# JGGL: A Social Network Where Moments Become Music

**Authors:** Andy Akwa, Dimitrii Saksonov, DZM
**Last Updated:** January 2026

## Abstract

Short-form video made sharing effortless, but creating *original* music and music videos remains complex, fragmented, and time-consuming. Users who want to turn a real-life moment—meeting friends, attending an event, traveling—into something uniquely expressive typically need multiple tools, prompt skills, and editing workflows that don't fit social timelines.

JGGL is an **AI music social network** that compresses creation into a single, natural action. A user records a short video, narrates the moment with their voice (up to ~1 minute), adds optional text and mood controls, and JGGL generates a **song and a short clip** that's ready to post. The result becomes a native social object—discoverable, shareable, and reactable through **JGGLs**.

To power this loop at global scale, JGGL introduces an AI stack purpose-built for consumer media: a language-and-planning layer that converts human intent into structured creative direction, specialized generators for music, images, and video, and a safety-and-consent system designed for real-time creation. The platform is supported by **$JGGL**, a utility token used for subscriptions, AI computation, creator rewards, marketplaces, and governance—aligning user demand, creator monetization, and compute costs in one cohesive system.

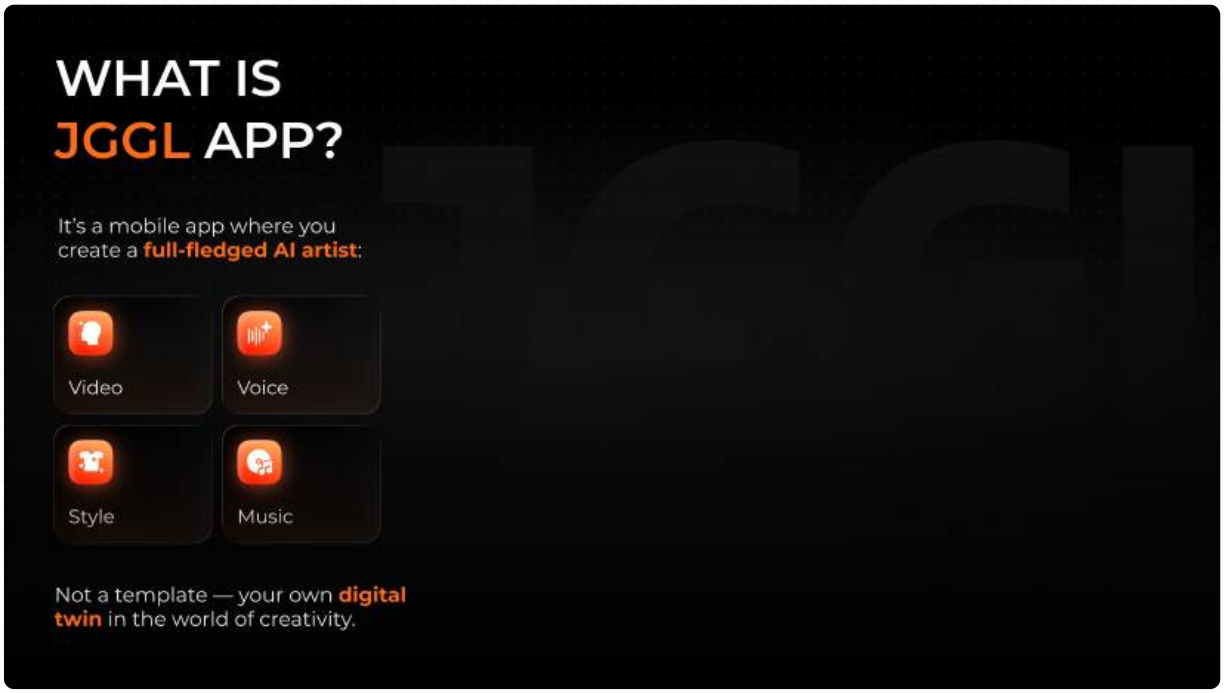
> This document is a product + technology + economy overview. It is not legal, financial, or investment advice.

---

## 1. Vision

The next wave of social platforms will be **creation-native**: instead of posting what happened, users will post what it *felt like* — as music and visuals.
JGGL turns moments into shareable media with near-zero creative friction.

## 2. The Product




### 2.1 What users do

- Capture a moment (short video)
- Explain the context by voice (up to ~60 seconds)
- Add text notes (optional)
- Adjust mood sliders (energy / genre / tempo / vibe)
- Generate and publish

### 2.2 What users publish

- **Track Post**: an original song generated from the moment
- **Clip Post**: track + short video (generated visuals, beat-synced edit, or both)
- **Remix Post** (optional roadmap): derivative content built from an existing post with attribution rules

### 2.3 Social primitives

- **JGGLs**: off-chain reactions (like a “heart”, but culturally unique)
- Comments, reposts, saves
- Creator profiles, discovery surfaces, trending

---

## 3. User Journey (End-to-End)


1. **Capture**
   User records a short video in the moment (friends, event, travel, etc.)

2. **Narrate**
   User speaks what happened and what it meant (natural language, not “prompt engineering”).

