Dunk Poge: Trustless NFT Ecosystem

A fully on-chain, immutable NFT collection with staking rewards
Much trustless • Very decentralized • Wow

Deployed on Ethereum Mainnet

Table of Contents

Philosophy & Design Principles

Alignment with The Trustless Manifesto

Dunk Poge embodies the principles outlined in The Trustless Manifesto:

Self-Sovereignty: Every user authorizes their own actions. No one mints, stakes, or claims on your behalf.

Verifiability: All NFT metadata lives on-chain as SVG. Anyone can verify what happened from public data.

Censorship Resistance: No admin can pause minting (except owner toggle), no one can prevent unstaking or claiming rewards.

Walkaway Test: If the team disappears, the contracts continue functioning. NFTs remain yours, staking rewards accumulate, and you can always unstake.

Accessibility: Mint price (0.002 ETH) and on-chain storage mean anyone can participate without servers or infrastructure.

Transparency of Incentives: Emission rates, decay curves, and multipliers are public constants. No hidden mechanics.

No Critical Secrets

- NFT generation uses `block.prevrandao`, `block.timestamp`, and transaction data—all public
- No private keys control user assets after deployment
- Seed generation algorithm is deterministic and verifiable

No Indispensable Intermediaries

- No relayers or off-chain services required
- Anyone can read NFT metadata directly from blockchain
- Staking contract has no admin functions after deployment
- POGE token has no special roles beyond standard ERC20

No Unverifiable Outcomes

- SVG rendering happens entirely on-chain
- Trait probabilities are encoded in contract logic
- Reward calculations are pure functions of public state
- All state changes emit events for transparency

System Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                    DUNK POGE ECOSYSTEM                          │
│                    (Ethereum Mainnet)                           │
│                                                                 │
│  ┌──────────────┐      ┌──────────────┐      ┌──────────────┐  │
│  │ DunkPoge     │      │ Pogecoin     │      │ Staking      │  │
│  │ NFT (721A)   │◄─────┤ (ERC20)      │◄─────┤ Contract     │  │
│  │              │      │              │      │              │  │
│  │ • 10K supply │      │ • 1B tokens  │      │ • Immut.     │  │
│  │ • On-chain   │      │ • No admin   │      │ • No gov     │  │
│  │ • SVG art    │      │ • Rewards    │      │ • Decay      │  │
│  └──────────────┘      └──────────────┘      └──────────────┘  │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Contract 1: DunkPoge NFT

- **Type**: ERC721A (gas-optimized batch minting)
- **Network**: Ethereum Mainnet
- **Supply**: 10,000 fixed
- **Storage**: Fully on-chain SVG generation
- **Admin**: Owner can toggle sale, airdrop, withdraw funds
- **Immutability**: Metadata generation logic is immutable

Contract 2: Pogecoin (POGE)

- **Type**: Standard ERC20
- **Network**: Ethereum Mainnet
- **Supply**: 1,000,000,000 (1 billion) fixed at deployment
- **Admin**: None post-deployment
- **Purpose**: Staking rewards token

Contract 3: DunkPogeStaking

- **Type**: NFT staking vault
- **Network**: Ethereum Mainnet
- **Admin**: None (fully immutable)
- **Rewards**: Decay from 10 POGE/day → 1 POGE/day over 730 days
- **Multipliers**: 1x → 2x over 180 days of continuous staking

---

Contract Specifications

1. DunkPoge NFT Contract

Key Parameters:

```
solidity

MAX_SUPPLY: 10,000
PRICE: 0.005 ETH
MAX_PER_WALLET: 10
ROYALTY_BPS: 500 (5%)
```

Core Functions:

`mint(uint256 quantity) payable`

- Mints 1-10 NFTs per transaction
- Requires `saleActive == true`
- Generates unique seed per token at mint time
- Stores seed for efficient metadata retrieval
- Emits `BatchMinted` event

`tokenURI(uint256 tokenId) → string`

- Returns base64-encoded JSON metadata
- Includes embedded SVG as base64 data URI
- No external dependencies (no IPFS/HTTP)
- Traits computed from stored seed

Seed Generation:

```solidity
baseSeed = keccak256(
    block.prevrandao,
    block.timestamp,
    startTokenId,
    minter,
    nonce,
    tx.gasprice
)

tokenSeed[tokenId] = keccak256(baseSeed, tokenId, index)
```

Trait System:

- 10 skin tones
- 7 eye colors
- 7 lip colors
- 18 hairstyles (80% styled, 20% none)
- 15 hair colors
- 11 eyewear styles (60% styled, 40% none)
- 7 frame colors
- 5 lens colors
- 10 headwear styles (60% styled, 40% none)
- 9 headwear colors
- 8 accessory layer 1 options (weighted)
- 4 accessory layer 2 options (85% none)
- 5 accessory layer 3 options (70% none)

Total Combinations: ~2 billion

Emergent Rarity: Actual rarity emerges from mint distribution, not predetermined weights. Attempting to game specific traits makes them more common.

