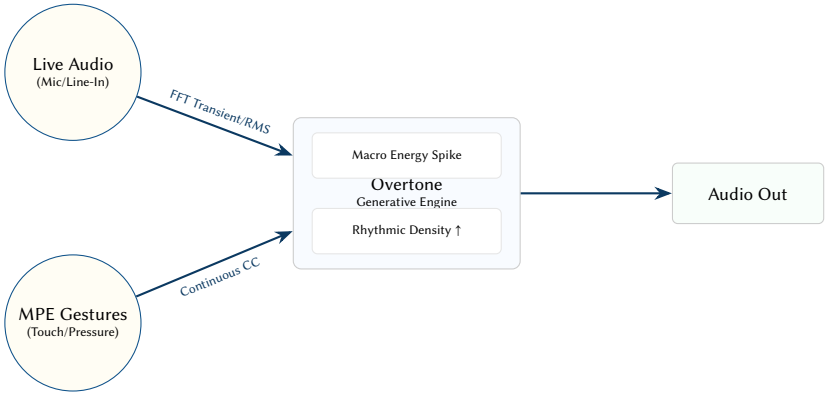


Figure 151. Environmental inputs acting as generative seeds. Instead of relying purely on pre-programmed automation curves, Overtone can analyze a live microphone feed (e.g., a crowd cheering or a live percussionist) or MPE gestural controllers to dynamically spike the internal energy model and force the algorithm into denser polyrhythms.

1. **Live Audio Analysis:** Listening to an external microphone or line-in (e.g., a live drummer playing a djembe). The engine calculates real-time energy and uses it to violently spike the `MacroCurve` or instantly increase the Humanization parameters.
2. **Gestural MIDI (MPE):** Polyphonic expression controllers (like a ROLI Seaboard) won't just map to filter cutoffs; they will map to macro-thematic variables. Pushing forcefully into a pad will dynamically force the algorithm to generate denser *tihai* polyrhythms in real-time, pulling the arrangement toward a climax solely through the performer's physical gesture.

21.2 Web UI and Visualisation Architecture

To support the Narrative Arc Builder and Semantic Zoom workflows, Overtone now has an initial decoupled client-server web stack alongside the Ratatui interface, with Phase 21 extending it into a richer long-form composition environment.

21.2.1 Headless Backend and SPA Frontend

The current implementation already includes an `axum` HTTP/WebSocket server that publishes config snapshots, engine state, Ableton diagnostics, and user intents. A React single-page frontend consumes these messages to render a node-based timeline view and browser-side control surface. A dedicated `–headless` workflow remains the natural next step for using the web UI without the terminal frontend.

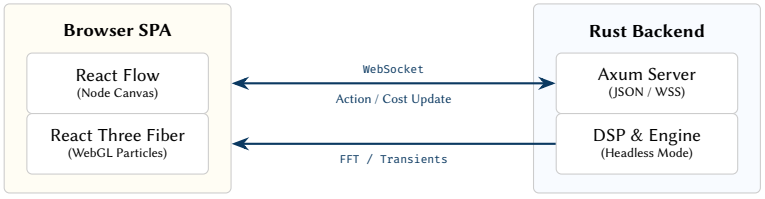


Figure 152. The Web UI Architecture. The Rust backend runs headless, exposing an Axum server. The React frontend syncs the `Timeline` state via WebSockets. The audio engine streams FFT and transient data directly to the Three.js particle system for zero-latency audio-reactive visuals.

- **React Flow:** Used to render the "Deck and Canvas" node-based timeline. It provides infinite panning/zooming for the Semantic Zoom mechanics.
- **React Three Fiber (R3F):** Drives the background WebGL particle engine. When the backend triggers a specific transient (e.g., a pitch-shifted vocal drop), the frontend intercepts the WebSocket event to generate synchronized, audio-reactive 3D bursts.
- **State telemetry:** Described in detail below.

21.2.2 WebSocket State Telemetry

The Axum server broadcasts a tagged-union `WebMessage` over the WebSocket link. Three message variants are currently defined:

- **`Config`** — a full `GeneratorConfig` snapshot, sent on initial connection and after any parameter change.
- **`StateUpdate`** — per-tick engine state containing: bar index, step index, energy level, playback flag, BPM, optional chapter metadata (darkness, euphoria, tension, groundedness, complexity), narrative log entries, Ableton connection flag, Ableton track count, and a list of missing Ableton tracks.
- **`Intent`** — a string-valued user intent (e.g. "build energy", "resolve to Sa") sent from the React UI back to the engine for the narrative oracle to interpret.

The Ableton diagnostics fields (`ableton_connected`, `ableton_track_count`, `ableton_missing_tracks`) allow the web client to display connection health and trigger auto-creation of missing Live tracks without requiring the user to interact with the TUI.

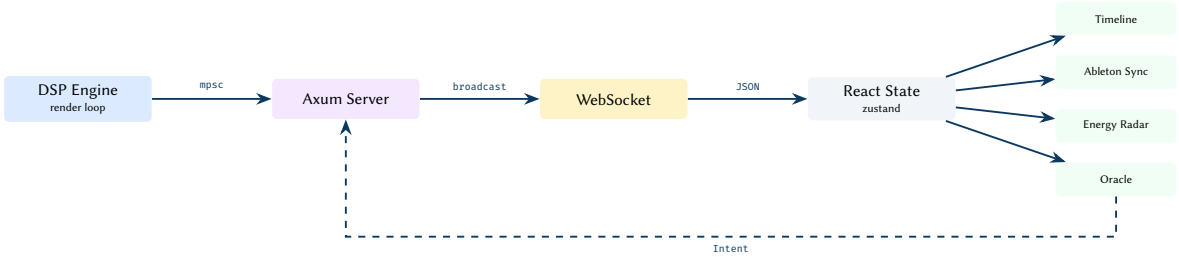


Figure 153. WebSocket telemetry flow. The DSP engine pushes `StateUpdate` messages through Axum to the React frontend, which distributes them to four UI panels. The Oracle panel sends `Intent` messages back through the same socket.