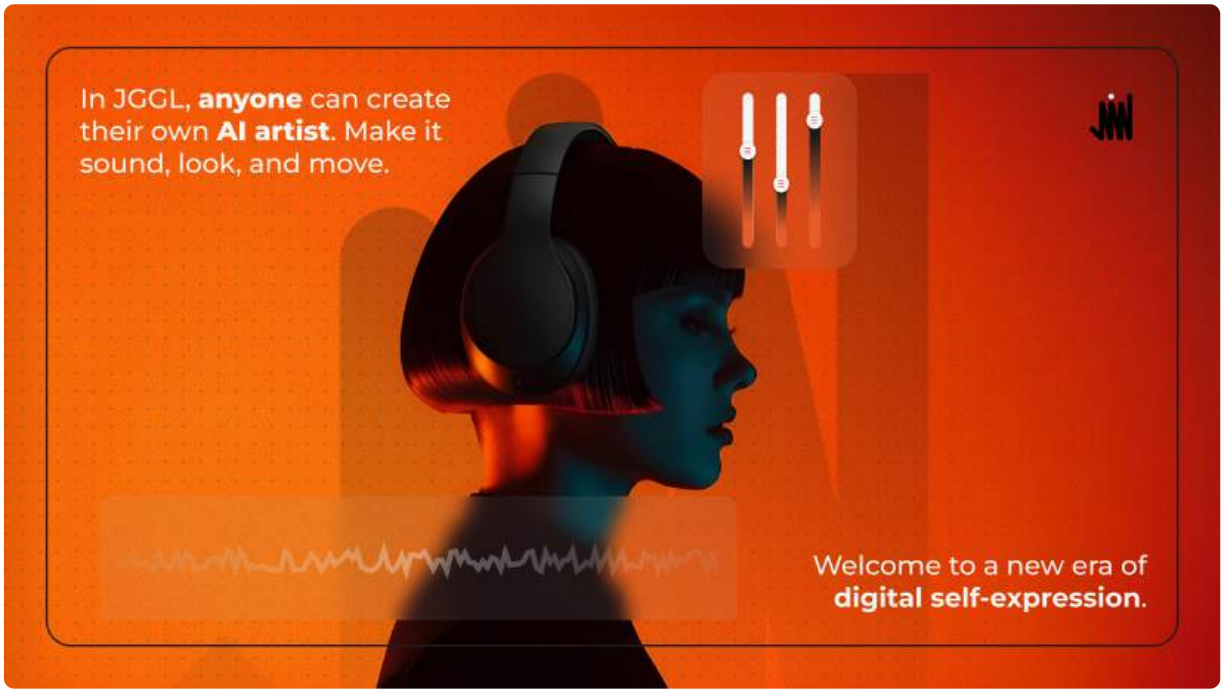
3. **Direct**
   User adds a few keywords and mood sliders. The platform converts this into a structured “creative spec”.

4. **Generate**
   Audio (song) + optional voice layer + cover art + optional short video.

5. **Publish & Interact**
   The generated content is posted to the feed. Audience reacts with (JGGLs) and shares.

---

## 4. AI Stack Overview




JGGL uses four internal model families. The names below refer to **JGGL modules** (not external models).

### 4.1 Thea AI — Image Generation

**Role:** Generate cover art, thumbnails, creator identity visuals, and video keyframes that visually "match" the music (mood, tempo, story beats).

#### Technology

Thea AI is built around a **text-conditioned latent diffusion** architecture optimized for fast iteration and style consistency.

#### Core components

- **Text / prompt encoder:** Converts the creative brief (and optional user text) into dense conditioning embeddings. This conditioning can include:
  - style presets (e.g., "cinematic neon", "grainy documentary"),
  - composition intents (portrait/landscape, subject placement),
  - mood descriptors (warm/cold, dreamy, aggressive),
  - brand/identity constraints (creator palette, recurring motifs).
- **Latent image model (denoiser):** A neural denoiser iteratively removes noise from a latent tensor over multiple steps, guided by text embeddings.

    - Thea uses **classifier-free guidance (CFG)**: it runs both conditioned and unconditioned predictions and combines them to control how strongly the output follows the prompt (high CFG = tighter adherence, with higher risk of artifacts).
- **VAE-style image codec:** Images are generated in **latent space** for efficiency, then decoded back into pixels. The same codec can also encode user-provided images (for edits / variations).
- **Sampler & noise schedule:** A sampler integrates the denoising process across timesteps. Different samplers trade off speed vs sharpness vs stability; Thea can select presets depending on whether the output is (a) cover art, (b) thumbnail, or © keyframe sequence.

**Control & consistency features**

- **Negative prompting / constraint list:** Thea accepts explicit “avoid” constraints to reduce common failure modes (extra limbs, mangled text, unwanted objects, etc.).
- **Structural guidance (optional):** Thea can condition on lightweight structural signals (e.g., edges, depth-like maps, pose hints, or layout grids) when consistency matters — particularly for video keyframes.
- **High-resolution refinement:** For covers and hero visuals, Thea can generate at a base resolution and then run a refinement pass for sharper details and cleaner textures.
- **Style adapters:** Lightweight adapters can specialize the base model into distinct aesthetics without retraining the full system (useful for creator identity packs and marketplace styles).

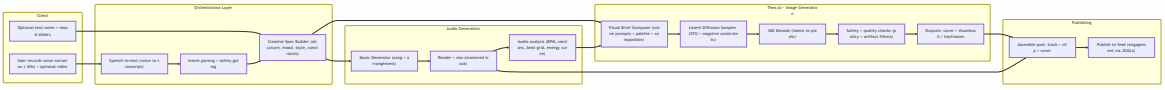
**Safety & policy**

- **Prompt sanitization:** policy rules + keyword and intent filtering before generation.
- **Output screening:** image classifiers and similarity checks post-generation to block disallowed content and prevent repeated abuse patterns.
- **Provenance hooks:** metadata tags and internal tracing IDs to support moderation, reporting, and auditability across the pipeline.

**Output formats**

1. **Cover images** (hero visuals)
2. **Template elements** (reusable identity assets)
3. **Keyframes** (beat-aligned frames used to anchor video generation)

**pipeline (voice → song → visuals)**



## 4.2 Perseia AI — Language & Creative Reasoning

**Role:** Convert human intent (voice + text + sliders) into a **machine-readable creative plan** that reliably drives downstream generation (music, visuals, video) without requiring prompt engineering.

### Technology

Perseia AI is a **transformer-based language and planning system** designed for real-time creative workflows. It functions less like a "chatbot" and more like a **creative operating layer** that compiles user intent into structured, enforceable instructions.