---

2. Pogecoin (POGE) Contract

Deployment:

```solidity
constructor(address recipient) {
    _mint(recipient, 1_000_000_000 * 10**18);
}
```

No Special Functions: Standard ERC20 with no mint/burn/pause capabilities post-deployment.

Initial Distribution: All 1B tokens minted to deployment address, then transferred to staking contract.

---

3. DunkPogeStaking Contract

Key Parameters:

```solidity
BASE_EMISSION: 11574074074074 (~1 POGE/day per NFT)
INITIAL_BONUS: 104166666666666 (~9 POGE/day per NFT)
DECAY_PERIOD: 730 days
MAX_MULTIPLIER: 2.0x
LOYALTY_PERIOD: 180 days
MAX_TOKENS_PER_TX: 100
MAX_STAKES_PER_USER: 500
```

Core Functions:

`stake(uint256[] tokenIds)`

- Transfers NFTs to staking contract
- Records stake timestamp for each token
- Initializes `lastClaimedAt` to current time
- No approval needed after initial `setApprovalForAll`
- Emits `Staked` event per token

`unstake(uint256[] tokenIds)`

- Calculates all pending rewards
- Transfers rewards (or partial if pool insufficient)
- Returns NFTs to owner
- Deletes stake records
- Emits `Unstaked` and `RewardsClaimed` events

`claimRewards()`

- Claims rewards without unstaking
- Updates `lastClaimedAt` for all staked tokens
- Gracefully handles insufficient pool balance
- Emits `RewardsClaimed` event

`emergencyWithdraw(uint256[] tokenIds)`

- Returns NFTs immediately
- **Forfeits all pending rewards**
- Use only if contract is compromised
- Emits `EmergencyWithdraw` event

Reward Calculation:

```
solidity

// Base emission decays quadratically over 730 days
baseReward = duration × BASE_EMISSION + bonusReward

bonusReward = INITIAL_BONUS × (t1² - t2²) / (2 × DECAY_PERIOD)

// Loyalty multiplier grows linearly over 180 days
multiplier = 1.0x + (1.0x × stakeDuration / 180 days)

finalReward = baseReward × multiplier
```

Achievements System:

1. **Early Adopter**: First 30 days of staking contract
2. **Diamond Paws**: Stake for 180+ days continuously
3. **Collector**: Own 10+ NFTs with 7+ days stake duration
4. **Poge Whale**: Earn 10,000+ POGE total

Pool Health Monitoring:

- Tracks `totalRewardsDistributed` vs `totalRewardsClaimed`
- Records `totalRewardsShortfall` if pool runs low
- Gracefully pays partial rewards when balance insufficient
- Emits `RewardPoolInsufficient` events for transparency

Trustlessness Guarantees

1. No External Dependencies

NFT Metadata: Entirely on-chain SVG generation. No IPFS, no HTTP endpoints, no server dependencies.

Reward Calculation: Pure deterministic functions. No oracles, no off-chain data.

Staking Logic: No privileged roles can pause, freeze, or modify user stakes.

2. User Control

Your NFTs: Always withdrawable via `unstake()` or `emergencyWithdraw()`.

Your Rewards: Claimable at any time. Pool depletion results in partial payment, not reverts.

No Lock-ins: No minimum stake duration enforced by contract.

3. Verifiability

Trait Generation: Deterministic from on-chain seed. Anyone can verify rarity.

Reward Amounts: Anyone can call `calculateRewards(tokenId)` to verify pending rewards.

Emission Rates: Public constants in contract code.

4. Censorship Resistance

Minting: While sale is active, anyone can mint (up to wallet limit).

Staking: Anyone with NFTs can stake. No whitelist.

Claiming: No conditions prevent reward claims beyond balance availability.

Ethereum Security: As a mainnet deployment, Dunk Poge inherits Ethereum's battle-tested security, decentralization, and immutability. The contracts are as permanent and censorship-resistant as Ethereum itself.

5. Immutability Where It Matters

DunkPogeStaking: Zero admin functions. Cannot be paused, upgraded, or modified.

Pogecoin: No mint/burn after deployment. Fixed supply.

DunkPoge NFT: Metadata generation logic is immutable. Owner can only toggle sale and withdraw funds—cannot change art or traits.