21.2.3 Live Remote Jamming (Multi-User Sync)

Because the engine state is entirely decoupled from the rendering UI, Overtone can support real-time remote collaboration. Using Conflict-free Replicated Data Types (CRDTs) like Automerge, two users in different locations can open the Web UI connected to a shared cloud relay. If User A drags a Goa Trance block to minute 30, User B sees it appear instantly and can draw an algorithmic spline connecting it to their 150 BPM Forest block. The backend resolves the timeline mathematically for both clients simultaneously.

21.3 The Live Ecosystem

Looking beyond the v2 Pro milestone, Overtone aims to integrate directly with live performance ecosystems and hardware environments.

1. **The Stage Brain (VJ & DMX Integration):** The generative timeline will emit standardized OSC commands mapped to narrative phases. This allows VJ software (e.g., Resolume) and DMX lighting bridges to automatically generate light shows that strobe precisely during rhythm morphs and shift color palettes during raga modulations.
2. **Generative Spoken Word (AI Vocals):** Integration with local Text-to-Speech (TTS) models to procedurally generate philosophical or sci-fi monologues. Overtone will analyze, time-stretch, and granularize these vocals to sit seamlessly within ambient breakdown chapters.
3. **Hardware Ecosystem Export:**
    - **Modular Bridge:** A CV/Gate output module utilizing DC-coupled audio interfaces to sequence analog Eurorack synthesizers directly from the generative timeline.
    - **Rekordbox / CDJ Export:** Overtone will export the full 2-hour set chopped into 15-minute gapless stems, formatted and analyzed for Denon Engine or Pioneer Rekordbox. It will automatically embed hot-cues at algorithmically generated transition boundaries, allowing DJs to perform the generated set on standard club hardware.

21.4 Global Ethnomusicology Expansion

While Overtone's initial architectural focus has been the fusion of Western electronic music with Indian classical theory (Hindustani and Carnatic), the platform's overarching goal is to act as a cross-cultural generative engine. Future phases will introduce deep theoretical and algorithmic models from other global traditions.

21.4.1 Arabic Maqam and 24-TET Microtonality

The Middle Eastern *Maqam* system relies on quarter-tones ("half-flats" and "half-sharps") that cannot be approximated in 12-TET.

- **Tuning Engine Upgrade:** The tuning layer will be expanded to natively support 24-TET and exact ratio-based *Just Intonation* specific to Arabic regions (e.g., $E\flat$ lowered by exactly 50 cents).
- **Maqamat Implementation:** Foundational scales such as *Bayati*, *Rast*, and *Hijaz* will be added. These are highly relevant to Psybient and "Desert Bass" subgenres.
- **Iqa'at Rhythms:** Middle Eastern rhythmic cycles (e.g., *Maqsum*, *Masmoudi*) will be added to the percussive generator, complete with *Dum* (low) and *Tak* (high) bol-to-synthesis mappings.

21.4.2 Indonesian Gamelan: Kotekan and Interlocking Rhythms

Gamelan music from Bali and Java offers a radically different approach to ensemble generation, relying heavily on *Kotekan*—rapid, interlocking parts played by multiple metallophones that combine to form a single continuous melodic line.

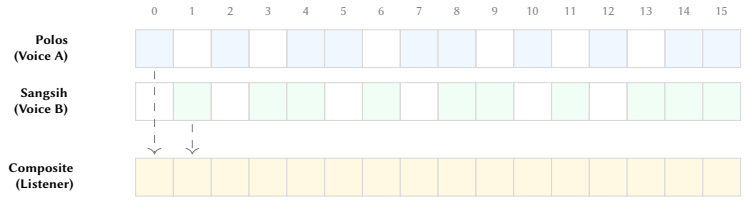


Figure 154. The Kotekan interlocking mechanism. The algorithm divides a high-speed 16th-note composite melody across two separate generative voices (*Polos* and *Sangsih*). Neither voice plays the full melody, but when panned left and right, the listener perceives a continuous, blazing-fast melodic line.

The *Pelog* (7-tone) and *Slendro* (5-tone) tuning systems will be implemented alongside a new physical modeling synthesis engine (FM-based bell and gong models). The Jugalbandi duet framework will be upgraded to support *Kotekan* generation, automatically splitting dense psytrance arpeggios across two hard-panned synth layers.

21.4.3 Japanese Aesthetics: Ma and Pentatonics

To support ultra-minimalist, ambient, and "Zen-forest" electronic tracks, the engine will integrate Japanese musical philosophy.

- **The Concept of Ma (Negative Space):** A new density heuristic that values silence as heavily as sound. In "Ma" mode, the engine intentionally leaves massive, unquantized gaps between notes, allowing long reverbs to breathe fully before the next strike.
- **Scales:** Implementation of the *In* (Sakura), *Yo*, and *Hirajoshi* pentatonic scales.
- **Synthesis Targets:** Granular processing tailored for Bamboo Flute (Shakuhachi) breath-noise textures, and a large-membrane physical model for Taiko drum impacts.

21.5 Genre DNA and Production Archetypes

Overtone's current configuration model expresses a track as a flat `GeneratorConfig` whose parameters capture energy, timbre, and harmonic intent but carry no explicit notion of *genre*. Genre knowledge is currently implicit in the twelve theme presets and in the mood metadata vectors (`darkness`, `euphoria`, `tension`, `groundedness`, `complexity`). This subsection formalises a complementary representation — the *Genre DNA vector* — that encodes the statistical signature of a production style as a compact, interpolable value type. When combined with the mood interpolation engine planned for Phase 28 (section 21), Genre DNA vectors enable principled cross-genre morphing while preserving perceptual fidelity to each tradition's production aesthetic.

21.5.1 Formalising Genre as a Parameter Vector

A *Genre DNA vector* $\mathbf{g} \in \mathbb{R}^N$ represents a genre as a point in a continuous N-dimensional production parameter space. Each coordinate captures one measurable, synthesisable attribute. The proposed eight-dimensional encoding is:

$$\mathbf{g} = \left(g_{\text{bpm}},\ g_{\text{swing}},\ g_{\text{bass}},\ g_{\text{harm}},\ g_{\text{rdens}},\ g_{\text{rvb}},\ g_{\text{fmov}},\ g_{\text{alen}}\right) \tag{30}$$

where each coordinate is normalised to $[0, 1]$ after linear mapping from its natural physical range. table 32 lists the axes, their physical units, and the mapping endpoints used for normalisation.