### Core responsibilities

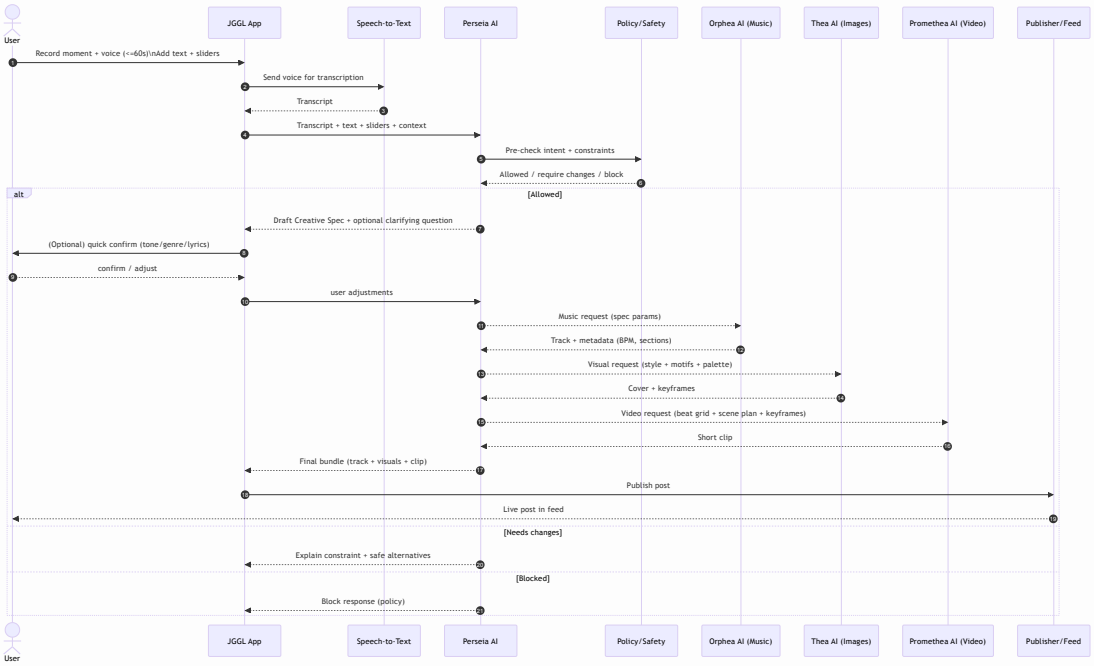
- **Intent extraction & normalization**
  - Parses voice transcript + user text + sliders into normalized signals (mood, genre direction, tempo band, intensity, narrative arc).
  - Resolves ambiguity by inferring defaults and asking minimal clarifying prompts only when needed.
- **Constraint-aware planning**
  - Produces a deterministic **Creative Spec** with fields like:
    - song structure (intro/hook/verse/bridge),
    - tempo range + energy curve,
    - instrumentation palette,
    - lyrical theme + rhyme density + vocabulary constraints (optional),
    - visual art direction (palette, motifs, composition notes),
    - video direction (scene prompts, pacing, cut points).
  - Maintains hard constraints (e.g., "no explicit lyrics", "no faces", "no violent themes").
- **Tool orchestration**
  - Calls downstream generators (music/image/video) with parameterized requests (not raw prompts).
  - Handles retries, fallbacks, and "reroll with constraints" flows (e.g., keep melody, change instrumentation).
- **Safety and policy gating**
  - Pre-generation filtering (disallowed requests, impersonation and abuse vectors).
  - Post-generation checks (sanity validation on specs; policy compliance metadata).

- **Personalization layer**
  - Builds a user preference profile (opt-in) from interaction history:
    - favored genres/tempos,
    - typical lyrical density,
    - visual style presets,
    - preferred output length and pacing.
  - Uses this to reduce friction and improve consistency over time.

### Output artifact

- A compact **Creative Spec** that is:
  - machine-readable (JSON-like structure),
  - versioned (for reproducibility),
  - auditable (for moderation and debugging),
  - portable across modules.

### how Perseia orchestrates creation



## 4.3 Orphea AI — Music Generation

**Role:** Generate complete, short-form ready songs (music + optional vocals) that feel coherent, “mix-ready,” and aligned with the user’s moment, narration, and mood controls.

### Technology (more detailed)

Orphea AI is a **neural music generation system** that composes and renders audio end-to-end while preserving musical structure (hooks, sections, dynamics) and production quality suitable for social sharing.

### Core building blocks

- **Creative Spec → Music Plan compiler**
  - Converts Perseia’s Creative Spec into a *musical blueprint*:
    - tempo range + groove,
    - section map (intro / hook / verse / bridge / outro),
    - energy curve (where intensity rises/falls),
    - instrumentation palette,
    - lyrical settings (if lyrics are enabled),
    - vocal style + timbre settings (if vocals are enabled).
- **Composition model (structure-first)**
  - Produces high-level musical decisions:
    - chord progression and harmonic rhythm,
    - melodic motifs + hook emphasis,
    - rhythmic patterns and section transitions,
    - arrangement density over time.
  - Enforces section boundaries so the output has recognizable “song form” rather than an endless loop.
- **Audio rendering model (fidelity-first)**
  - Converts the composition into high-quality audio using learned audio representations.
  - Rendering can be conditioned on:
    - genre timbre priors,
    - instrument embeddings,
    - mix and mastering targets (loudness, dynamics, clarity).
- **Vocal layer (optional)**

    - If lyrics are enabled, Orphea can generate vocals aligned to phrasing and rhythm.
    - If voice cloning is enabled, the vocal timbre can be conditioned to the user's registered voice profile (consent-based).
- **Mixing & mastering stage**
    - Loudness normalization and spectral balancing to avoid "muddy" results.
    - Peak limiting and dynamic control for consistent playback in a social feed.
    - Optional stereo widening and transient shaping based on style presets.