User Interactions

Minting Flow

1. **Check Sale Status**: Read `saleActive` from contract
2. **Connect Wallet**: Standard web3 connection
3. **Select Quantity**: 1-10 NFTs (respects `MAX_PER_WALLET`)
4. **Approve Transaction**: Pay `0.002 ETH × quantity`
5. **Receive NFTs**: Seeds generated and stored on-chain
6. **View Metadata**: Call `tokenURI(tokenId)` to see your Dunk

Gas Optimization: ERC721A allows batch minting multiple NFTs for near-constant gas cost.

Staking Flow

1. **Approve Staking Contract**: One-time `setApprovalForAll(stakingContract, true)`
2. **Select NFTs to Stake**: Choose from owned collection
3. **Call `stake(tokenIds[])`**: NFTs transferred to staking contract
4. **Earn Rewards**: Accumulate POGE automatically
5. **View Pending**: Call `calculateRewards(tokenId)` anytime
6. **Claim or Unstake**: Get rewards with or without returning NFTs

Reward Claiming

Option A: Claim Without Unstaking

```javascript
stakingContract.claimRewards()
```

- Keeps NFTs staked, earning continues
- Resets `lastClaimedAt` to now

Option B: Claim With Unstaking

```javascript
stakingContract.unstake([tokenId1, tokenId2, ...])
```

- Returns NFTs to wallet
- Sends all pending rewards
- Stops earning on those NFTs

Graceful Degradation: If pool balance < owed rewards, contract pays what's available and emits `RewardPoolInsufficient` event. Your stake record is still cleared—this isn't a revert, it's transparent shortfall handling.

Technical Deep Dive

On-Chain SVG Generation

Why SVG?

- Resolution-independent (scale infinitely)
- Smaller than base64 image data
- Allows dynamic styling and animation
- Native browser support

Generation Process:

```solidity
1. Retrieve seed for tokenId
2. Derive traits from seed (deterministic)
3. Build CSS with trait colors
4. Assemble SVG layers (background, face, accessories, hair, etc.)
5. Base64 encode SVG
6. Wrap in JSON metadata
7. Base64 encode entire JSON
8. Return as data URI
```

Sample Output:

```json
{
  "name": "Dunk Poge #1234",
  "description": "Dunk Poge: A 10,000 Fully On-Chain, Generative NFT Collection with Emergent Rarity",
  "image": "data:image/svg+xml;base64,PHN2ZyB4bWxucz0...",
  "attributes": [
    {"trait_type": "Skin", "value": "wheat"},
    {"trait_type": "Hair", "value": "Wild Hair purple"},
    {"trait_type": "Eyewear", "value": "Classic Shades black"},
    ...
  ]
}
```

Gas Costs:

- Minting: ~150k-250k gas (varies with trait complexity)
- Metadata Read: Free (view function)
- No storage bloat: Traits computed on-demand from seed

Emission Decay Mathematics

Intuition: Start at 10 POGE/day per NFT, decay to 1 POGE/day over 2 years.

Implementation:

```solidity
// After 730 days: only base emission remains
if (globalTime >= DECAY_PERIOD) {
    return duration × BASE_EMISSION;
}

// During decay: base + quadratic bonus decay
baseReward = duration × BASE_EMISSION;

// Quadratic decay: integral of linear function
bonusReward = INITIAL_BONUS × (t1² - t2²) / (2 × DECAY_PERIOD);

return baseReward + bonusReward;
```

Why Quadratic? Linear emission decay would be a constant slope. We want the decay to slow down over time (more rewards early, gentle taper). The integral of a linear decay is a quadratic.

Example Timeline:

- Day 1: ~10 POGE/day per NFT
- Day 180: ~6.25 POGE/day per NFT (with 1.5x multiplier → 9.375 effective)
- Day 365: ~3.75 POGE/day per NFT (with 2x multiplier → 7.5 effective)
- Day 730: ~1 POGE/day per NFT (with 2x multiplier → 2 effective)

Loyalty Multiplier

Purpose: Reward long-term stakers.

Mechanics:

```solidity
function getLoyaltyMultiplier(uint256 stakedAt) returns (uint256) {
    uint256 duration = block.timestamp - stakedAt;
    if (duration >= 180 days) return 2e18; // 2.0x

    return 1e18 + ((1e18 × duration) / 180 days); // Linear growth
}
```

Effect:

- Stake 0 days: 1.0x
- Stake 90 days: 1.5x
- Stake 180+ days: 2.0x (max)

Applies to ALL pending rewards: When you unstake or claim, the multiplier is calculated at that moment and applied to the entire unclaimed period.