Table 32. Genre DNA axis definitions. All coordinates are normalised to $[0, 1]$ for interpolation; the "Natural Range" column gives the physical endpoints that map to 0 and 1.

Coordinate	Axis Name	Natural Range	Unit
g_{bpm}	BPM	60 – 200	beats/min
g_{swing}	Swing / Shuffle	0 – 33 %	% 16th offset
g_{bass}	Bass Weight	0 – 1	loudness ratio
g_{harm}	Harmonic Complexity	0 – 1	chord tone count
g_{rdens}	Rhythmic Density	0 – 1	hit density
g_{rvb}	Reverb Depth	0 – 1	wet mix
g_{fmov}	Filter Movement	0 – 1	LFO depth
g_{alen}	Arrangement Length	8 – 128	bars/section

The corresponding Rust struct mirrors this layout directly, keeping the vector in a fixed-size array to enable zero-allocation arithmetic:

```rust
/// An eight-dimensional production fingerprint for a genre archetype.
/// All fields are normalised to [0.0, 1.0] unless noted.
#[derive(Debug, Clone, Copy, PartialEq)]
pub struct GenreDna {
    pub bpm: f64,
    pub swing: f64,
    pub bass_weight: f64,
    pub harmonic_complexity: f64,
    pub rhythmic_density: f64,
    pub reverb_depth: f64,
    pub filter_movement: f64,
    pub arrangement_length: f64,
}

impl GenreDna {
    /// Linear interpolation in all eight dimensions simultaneously.
    pub fn lerp(&self, other: &GenreDna, t: f64) -> GenreDna {
        let t = t.clamp(0.0, 1.0);
        GenreDna {
            bpm:                self.bpm                + t * (other.bpm                - self.bpm
    ↪ ),
            swing:              self.swing              + t * (other.swing              - self.
    ↪ swing),
            bass_weight:        self.bass_weight        + t * (other.bass_weight        - self.
    ↪ bass_weight),
            harmonic_complexity:self.harmonic_complexity + t * (other.harmonic_complexity- self.
    ↪ harmonic_complexity),
            rhythmic_density:   self.rhythmic_density   + t * (other.rhythmic_density   - self.
    ↪ rhythmic_density),
            reverb_depth:       self.reverb_depth       + t * (other.reverb_depth       - self.
    ↪ reverb_depth),
            filter_movement:    self.filter_movement    + t * (other.filter_movement    - self.
    ↪ filter_movement),
            arrangement_length: self.arrangement_length + t * (other.arrangement_length - self.
    ↪ arrangement_length),
        }
    }

    /// Euclidean distance in the eight-dimensional production space.
    pub fn distance(&self, other: &GenreDna) -> f64 {
        let d = |a: f64, b: f64| (a - b).powi(2);
        (d(self.bpm, other.bpm)
            + d(self.swing, other.swing)
            + d(self.bass_weight, other.bass_weight)
            + d(self.harmonic_complexity, other.harmonic_complexity)
            + d(self.rhythmic_density, other.rhythmic_density)
            + d(self.reverb_depth, other.reverb_depth)
            + d(self.filter_movement, other.filter_movement)
```

```
            + d(self.arrangement_length, other.arrangement_length))
            .sqrt()
    }
}
```

The `GenreDna::lerp` method is the primitive operation consumed by the mood interpolation engine (section 21). Because all coordinates are independently bounded in $[0, 1]$, interpolated vectors are guaranteed to remain within the valid production parameter space without additional clamping.

21.5.2 Genre Profile Catalogue

table 33 enumerates the ten target genre archetypes with their characteristic DNA coordinates. The *Key Axis* column identifies the two coordinates that most strongly differentiate each genre from its neighbours in the space.

Table 33. Genre DNA profiles. BPM values are the nominal centre of each genre's typical tempo range. Swing is expressed as a percentage of the 16th-note grid offset. Remaining columns use the $[0, 1]$ normalised coordinates defined in table 32.

Genre	BPM	Sw	Bss	Hrm	Rdn	Rvb	Flt	Len	Key Axes
Acid Techno	135	0	0.80	0.20	0.85	0.25	0.90	16	g_{fmov}, g_{rden}
Deep Dub	138	0	0.90	0.25	0.60	0.50	0.55	32	g_{bass}, g_{rvb}
Hi-Tech Psytrance	148	0	0.70	0.15	0.95	0.20	0.40	16	g_{bpm}, g_{rden}
Progressive Trance	138	0	0.65	0.55	0.70	0.60	0.70	64	g_{harm}, g_{alen}
Forest Psy	150	0	0.75	0.10	0.88	0.45	0.30	16	g_{bpm}, g_{harm}
Ambient / Psybient	100	0	0.30	0.70	0.20	0.90	0.50	128	g_{rvb}, g_{alen}
Drum & Bass	174	0	0.85	0.30	0.80	0.35	0.45	8	g_{bpm}, g_{alen}
Breakbeat	125	22	0.70	0.40	0.65	0.40	0.50	16	g_{swing}, g_{harm}
Minimal Techno	130	0	0.60	0.10	0.40	0.20	0.35	32	g_{rden}, g_{fmov}
IDM	120	28	0.45	0.75	0.70	0.55	0.60	24	g_{swing}, g_{harm}

21.5.3 Genre Interpolation via the Mood Engine

The mood interpolation engine planned for Phase 28 (section 21) provides a natural host for Genre DNA blending. In that design, two `GeneratorConfig` snapshots A and B are interpolated with a scalar blend parameter $t \in [0, 1]$ across numeric fields. Genre DNA extends this by providing a *production-space prior*: rather than blending raw synthesis parameters directly, the engine first computes the interpolated DNA vector $\mathbf{g}(t) = \text{lerp}(\mathbf{g}_A, \mathbf{g}_B, t)$ and then maps $\mathbf{g}(t)$ back to synthesis parameters via a differentiable *projection function* $\phi : \mathbb{R}^8 \to$ `GeneratorConfig`.