**Control surface**
Orphea is designed to be controllable without prompt engineering:

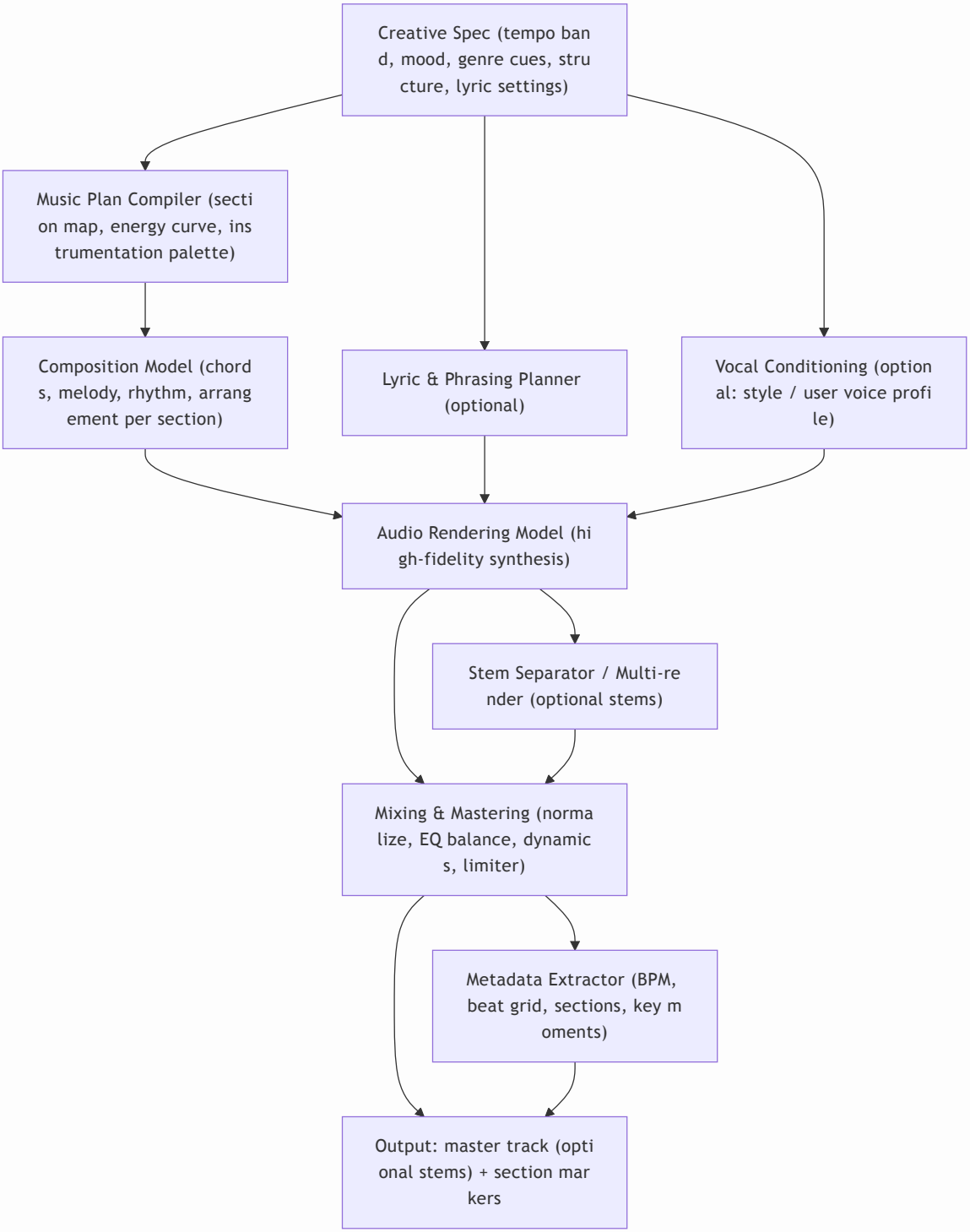
- **hard constraints** (tempo band, explicit content off, no aggressive vocals, etc.)
- **soft preferences** (more "uplifting," "more punchy drums," "less dense mix")
- **iterative edits** (keep structure, change genre; keep melody, change instrumentation; shorten intro)

**Stems / segments (optional)**
When enabled, Orphea can output:

- **stems** (vocals / drums / bass / harmony / leads),
- **segments** aligned to sections (hook / verse / bridge),
  which unlock remix workflows and future marketplace mechanics.

**Output:** A complete, mixed track ready to publish (plus optional stems and section markers).

## Low-level pipeline (spec → composition → render → master)



## 4.4 Promethea AI — Video Generation

**Role:** Generate short-form videos that are *musically synchronized* (beats, sections, energy changes) and visually consistent enough to feel "edited," not random frames.

### Technology (more detailed)

Promethea AI is a **text- and audio-conditioned latent video generator** designed for short clips. Its core challenge is not single-frame quality, but **temporal coherence**: keeping subjects, style, and motion stable across time while still producing dynamic visuals.

### Core components

- **Video brief composer**
  - Converts the Creative Spec + audio metadata into a shot plan:
    - scene prompts per section (intro/hook/verse),
    - pacing rules (cut frequency, motion intensity),
    - style constraints (palette, grain, lighting),
    - "do / don't" constraints (faces off, text avoidance, etc.).
- **Latent video model**
  - Generates in latent space for efficiency, then decodes to pixel frames.
  - Maintains temporal consistency via internal motion representations (so identity/style doesn't drift frame to frame).
- **Beat & structure alignment**
  - Uses the track's **beat grid** and **section markers** to:
    - trigger scene transitions on musical boundaries,
    - modulate motion intensity with energy curve,
    - place visual accents on kicks/snares/drops.
- **Keyframe anchoring (optional)**
  - Thea AI outputs can be used as:
    - opening frame anchors,
    - section keyframes,
    - style references to reduce drift.

### Lip Sync (optional)

When Promethea generates clips featuring performers or speaking/singing faces, it can apply a dedicated **lip-sync stage** so mouth motion matches vocals.

**How it works (conceptual):**

- **Phoneme timing extraction:** the vocal track is analyzed to estimate phoneme boundaries and timing (aligned to words/lyrics where available).
- **Face / landmark tracking:** the system detects and tracks facial landmarks across frames to preserve identity and reduce jitter.
- **Mouth-region synthesis:** a specialized model renders mouth shapes conditioned on phoneme timing while respecting:
  - head pose,
  - lighting and skin tone,
  - temporal continuity (avoid frame-to-frame “popping”).
- **Compositing:** the synthesized mouth region is blended back into frames using feathered masks and temporal smoothing.