Supply Planning

Total POGE Supply: 1,000,000,000 (1B)

Worst Case Usage (all 10K NFTs staked with max multipliers):

- ~724M POGE distributed over 730 days
- **276M POGE buffer** (27.6% safety margin)

Realistic Case (70% participation average):

- ~400M POGE used
- **600M POGE buffer**

Shortfall Handling: Contract tracks `totalRewardsShortfall` for transparency. If pool runs low, partial payments are made and events emitted.

Security Considerations

Audited Patterns

✅**ReentrancyGuard**: All state-changing functions protected
✅**SafeERC20**: Prevents silent transfer failures
✅**ERC721A**: Battle-tested batch minting library
✅**Checks-Effects-Interactions**: NFT transfers happen after state updates

Attack Surface Minimization

No Admin in Staking: After deployment, zero functions can modify logic, pause, or seize funds.

No Upgradability: Contracts are immutable. No proxy patterns.

No External Calls (except ERC20/721 transfers): No integration with unaudited protocols.

Known Limitations

⚠️**Pool Depletion Risk**: If actual participation exceeds 70% with max multipliers, pool could run low before 730 days. Mitigated by:

- 276M POGE buffer in worst case
- Graceful degradation (partial payments, not reverts)
- Public `getRewardPoolHealth()` for monitoring

⚠️**Frontrunning Mints**: Public mempool means trait sniping is possible. Mitigated by:

- Emergent rarity (sniping makes traits common, not rare)
- Low mint price (0.002 ETH) reduces incentive
- Fast block times on Base reduce window

⚠️**Owner Powers** (NFT contract only):

- Can toggle sale (pause/unpause minting)
- Can airdrop remaining supply
- Can withdraw mint funds
- **Cannot**: Change metadata, modify traits, freeze NFTs, or affect staking

Emergency Procedures

If Staking Contract Compromised:

1. Call `emergencyWithdraw(tokenIds)` to retrieve NFTs
2. Pending rewards are forfeited (acceptable trade-off for asset recovery)
3. NFTs are immediately returned (no approval needed)

If NFT Contract Compromised:

- Metadata remains on-chain and accessible
- Secondary markets (OpenSea, etc.) can still trade
- Staking contract operates independently

Frontend Integration

Key Design Principles

Trustless Access: Frontend is a convenience layer. Users can interact directly with contracts via Etherscan if frontend is unavailable.

Progressive Enhancement: NFT simulator works without wallet connection. Ownership and staking require web3.

No Server Dependencies:

- No API calls for metadata (read from contract)
- No backend for staking status (query contract directly)
- No database for achievements (computed from on-chain events)

Architecture

```
src/
├── hooks/
│   ├── useWeb3.js          # Wallet connection
│   ├── useContracts.js     # Contract instances & state
│   └── useSelectionManager.js  # NFT selection logic
├── views/
│   ├── MintView.jsx        # Public minting interface
│   ├── StakeView/          # Staking dashboard
│   ├── RewardsView.jsx     # Claim & achievements
│   └── MythologyView.jsx   # Lore & song
└── components/
    └── NFTSimulator/       # Free preview of art
```

Contract Interactions

Read Operations (free, no gas):

```javascript
// NFT metadata
const tokenURI = await nftContract.tokenURI(tokenId);

// Staking status
const stakeInfo = await stakingContract.getStakeInfo(tokenId);

// Pending rewards
const pending = await stakingContract.calculateRewards(tokenId);

// User achievements
const achievements = await stakingContract.getUserAchievements(address);
```

Write Operations (cost gas):

```javascript
// Mint NFTs
await nftContract.mint(quantity, { value: price * quantity });

// Approve staking
await nftContract.setApprovalForAll(stakingAddress, true);

// Stake NFTs
await stakingContract.stake([tokenId1, tokenId2, ...]);

// Claim rewards
await stakingContract.claimRewards();

// Unstake NFTs
await stakingContract.unstake([tokenId1, tokenId2, ...]);
```

Brutalist UI Philosophy

Why This Aesthetic?

- **Nostalgia**: Early web, pixel art, CryptoPunks era
- **Clarity**: High contrast, bold typography, clear CTAs
- **Trustlessness**: No polish hiding complexity. What you see is what you get.