The projection ϕ maps each DNA coordinate to one or more `GeneratorConfig` fields. Key mappings include:

- $g_{\text{bpm}} \to$ `bpm`: after de-normalisation to the range $[60, 200]$, fed directly to the tempo ramp engine.
- $g_{\text{swing}} \to$ `syncopation_style`: values below 0.08 map to `SyncopationStyle::Straight`; $[0.08, 0.20]$ to `PsyGroove`; above 0.20 to `Shuffle` or `Syncopated` depending on g_{harm}.
- $g_{\text{fmov}} \to$ `lfo_depth` and `filter_env_depth`: split equally across the two modulation sources so that filter movement feels both animated and organic.
- $g_{\text{rvb}} \to$ `reverb_wet`: mapped directly; during transitions the reverb tail is permitted to exceed the target wet level for up to 2 bars to avoid the perceptible "dry-up" artefact.

> **Design Decision 14: Genre Distance as Transition Cost**
>
> The Euclidean distance $\|\mathbf{g}_A - \mathbf{g}_B\|_2$ in production space provides a genre-aware component for the Transition Cost C_T introduced in section 21.1. A distance above 0.7 (roughly the Hi-Tech Psytrance ↔ Ambient/Psybient pair) triggers an automatic recommendation for a breakdown section before the genre morph begins.

21.5.4 Historical Production Emulation

Genre DNA vectors capture *contemporary* production norms, but each genre has a historical arc. Three eras merit specific treatment within Overtone's emulation layer.

Vintage Acid (TB-303 Emulation Constraints, 1987–1993). Early acid house and acid techno were defined by the Roland TB-303's circuit limitations: the filter is a 4-pole Sallen-Key lowpass with characteristic transistor ladder behaviour at high resonance, the sequencer is monophonic with slide and accent per step, and

the envelope follower produces the distinctive "blurp" contour absent from modern emulations. The emulation constraint set restricts `bass_osc_voices` to 1 (no detuning), forces the filter type to `FilterMode::Ladder`, caps `resonance` at 0.95, and enables the two-state accent velocity ($v \in \{64, 127\}$) rather than continuous humanisation.

Classic Goa Trance (1994–1997 FM Textures). The Goa trance sound of the mid-1990s was built almost entirely from Yamaha DX7 and TX81Z FM synthesis. Key constraints are: the lead synthesiser must use FM operator ratios drawn from DX7 algorithm 5, the pad layer is restricted to 2-operator sine-over-sine FM with a slow attack (≥ 800 ms), and the characteristic "cosmic" arpeggios use 4-note patterns cycling at dotted-8th note intervals. The `reverb_decay` target is 3.5 s with a pre-delay of 25 ms.

Early Minimal Techno (1999–2004 Sparse Percussion). The defining aesthetic of early Robert Hood and Plastikman-era minimal techno is radical subtraction. Emulation constraints cap active voices at 2 simultaneously, restrict pattern density via $g_{\text{rdens}} \leq 0.35$, and disable the sidechain duck on the bass bus. The absence of reverb ($g_{\text{rvb}} \leq 0.15$) and near-zero swing give this era its forensic, clinical texture.

> **Era Presets as DNA Constraints**
>
> Historical era presets are implemented as `GenreDna` values with additional *constraint masks* that override specific `GeneratorConfig` fields unconditionally — bypassing the continuous interpolation path. This models the fact that a TB-303 physically cannot produce two-oscillator bass: the constraint is categorical, not a matter of degree.

21.5.5 Genre DNA Radar Chart

fig. 155 visualises four representative genre profiles as a radar chart across the eight DNA axes.

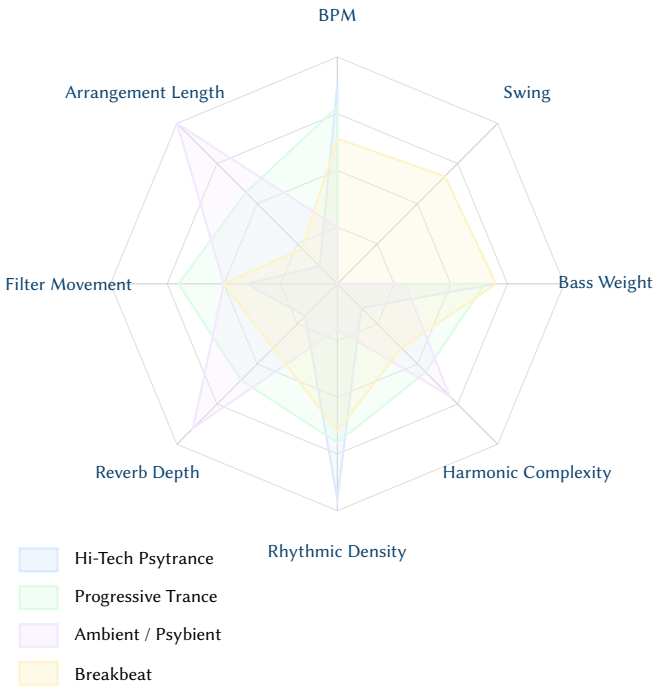


Figure 155. Genre DNA radar chart for four representative archetypes across the eight production axes defined in eq. (30). Hi-Tech Psytrance and Ambient/Psybient occupy near-antipodal positions. Breakbeat is the only profile with non-zero Swing, visible as the asymmetric spike at 45°.

21.6 Composition Archetypes and Structural Narratives

The chapter system described in section 21.1 governs *when* a timeline changes; a *composition archetype* governs *how* that change is shaped. An archetype is a high-level structural contract: a named energy-curve template that specifies the expected trajectory of tension, density, and timbral character across the full duration of a track or set segment. The planned `ArchetypeEngine` will provide a library of proven structural patterns drawn from electronic music, Western classical theory, and Indian classical performance practice, each expressed as a parameterised function over normalised time $\tau \in [0, 1]$.