**Why it matters:**

- makes “performer-style” clips feel intentional and human,
- improves perceived quality for vocals-heavy tracks,
- reduces uncanny artifacts that break immersion.

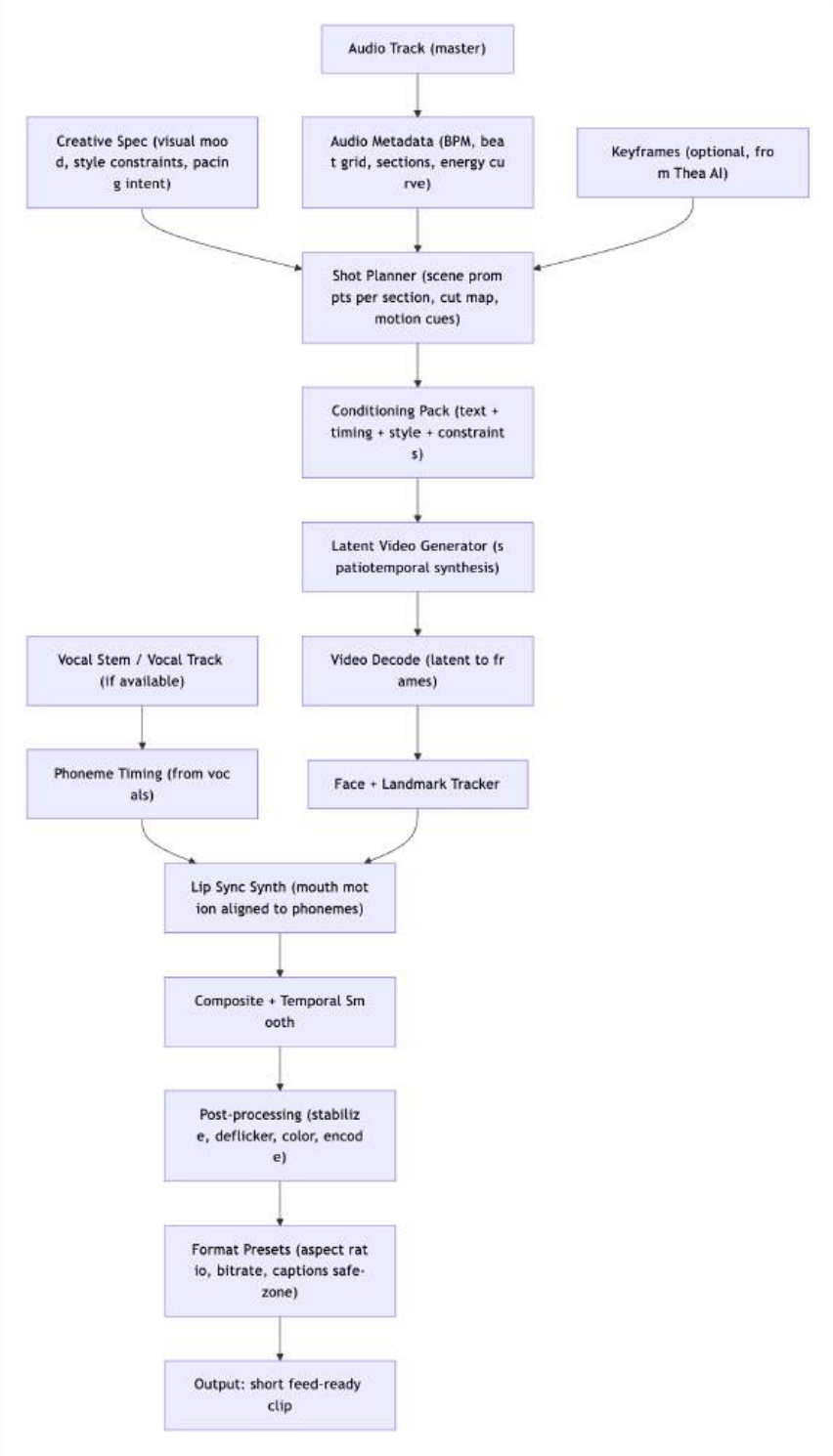
**Safety note:** lip-sync is gated behind policy rules to prevent impersonation and misuse (e.g., unauthorized identity mimicry).

**Post-processing stack**

- Temporal smoothing and stabilization (reduce flicker/shimmer).
- Optical-flow-aware interpolation (optional) to improve motion continuity.
- Lip-sync refinement smoothing (when enabled).
- Format presets: aspect ratios, safe zones, bitrate targets for feed playback.

**Output:** Short video clips optimized for scrolling feeds (consistent look, beat-matched pacing, platform-ready formatting).

## Low-level pipeline (track + spec → shot plan → latent video → lip-sync → post-process)



# 5. Voice Cloning

JGGL supports **voice cloning** as an opt-in capability that allows users to generate vocals (or spoken segments) in a voice that matches their registered voice profile. The feature is designed to be **consent-driven**, **auditable**, and **restricted** to prevent impersonation and misuse.

## 5.1 Consent

Voice cloning is enabled only after **explicit opt-in** during registration.

**What consent should cover**

- **Scope of use:** voice cloning is used only to generate content inside JGGL (e.g., vocals for songs, spoken intros/outros), not as a generic “voice export” tool.
- **Data handling:** clarify whether the platform stores a persistent voice profile/embedding or derives it from recordings; the user should understand what is retained and for what purpose.
- **Revocability:** users should be able to disable voice cloning in settings at any time. Disabling should stop future generation using the voice profile (existing published content is governed by platform policy and terms).
- **Transparency:** whenever voice cloning is applied to an output, the UI should clearly indicate it (e.g., a label on the generation screen and on the published post’s metadata panel).

**Operational controls**

- **In-product toggle:** a simple on/off control that updates eligibility instantly.
- **Re-consent on material changes:** if the feature expands materially (e.g., new use cases like third-party model marketplace usage), prompt users to re-consent.
- **Age / regional gating (if required):** the platform can restrict availability depending on legal requirements.