Visual Language:

- Heavy borders (`border-4`, `border-8`)
- Drop shadows (`shadow-[8px_8px_0_0_#000]`)
- Uppercase text (`uppercase`, `font-black`)
- Bright accent colors (neon green, cyan, magenta)
- Monospace fonts for data
- No gradients or smooth transitions (except where functional)

Component Examples:

```jsx
// Brutalist Button
<button className="
  bg-blue-600 text-white font-black text-xl
  py-5 border-4 border-white uppercase
  shadow-[8px_8px_0_0_#000]
  hover:-translate-x-0.5 hover:-translate-y-0.5
  hover:shadow-[10px_10px_0_0_#000]
">
  MINT NOW
</button>

// Info Card
<div className="
  bg-white text-black p-8
  border-4 border-green-400
  shadow-[12px_12px_0_0_#00ff00]
">
  <h3 className="text-3xl font-black mb-6 uppercase">
    MINT INFO
  </h3>
  {/* content */}
</div>
```

Accessibility

Despite the brutalist aesthetic, accessibility is maintained:

- Semantic HTML (`<button>`, `<nav>`, proper headings)
- ARIA labels where needed
- Keyboard navigation support
- High contrast for readability
- Clear focus states

Privacy & Data Collection

We Collect Nothing

Zero Data Collection: This project collects no user data. Period.

- **No Analytics**: No Google Analytics, Mixpanel, Amplitude, or similar
- **No Tracking Pixels**: No Facebook Pixel, Twitter conversion tracking, etc.
- **No Cookies**: No session cookies, tracking cookies, or localStorage beyond wallet connection
- **No Server Logs**: Static site deployment means no server-side logging
- **No Email Collection**: No newsletter, no accounts, no sign-ups
- **No IP Tracking**: We don't know who you are or where you're from

How This Is Possible:

The entire application is a static frontend that interacts directly with Ethereum smart contracts. There is literally no backend to send data to.

What Ethereum Knows (public blockchain):

- Your wallet address
- Your transaction history (mints, stakes, claims)
- Your NFT ownership
- Your POGE balance

What We Don't Know:

- Who you are
- Where you're located
- What browser you use
- How you found the site
- What you do on the site
- Literally anything else

Wallet Connection: When you connect your wallet (MetaMask, WalletConnect, etc.), that connection is between you and the Ethereum network. We don't see your private keys, seed phrases, or transaction signing process.

Frontend Hosting: Deployed as static files to IPFS/Arweave/decentralized hosting. No server means no server logs.

Third-Party Code:

- Web3 libraries (ethers.js, wagmi) run locally in your browser
- No external API calls except to Ethereum RPC nodes (which you can self-host)
- No CDN tracking (all dependencies bundled)

This Is A Feature, Not A Bug

Most web3 projects track extensively:

- "Wallet analytics" to understand user behavior
- Email collection for airdrops and announcements
- Session tracking to optimize conversion funnels

We don't do any of that because:

1. **Privacy is sovereignty** - Your data is yours
2. **Trustlessness extends to privacy** - You shouldn't trust us not to track you; we architecturally cannot
3. **Compliance by design** - No data = no GDPR, CCPA, or privacy law concerns
4. **Attack surface reduction** - Can't leak data we don't have
5. **Philosophical alignment** - If you don't trust intermediaries for assets, why trust them with data?

How to Verify:

- Inspect the source code (all open source)
- Check your browser's network tab (only RPC calls)
- Review the deployed static files
- Run your own frontend locally

This is what true decentralization looks like: **we don't even know you exist unless you transact on-chain**.

Conclusion

Dunk Poge is a trustless NFT ecosystem built on three immutable contracts:

1. **DunkPoge NFT**: 10,000 fully on-chain generative art pieces
2. **Pogecoin (POGE)**: 1B fixed-supply ERC20 reward token
3. **DunkPogeStaking**: Zero-admin staking with emission decay

Trustless Properties:

- No external dependencies (no IPFS, no servers, no oracles)
- No admin control over user assets or rewards
- Transparent, verifiable logic on-chain
- Graceful degradation (no surprise reverts)
- Immutable where it matters (staking, supply, metadata logic)

User Benefits:

- Own your NFTs forever (on-chain storage)
- Stake without trust (smart contract enforced)
- Claim rewards anytime (permissionless)
- Verify everything (open source, on-chain)

Cultural Remix:

- CryptoPunks aesthetic meets Doge spirit
- AI-generated mythology (the song)
- Brutalist UI (nostalgic, honest, functional)

Links

- **Contracts**: [Etherscan verification pending]
- **Manifesto**: https://trustlessness.eth.limo/general/2025/11/11/the-trustless-manifesto.html
- **Frontend**: [Deploy URL]
- **Song**: "Dunk Poge" (AI-generated, permanent on-chain)

Much documentation • Very trustless • Wow

"Life's a whole carousel-colored Ponzi scheme"
— The Song, Verse IV