Design Decision 15: Design Contract

An archetype is not a preset; it is a *shape constraint*. The parameter values inside each `Chapter` (BPM, raga, syncopation style) remain freely configurable. The archetype only prescribes the envelope of `macro_energy`, `structural_tension`, and `element_density` over the full timeline.

21.6.1 The Journey (Ambient → Peak → Denouement)

The most common psytrance and progressive electronic form: a ten-minute gradual evolution from ambient or low-energy opening through an accelerating build to a high-energy peak, then a measured denouement. The energy function is approximately sigmoidal on the ascent and exponential-decay on the descent:

$$E_{\text{journey}}(\tau) = \begin{cases} \dfrac{1}{1 + e^{-12(\tau - 0.35)}} & 0 \le \tau < 0.70 \\ e^{-6(\tau - 0.70)} & 0.70 \le \tau \le 1 \end{cases}$$

Sub-channel gating (section 7) ensures that bass, pad, and lead enter in staggered onset order. The denouement phase mirrors the alap stripping sequence: elements are muted in reverse energy-contribution order (lead first, then pad, then bass), leaving the kick and drone to carry the final bars.

21.6.2 The Drop (Tension-Release Concentrated in 4–8 Bars)

The canonical EDM structural device. The `DropArchetype` models this as a piecewise function with four regions:

1. **Pre-Drop Build** ($0 \le \tau < 0.60$): energy rises monotonically; `structural_tension` tracks energy with a +0.15 offset.
2. **Strip** ($0.60 \le \tau < 0.70$): all elements except the riser are gated off. Tension peaks at 1.0.
3. **Drop** ($\tau = 0.70$): `macro_energy` jumps discontinuously from 0 to its maximum. All elements re-enter.
4. **Sustain and Decay** ($0.70 < \tau \le 1.0$): energy plateaus then decays slowly.

21.6.3 The Meditation (Near-Static Ambient with Micro-Variation)

Drone-based composition requires a fundamentally different energy model: the macro curve is nearly flat ($[0.10, 0.30]$), yet the track must avoid stasis. The `DeterministicMacroModulator` (section 9.5) applies ultra-slow LFO movement to bass filter cutoff and reverb wet level, cycling over 32–64 bars. Raga gamaka ornaments (meend, andolan) are applied with higher intensity, since melodic surface variation compensates for rhythmic stillness.

21.6.4 Classical Form Adaptation

Sonata Form (Exposition–Development–Recapitulation). The exposition establishes the primary material in the home raga or key; the development section modulates through related ragas using the `RagaModulationEngine`; the recapitulation returns to the original material.

Rondo Form (ABACABA). The recurring A section acts as a stable anchor between contrasting episodes. A is a saved `GeneratorConfig` snapshot (section 15) that is re-instantiated at each return.

Theme and Variations. Successive chapters share the same root raga and core rhythmic pattern but each introduces one new element while holding all other parameters constant.

Table 34. Classical form archetypes and their Overtone chapter mappings.

Form	Structure	Chapter Count	Key Mechanism
Sonata	Exp–Dev–Recap	7–15	`RagaModulationEngine`
Rondo	ABACABA	7	Config snapshot restore
Theme/Var.	$T, T_1, \ldots, T_n$	3–8	`SangatEngine`
Binary	AB (repeated)	2–4	Tension offset per section
Ternary	ABA	3	Energy mirror symmetry

21.6.5 The DJ Tool (Loop-Based, Mixing-Compatible)

Tracks intended for DJ use impose structural constraints that are almost the inverse of live-performance archetypes. The DJ Tool archetype enforces extended intro and outro of at least 16 bars each (kick and bass only), a flat energy plateau (`macro_energy` held between 0.65 and 0.80), phrase alignment on 8-bar or 16-bar boundaries (`phrase_lock = true`), and explicit key declaration for harmonic mixing software.

21.6.6 The Alap-Gat Arc (Indian Classical Form as Archetype)

The raga concert structure described in section 3.27 maps naturally onto a long-form generative track:

1. **Alap** ($0 \leq \tau < 0.30$): drone and lead only, no percussion. Free-time mode with maximum gamaka intensity.
2. **Jor** ($0.30 \leq \tau < 0.55$): bass and kick enter. `LayakariEngine` begins in Vilambit (half-speed).
3. **Gat and Elaboration** ($0.55 \leq \tau < 0.85$): all elements active. Korvai, tihai, and tani avartanam are enabled. Layakari advances through Dugun and Tigun toward Chaugun.
4. **Jhala and Resolution** ($0.85 \leq \tau \leq 1.0$): jhala mode engages, energy peaks then resolves to the Sa drone, and a final tihai cadence lands on beat 1.

21.6.7 Archetype Selection and Timeline Mapping

The planned `ArchetypeEngine` maps a chosen archetype onto a `Timeline` by segmenting the total bar count into archetype regions, generating a `Chapter` for each region with the appropriate energy and tension targets, preserving user-specified parameters unchanged, and inserting transition zones based on C_T compatibility scores. Archetypes can be *nested*: a full-length set archetype can contain inner archetypes applied at the chapter level.