---

## 5.2 Safety expectations

Voice cloning has a higher abuse surface than standard music generation, so safety is enforced across policy, product design, and technical controls.

**Anti-impersonation**

- Default rule: users may only clone **their own** voice unless an explicit, verifiable authorization mechanism exists for additional voices.

- Prohibit content intended to deceive (e.g., pretending to be another person) and enforce this with both user reporting and automated detection signals.
- Strong UX cues: prevent ambiguous “celebrity” or third-party identity flows; do not provide templates that encourage impersonation.

**Provenance and labeling**

- Maintain internal provenance metadata that indicates:
  - voice cloning was used,
  - which consented profile was applied (internal ID),
  - when and how the output was generated.
- Optional outward-facing labeling for transparency in the social layer (at minimum in a metadata panel; optionally as a subtle badge).

**Watermarking**

- Support watermarking approaches that survive typical transformations (compression, trimming) and allow later verification that vocals originated from JGGL generation.
- Watermarking should be applied in a way that minimally impacts perceived audio quality.

**Rate limits and abuse monitoring**

- Apply limits on:
  - number of voice-cloned generations per time window,
  - rapid re-tries that indicate “prompt brute forcing,”
  - high-volume output patterns typical of abuse.
- Behavioral monitoring:
  - anomaly detection on new accounts,
  - device/IP reputation,
  - suspicious clustering of content themes.

**Content policy enforcement**

- Block disallowed content before generation (intent-level filters).
- Post-generation checks when feasible (keyword alignment, speech-to-text on vocal output for policy screening).
- Maintain escalation tiers: soft warnings → temporary restrictions → feature lock → account enforcement.

**Takedown and dispute process**

- Simple user-facing reporting for “impersonation” and “voice misuse.”
- Fast internal review flow for high-risk reports.
- Evidence handling:
  - preserve generation metadata for investigations,
  - provide an internal verification path using provenance/watermark signals.
- Remediation options:
  - removal of the specific post,
  - disabling voice cloning for the account,
  - account sanctions for repeat abuse.

---

## 5.3 Privacy and data minimization

**Minimize retained data**

- Prefer storing **compact voice embeddings** rather than raw recordings where possible.
- If raw audio is stored (e.g., for quality improvement), separate it from identity details and enforce strict retention policies.

**Access control**

- Voice profiles should be protected as sensitive data:
  - encryption at rest,
  - limited internal access,
  - audit logs for access and generation events.

**User control**

- Allow users to:
  - disable the feature,
  - request deletion of their voice profile (subject to platform policy and legal constraints),
  - view when the feature was last used (transparency).

---

## 5.4 Product-level transparency

To keep trust high, the experience should be explicit:

- “Voice cloning is ON/OFF” shown at creation time.
- A visible indicator on generated outputs when voice cloning was applied.
- Clear explanation (one screen) of what the feature does and does not do:

- does: generate vocals in your voice for your content
- does not: let others use your voice; export a general-purpose voice tool; enable impersonation

---

### 5.5 Future hardening

- Stronger verification for any multi-voice support.
- Third-party audits of watermark/provenance mechanisms.
- Public transparency reporting on voice-related enforcement statistics.
- Progressive rollouts with safety thresholds before expanding access.

---

## 6. Infrastructure: Today and the Decentralized Path

### 6.1 Current deployment (near-term)

- Models are hosted on JGGL-controlled infrastructure for:
  - predictable latency
  - consistent quality gating
  - unified safety enforcement
  - fast iteration cycles

### 6.2 Decentralized compute & training (mid-term vision)

JGGL plans a shift to a decentralized architecture where external compute providers can contribute GPU resources.

**Core concepts:**

- **Compute marketplace:** independent operators register hardware and stake/lock collateral to join.
- **Job routing:** generation requests are dispatched to providers with performance proofs and reputation scores.
- **Verification:** outputs are validated through lightweight checks (hash commitments, redundancy sampling, content integrity tests).
- **Economic incentives:** providers are paid per job; poor performance or policy violations reduce rewards or trigger penalties.
- **Decentralized fine-tuning:** community or creators can train specialized adapters (e.g., style packs) with auditable metadata and licensing rules.

**Why this matters:**

- elastic scaling during viral bursts
- reduced dependency on a single infrastructure footprint
- a path to creator-owned model customization (see Model Marketplace)

---

# 7. The $JGGL Token

JGGL TOKENOMICS

| TOKEN DISTRIBUTION | TYPE | PERCENTAGE | NO. OF TOKENS | OFFERING PRICE (USD) | VALUE (USD) | INITIAL TGE TOKENS | CLIFF (MONTHS) | VESTING (MONTHS) |
|---|---|---|---|---|---|---|---|---|
| Early Round | Angel/HNI investor allocation | 6.00% | 60,000,000 | $0.0750 | $4,650,000.0 | 3,100,000 | 6 | 18 |
| Private Round 2nd | Institutional investors / Funds / closed communities | 8.00% | 80,000,000 | $0.0800 | $6,240,000.0 | 3,900,000 | 6 | 18 |
| Community "Last Chance" Round | BoostyFi | 10.00% | 100,000,000 | $0.0900 | $9,000,000.0 | 5,000,000 | 6 | 18 |
| Advisors | Advisors and consultants allocation | 3.40% | 34,000,000 | — | — | 850,000 | 12 | 24 |
| Current and Future Core Contributors | Team and Developers | 13.50% | 135,000,000 | — | — | 0 | 12 | 24 |
| JGGL Operations Fund | DAO Treasury and operations | 22.50% | 225,000,000 | — | — | 22,500,000 | 6 | 48 |
| CEX+DEX Liquidity | Market Making | 10.00% | 100,000,000 | — | — | 8,333,333 | 0 | 12 |
| Liquidity Program Fund (DEX + Staking) | LP farming and staking rewards | 6.60% | 66,000,000 | — | — | 1,100,000 | 0 | 60 |
| Ecosystem Development Fund | Grants and ecosystem development support | 7.20% | 72,000,000 | — | — | 1,200,000 | 0 | 60 |
| User Growth Pool | Airdrops and marketing activities | 6.50% | 65,000,000 | — | — | 1,083,333 | 0 | 60 |
| Growth Fund | Partnerships | 6.30% | 63,000,000 | — | — | 2,520,000 | 0 | 25 |
| TOTAL | | 100.0% | 1,000,000,000 | | | 49,586,667 | | |

FREEFLOAT ON TGE % = 4.96%

$JGGL is the utility token used to power creation, monetization, and community primitives inside JGGL.

## 7.1 Utilities (official list)

1. **Subscriptions**
   Unlock premium creation tools and capabilities with $JGGL.
   **Note:** subscription spend can be implemented as a **token sink** via on-chain burning (see **7.2**).

2. **AI Computation**
   Pay-per-generation model. $JGGL functions as “digital gas” to access premium models and faster generation tiers.

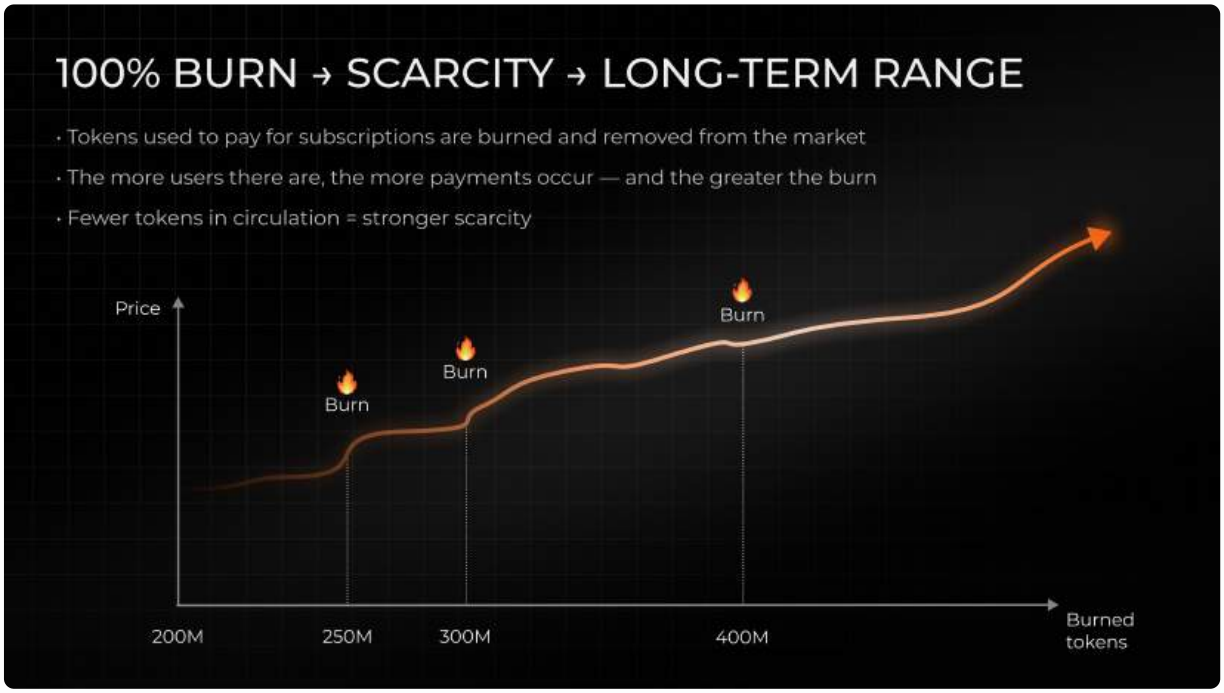
3. **Creator Rewards**
   A defined share of transactional activity is distributed to creators.

4. **Marketplace**
   Buy, sell, or license creative assets (beats, templates, visual packs, sound palettes).

5. **Model Marketplace**
   Creators can publish model adapters (e.g., style packs / fine-tunes) and earn $JGGL when used.

6. **Governance**
   Community voting on platform parameters (e.g., reward curves, marketplace policies, feature priorities).

## 7.2 Burn Mechanism (On-Chain)




At the token level, $JGGL is **burnable**: the deployed contract inherits `ERC20Burnable`, which enables permanent removal of tokens from circulation (reducing total supply).

### What “burnable” means in practice

- Any holder can burn **their own** tokens.
- A third party (e.g., a subscription contract) can burn tokens **only if** the holder has granted an allowance first ( `approve` → `burnFrom` ).
  This enables product-driven burns while remaining permissioned by the user.

### Proof in verified code (inheritance)

```
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";

contract JgglToken is ERC20, ERC20Permit, ERC20Burnable, Ownable {
    // ...
}
```

### Core burn functions (conceptual)

```
// Holder burns own tokens
function burn(uint256 amount) public {
```

```
    _burn(_msgSender(), amount);
}

// Allowance-based burn (requires prior approve)
function burnFrom(address account, uint256 amount) public {
    _spendAllowance(account, _msgSender(), amount);
    _burn(account, amount);
}
```

### Example: “subscription = burn” pattern (illustrative)

```
// Illustrative example (not deployed)
interface IERC20Burnable {
    function transferFrom(address from, address to, uint256 amount) external retu
    function burn(uint256 amount) external;
}

contract Subscription {
    IERC20Burnable public immutable jggl;

    constructor(address jgglToken) {
        jggl = IERC20Burnable(jgglToken);
    }

    function payAndBurn(uint256 amount) external {
        // User must approve(amount) beforehand
        require(jggl.transferFrom(msg.sender, address(this), amount), "transfer f
        jggl.burn(amount); // burns from this contract's balance
    }
}
```

### Verified contract code

```
https://etherscan.io/token/0x1c9b53aaca3bb34ab1039c6308671fba173adc0e#code
```

---

## 8. Creator Monetization

JGGL uses two primary reward channels:

### 8.1 Subscription Pool

A portion of subscription revenue is aggregated and distributed to creators based on measurable participation signals (examples):

- consumption (listens / watch time)
- creation impact (shares, reposts)
- follower growth and retention
- engagement quality signals (anti-bot filtered)

### 8.2 Direct Tips

Users can tip creators directly in $JGGL. Tips are explicit, creator-directed, and can be used as a signal for discovery.