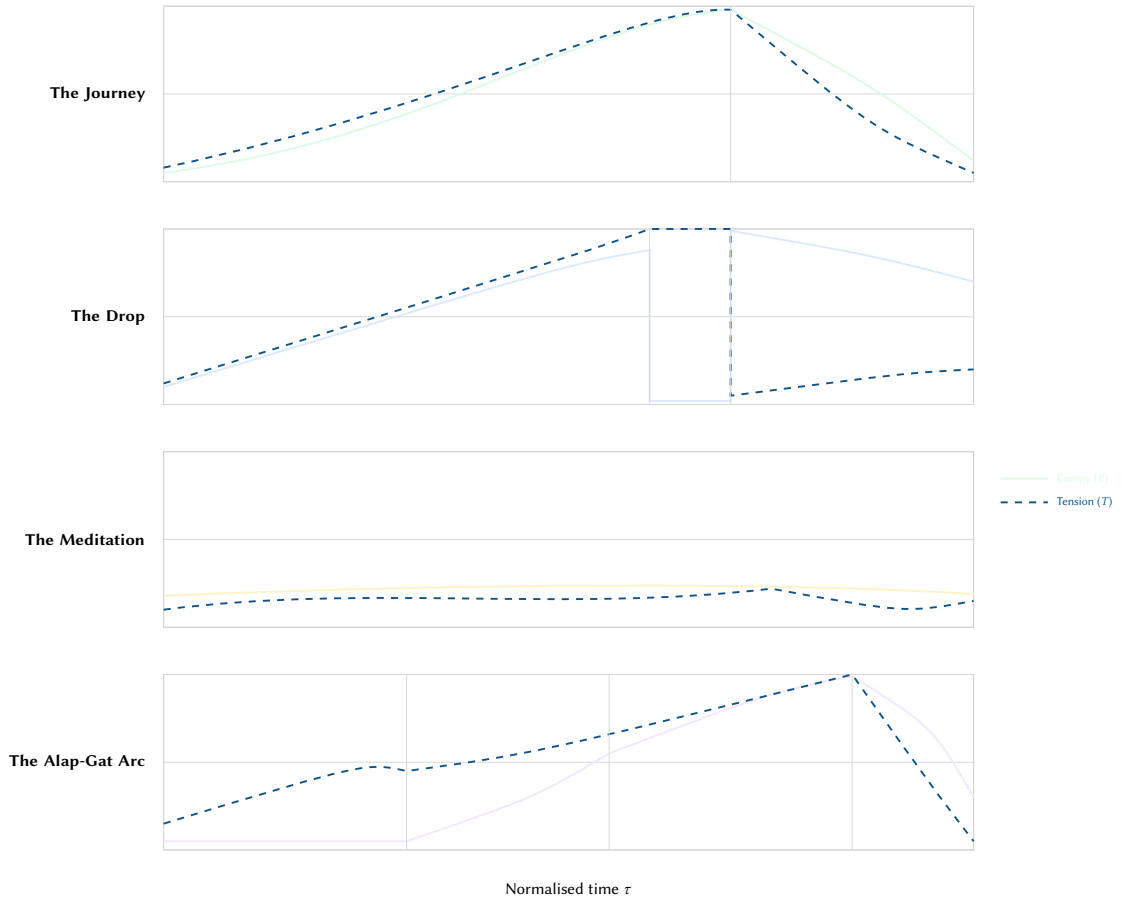


Figure 156. Normalised energy (solid) and structural tension (dashed) profiles for four composition archetypes. The Drop archetype's discontinuous energy jump at $\tau = 0.70$ is the defining structural event. The Alap-Gat Arc shows tension preceding energy during the alap phase.

21.7 Multi-Track Set Building UX

While the Narrative Arc Builder (section 21.1) addresses the *algorithmic* problem of sequencing chapters, the Set Building UX layer addresses the *ergonomic* problem: how does a performer plan, rehearse, and refine a live set interactively?

21.7.1 Set Timeline Ruler

The primary viewport is a horizontal ruler displaying two synchronised time axes: wall-clock time (`hh:mm:ss`) at 30-second tick marks and bar numbers derived from the tempo map. The engine maintains a `TempoMap` structure—a sorted vec of `(bar, bpm, wall_offset_secs)` triples—from which any bar-to-clock conversion is an $O(\log n)$ bisection.

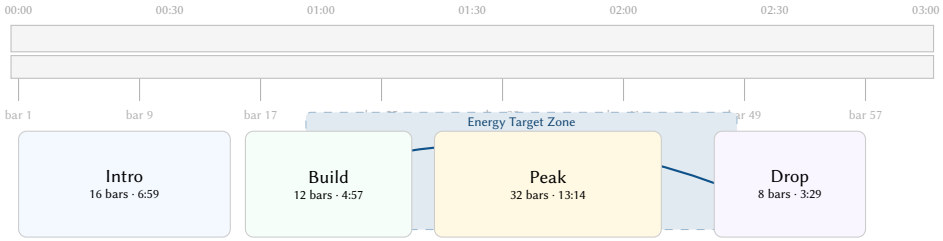


Figure 157. Set timeline ruler with dual clock/bar axes, energy target zone (shaded), and chapter blocks.

21.7.2 Energy Target Zones

The user drags a rectangular selection onto the timeline and sketches a freehand energy envelope using a Bezier pen tool. The engine receives a `ZoneFillRequest` containing the clock range and a sampled energy curve (256 scalars normalised to $[0, 1]$). The fill algorithm segments the energy curve into monotone runs, assigns BPM per chapter respecting $|\Delta\text{BPM}| \leq 8$, and instantiates chapters to fill the zone.

21.7.3 Crowd Energy Simulation

The engine includes a lightweight `CrowdModel` that tracks a scalar *crowd energy state* $\rho(t) \in [0, 1]$:

$$\rho_{n+1} = \rho_n + \alpha\big(E_n - \rho_n\big) - \beta \cdot \max(0,\ \rho_n - 0.7)^2$$

where $\alpha = 0.35$ is the engagement rate and $\beta = 0.18$ encodes the fatigue penalty above $\rho = 0.7$. The predicted ρ curve is overlaid on the timeline, giving immediate feedback on whether the planned arc will fatigue the audience before the intended climax.

21.7.4 Set Rehearsal Mode

Rehearsal mode compresses a 90-minute set to approximately 3–4 minutes by playing a 2-second representative excerpt (highest RMS) from each chapter at 4× or 8× speed via `PlaybackHandle::play_at_rate()`.

21.7.5 Setlist Templates

Table 35. Built-in setlist templates.

Template	Duration	Chapters	BPM range	Arc shape
Festival Headline	90 min	9	136–148	Low open → triple-peak → euphoric close
Club Warm-up	60 min	6	120–138	Gradual build, no hard peak, hand-off plateau
Ambient Yoga	45 min	7	72–98	Breath-in, standing peak, slow savasana decay
Late-Night Closing	75 min	8	140–150	Sustained peak → breakdown → sunrise resolve
Short DJ Set	30 min	4	130–145	Immediate energy, single peak, clean exit

21.7.6 Chapter Library

The chapter library panel exposes three tabs: **Recent** (last 32 rendered chapters), **Saved** (starred and persisted), and **Imported** (external JSON exports). Chapters are stored as `ChapterSnapshot` documents. Dragging a snapshot onto the timeline shows a compatibility badge based on C_T (section 21.1).

21.8 Export and Integration Ecosystem

The render pipeline currently produces a single stereo WAV stream and an optional MIDI file. As the `Timeline`/`Chapter` model matures, the internal representation becomes rich enough to drive a fan-out export layer targeting DJ analysis databases, modular CV bridges, video-sync formats, podcast delivery, and streaming-platform metadata.

> **Prerequisite: Chapter-Level Metadata**
>
> All export targets below depend on the `Chapter` struct carrying `bpm`, `key`, `energy_curve`, and `cue_points` as first-class fields. The current pre-render model already computes this data; the gap is serialising it into each target format.

21.8.1 Rekordbox / Engine DJ Export

Because Overtone generates audio *and* knows its own BPM grid, key, and section boundaries deterministically, it can write a fully pre-analysed database: BPM grid entries, Camelot Wheel key tags, hot cues at transition boundaries, memory cues at chapter starts, and a beatgrid offset of exactly 0.000 (no half-beat drift).

> **Design Decision 16: Deterministic Analysis**
>
> Because Overtone is the *author* of the audio, its analysis is exact by construction. A `rekordbox.xml` written from `RenderResult` metadata will never contain a shifted grid or wrong Camelot key.

21.8.2 Modular / Eurorack CV Bridge

DC-coupled audio interfaces allow sending control voltages directly to Eurorack. Planned `output/cv_bridge` will render four additional channel pairs: gate outputs per primary voice, energy CV (0–5 V unipolar from macro energy curve), tension CV from `TensionCurve` (section 12), and pitch CV (1 V/octave) for the quantised bass pitch.

21.8.3 Video Sync and VJ Integration

Two mechanisms are planned: SMPTE LTC on an auxiliary audio channel (25 fps, sample-accurate, no runtime jitter) and an OSC event bus broadcasting `/overtone/tension`, `/overtone/energy`, `/overtone/chapter`, and `/overtone/beat` to Resolume or TouchDesigner.

21.8.4 Podcast and Continuous-Mix Format

When rendering a multi-chapter `Timeline` as a single file, Overtone will write chapter marker metadata: `CHAP` frames (ID3v2.3) for MP3, `CHAPTER`*`NNN`* Vorbis comments for FLAC/Ogg, and QuickTime chapter tracks for AAC/M4A. Crossfade-safe transition points are detected where energy and tension are simultaneously at local minima.

21.8.5 Streaming Platform Metadata

- **Spotify Canvas.** Looping 3–8 s video whose motion intensity is keyed to peak-chapter macro energy.
- **Apple Music Spatial Audio.** ADM BWF file mapping Mid/Side output to a stereo bed with melody and drone as Atmos objects.
- **SoundCloud Timed Comments.** JSON batch of timed comments at each chapter boundary, annotating the waveform with section names and dominant raga.

21.8.6 Format–Metadata Support Matrix

Table 36. Export format vs. metadata capability matrix. ✓ = native support; † = non-standard extension; — = no support.

Format	BPM	Key	Cue pts	Chapters	Spatial	CV/Gate
WAV (BWF)	✓	✓	†	†	✓	—
FLAC	✓	✓	†	✓	—	—
MP3 (ID3v2)	✓	✓	†	✓	—	—
AAC / M4A	✓	✓	—	✓	✓	—
Ogg Vorbis	✓	✓	†	✓	—	—
MIDI	✓	✓	✓	†	—	—
ALS (Ableton)	✓	✓	✓	✓	—	—
ADM BWF	✓	✓	✓	✓	✓	—
CV audio	✓	✓	—	—	—	✓

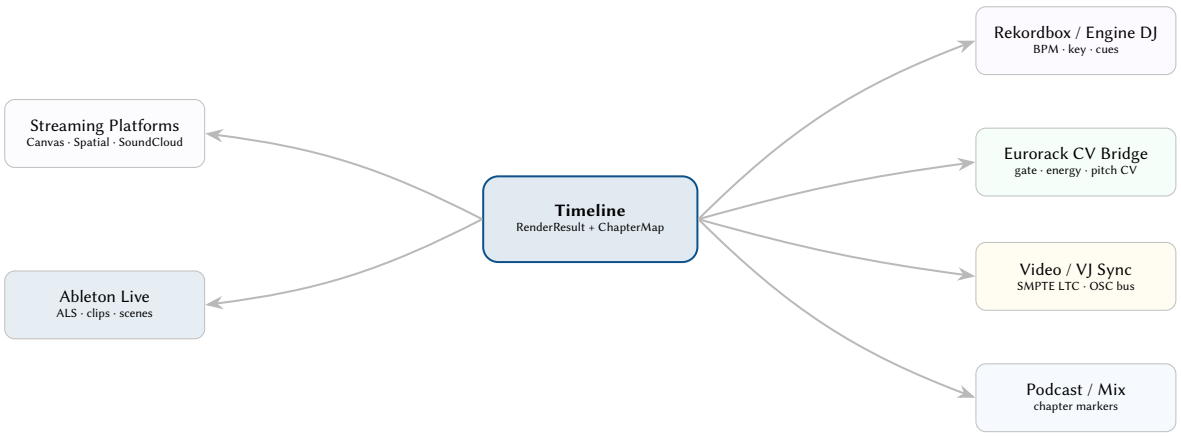


Figure 158. Export pipeline fan-out from the rendered `Timeline` to six target formats.

22 Conclusion

This report has presented Overtone, an energy-driven cross-cultural generative music engine that produces full arrangements from declarative parameter configurations. The modular architecture (section 6) separates compositional logic from audio generation, while the four-level energy model (section 7) replaces manual arrangement with a hierarchical abstraction: macro curves define the track arc, meso patterns add phrase-level breathing, micro grooves shape per-step accents, and sub-channel gating controls per-element entry and response characteristics.