### 8.3 Creator share

**70% of each transaction goes to creators** (as specified in the token utility design).

> The exact distribution policy (timing, eligibility, anti-fraud thresholds) should be published as a living “Rewards Policy” document.

---

## 9. Economic Design Principles

### 9.1 Product-first utility

$JGGL is designed to be spent on real platform actions: generation, subscriptions, assets, and creator support.

### 9.2 Sustainability of compute

AI generation has real costs. The economic layer aligns:

- user demand (generation volume)
- creator earnings
- infrastructure spend
  so the platform can scale with usage.

### 9.3 Transparent allocation

For trust, the system should keep a public policy for:

- what portion of spend funds creators
- what portion funds operations/treasury
- what portion funds compute providers (especially on the decentralized path)
- whether any portion is burned (if applicable)

---

## 10. Indicative Performance Targets (Illustrative)

These are **non-binding targets** meant to communicate expected UX standards (not guarantees).

- **Audio generation latency:** 20–60 seconds per short track (depending on quality tier)
- **Video generation latency:** 30–120 seconds for a short clip (depending on resolution/tier)
- **Median publish time:** < 3 minutes from capture → post (including retries)
- **Daily creator activation:** 20–35% of active users generating at least one post/day (goal range)
- **Cost envelope (compute):** managed by tiering (standard vs premium), batching, and caching

> Replace these with measured KPIs once production telemetry is stable.

---

## 11. Security & Integrity (What Matters Most)

- Account security and anti-sybil protections (especially for rewards)
- Output provenance tagging (generated-in-platform metadata)
- Abuse prevention for voice/video generation
- Fraud-resistant reward distribution and marketplace enforcement
- Clear reporting + takedown flow

---

# 12. Roadmap & Traction

## 12.1 Milestones to Date

**Q2–Q3 2024 (done)**

- Kickoff (team, vision, brand, channels)
- Proprietary AI R&D; internal prototypes
- App baseline assembled; content engine bootstrapped
- Promo plan; first alpha/UX tests; community seeding

**Q4 2024 – Q1 2025 (done)**

- App baseline ready; first AI artists created & released
- Closed tests (load/security); analytics baseline
- Marketplace concept & specs; development in progress
- Promo campaign live; distribution on platforms

**Q2–Q3 2025 (done)**

- New AI artists onboarded; ecosystem roles clarified
- Distribution expanded; **1,000,000 total plays/views** achieved
- Marketplace **v0.8 → v0.9** (pre-release); go-to-market preparation

---

**Q4 2025 – Q1 2026 (planned)**

- Core model upgrades and hardening across the generation stack:
  - **Orphea AI** (music): higher coherence, faster generation tiers, improved mix consistency
  - **Promethea AI** (video): stronger temporal stability, beat-sync accuracy, and optional lip-sync quality
  - **Thea AI** (image): improved style consistency and keyframe reliability for video anchoring
  - **Perseia AI** (planning): better intent-to-spec compilation and safety gating
- Production readiness:
  - latency and cost optimization
  - reliability improvements (retry logic, quality gates)
  - safety & consent enforcement refinement (voice cloning controls)
- Business and ecosystem milestones:

- **Seed round completion**
- **$JGGL token launch** and initial utility rollout (subscriptions, compute tiers, rewards primitives)

### Q2’26 — App Store Launch (Catalyst Milestone)

The roadmap anticipates a major distribution inflection point with the **native JGGL App launch** in **Q2’26** (public release via mobile app stores). This milestone is expected to materially improve:

- **Acquisition efficiency:** lower onboarding friction versus web/bot entry points.
- **Retention and frequency:** better session loops (capture → narrate → generate → publish) in a mobile-native experience.
- **Conversion into paid usage:** smoother subscription and compute-tier upsells within the core creation flow.
- **Creator economy throughput:** higher content velocity and engagement density, increasing marketplace and rewards activity.

From an ecosystem perspective, the App Store launch is positioned as the primary **growth catalyst** for userbase expansion and overall platform activity, which in turn increases **utility demand** for $JGGL across subscriptions, AI computation, marketplaces, and creator rewards.

---

## 12.2 Traction Snapshot

- Initial test version of the JGGL App launched ~**1.5 years** ago
- **JGGL Bot** and **JGGL Web** are live
- Platform metrics (as reported):
  - **$20M+** invested in JGGL
  - **11M+** users discovered JGGL
  - **110K+** users registered and generated a track
  - **46K+** active users

---

## 12.3 Forward Roadmap (High-Level)

**Phase 1 — Creation-first social**

- Stable capture → narrate → generate → publish loop at scale
- Feed ranking optimized for completion, replays, shares, and JGGLs engagement
- $JGGL subscriptions + compute tiers live (including burn-driven subscription sink)

**Phase 2 — Creator economy**

- Creator rewards via subscription pool + direct tips
- Marketplace launch (assets + promotions)
- Reward policy hardening (anti-fraud, transparent accounting)
- Model Marketplace alpha (creator-published adapters)

**Phase 3 — Decentralized scaling**

- External compute onboarding with verifiable execution and reputation
- Provider incentives + penalty mechanisms for quality and uptime
- Community-driven adapter ecosystem and governance for marketplace parameters

---

## 13. Glossary

- **Creative Spec:** Structured instructions derived from user story + sliders (tempo, mood, structure, visual style).
- **Latent Diffusion:** Image/video generation in compressed representation before decoding to pixels.
- **Conditioning:** Inputs that steer a generative model (tokens, embeddings, control signals).
- **Model Adapter:** Lightweight fine-tune component that changes style/behavior without retraining the full model.
- **JGGLs:** Off-chain reaction primitive used across the social layer.

---