Nine instruments (section 5)—each occupying a distinct spectral and rhythmic niche—are realised by dedicated synthesis engines (section 8): kick, bass, hi-hat, clap, FM lead, tanpura drone, ensemble pad, Karplus–Strong plucked strings (sitar, veena, sarod, santoor), shruti box, and tabla-articulated tom—all operating at 64-bit precision with per-sample filter modulation. The DSP layer (section 9) provides wavetable morphing with bilinear cross-cycle interpolation and PolyBLEP anti-aliased oscillators alongside biquad filters, Schroeder reverb, and ping-pong delay.

The sample intelligence subsystem (section 10) extends the sonic palette by analysing, classifying, and blending audio samples with synthesised output based on the energy state. The sample library (section 14.4) indexes samples by category and energy zone with automatic redistribution and round-robin selection.

The eastern musical traditions layer (section 3) implements the 72-melakarta Carnatic classification, the 10 Hindustani thaats, the navarasa aesthetic framework, and murchhana modal rotation for scale discovery. Six classical rhythmic devices—tihai, korvai, layakari, nadai, tani avartanam, and sangati—compose within an Alap–Jor–Gat performance arc that progressively activates synthesis elements and ornamental techniques. Raga-aware melody generation respects aroh/avroh paths and assigns per-note gamaka ornaments (andolan, kampita, meend, kan-swar).

The Ableton integration (section 17) bridges the gap between generative composition and professional production, enabling MIDI export/import, real-time TCP/JSON clip pushing via a custom Control Surface, tempo/transport synchronisation, and energy-driven scene control. Session and preset management (section 15) provides full-state persistence and reproducible sound design snapshots.

The streaming engine (section 14) provides sample-accurate, event-driven block processing as an alternative to the batch pre-render model, with energy-aware synth/sample blending and breath LFO modulation.

The pattern generation layer (section 11) maps energy to rhythmic density, syncopation, and raga-aware melody, while the terminal user interface (section 16) provides a state-machine workflow with six screens and a re-render-on-change protocol. Rendering performance (section 18) exceeds 5,000× real time for typical track lengths, enabling any parameter adjustment to produce an updated mix in under 20 ms.

The harmonic planning and tension architecture (section 12) introduces a multi-dimensional tension model that operates independently of energy—controlling chord voicing, cadential patterns, key modulation, rhythmic syncopation, register placement, pedal tones, and resolution hierarchy. The Composition Studio (section 13) provides a phrase-level GUI for manipulating musical ideas rather than individual notes, with a drag-and-drop phrase palette, context-aware theory advisor, real-time voice constellation, Live Jam Mode with gesture recording, and a five-level zoom timeline from conceptual themes down to individual MIDI events.

The visualisation architecture (section 19) defines a catalogue of seven visual modes—spectral waterfall, mood space navigator, harmonic series tree, rhythmic phase wheel, motif lineage graph, tension heatmap, and energy landscape—sharing a unified telemetry pipeline rooted in the WebSocket state stream. Six UX workflow patterns (section 20)—top-down, bottom-up, hybrid, performative, teaching, and iterative—provide

complementary creative entry points into the same generative engine, connected by a lossless state-machine transition graph. Accessibility provisions (section 19.5) ensure that colour-blind safe palettes, sonification of visual data, reduced-motion rendering, and keyboard-first navigation make the full composition workflow available regardless of sensory or motor ability.

The sound design workflow (section 13.6) introduces macro-knob parameter mapping, preset inheritance trees, A/B patch comparison, bounded randomisation, automation recording, and global timbral coherence controls. Genre DNA vectors (section 21.5) formalise production styles as interpolable eight-dimensional parameter vectors, enabling principled cross-genre morphing and historical era emulation. Composition archetypes (section 21.6) provide named structural contracts—The Journey, The Drop, The Meditation, classical form adaptations, The DJ Tool, and The Alap-Gat Arc—that prescribe energy and tension envelopes without constraining timbre.

Future directions (section 21) include a narrative arc builder for multi-track set generation, a web-based visualisation interface, integration with live performance ecosystems, expansion into global ethnomusicological traditions, multi-track set building UX with crowd energy simulation (section 21.7), and a comprehensive export and integration ecosystem (section 21.8) targeting DJ analysis databases, modular CV bridges, video sync, podcast delivery, and streaming platform metadata.

Critical Assessment

Despite these achievements, the current architecture harbours significant technical debt and rigid design biases. The reliance on 64-bit precision and per-sample coefficient updates wastes computational headroom that could be better spent on oversampling and anti-aliasing—crucial omissions that degrade the high-frequency fidelity of the FM lead and basic oscillators. Furthermore, the synthesiser implementations are heavily reliant on hardcoded parameters (e.g., fixed ADSR slopes, rigid clap multi-taps, exact 0.3s kick durations). This lack of parameterization locks the engine into its primary psytrance use case, directly conflicting with its goal of broad cross-cultural and cross-genre flexibility. Addressing these limitations through modular parameterization and proper bandlimiting is essential before the engine can be considered a truly general-purpose generative tool.

References

[1] K. Karplus and A. Strong, "Digital synthesis of plucked-string and drum timbres," *Computer Music Journal*, vol. 7, no. 2, pp. 43–55, 1983.

[2] R. Bristow-Johnson, *Audio EQ cookbook*, 2021. [Online]. Available: https://www.w3.org/2011/audio/audio-eq-cookbook.html

[3] M. R. Schroeder, "Natural sounding artificial reverberation," *Journal of the Audio Engineering Society*, vol. 10, no. 3, pp. 219–223, 1962.

[4] rustfft contributors, *RustFFT: A mixed-radix FFT library*, 2024. [Online]. Available: https://crates.io/crates/rustfft

[5] G. Toussaint, "The Euclidean algorithm generates traditional musical rhythms," in *Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science*, 2005, pp. 47–56.

[6] Ratatui contributors, *Ratatui: A Rust library for terminal user interfaces*, 2024. [Online]. Available: https://ratatui.rs/

Revision History

Version	Date	Changes
1.1.0	March 2026	Post-release update: DSP refresh, chapter-level rhythm overrides, live filter-mode control, and web/Ableton reporting improvements
1.0.0	March 2026	Initial release: 12 phases complete