

L21: NFT Technology Deep Dive

Module C: NFTs & Digital Assets

Blockchain & Cryptocurrency Course

December 2025

By the end of this lesson, you will be able to:

- Explain the difference between on-chain and off-chain NFT data
- Understand the role of token URIs in NFT metadata
- Describe the ERC-721 token standard internals
- Analyze NFT provenance and ownership verification
- Evaluate the technical limitations of current NFT implementations

What is an NFT? Technical Definition

Non-Fungible Token (NFT): A unique digital asset on a blockchain

Key Technical Properties:

- **Non-fungible:** Each token is unique and not interchangeable
- **Token ID:** Unique identifier within a smart contract
- **Ownership:** Blockchain-verified ownership record
- **Transferability:** Can be bought, sold, or transferred
- **Programmability:** Smart contract logic defines behavior

Contrast with Fungible Tokens (ERC-20):

- Fungible: 1 ETH = 1 ETH (interchangeable)
- Non-fungible: Token #1 ≠ Token #2 (unique)

ERC-721: The NFT Standard

ERC-721 introduced by Dieter Shirley (CryptoKitties) in 2017

Core Functions:

- `balanceOf(owner)` – Returns number of tokens owned
- `ownerOf(tokenId)` – Returns owner of specific token
- `transferFrom(from, to, tokenId)` – Transfers ownership
- `approve(to, tokenId)` – Grants transfer permission
- `tokenURI(tokenId)` – Returns metadata URI

Events:

- `Transfer(from, to, tokenId)` – Emitted on ownership change
- `Approval(owner, approved, tokenId)` – Emitted on approval

On-Chain vs Off-Chain Data

On-Chain Data:

- Stored directly on the blockchain
- Immutable and permanently accessible
- Expensive (gas costs for storage)
- Examples: Token ID, owner address, contract logic
- Typical size: Minimal (addresses, IDs, small integers)

Off-Chain Data:

- Stored externally (IPFS, Arweave, centralized servers)
- Referenced by on-chain URI pointer
- Cost-effective for large files (images, videos)
- Examples: Artwork, metadata JSON, high-res images
- Risk: External storage may become unavailable

Token URI: The Metadata Bridge

tokenURI Function: Links on-chain token to off-chain metadata

Example URI Patterns:

- ① **IPFS:** ipfs://QmXYZ.../metadata.json
- ② **HTTP:** https://api.project.com/token/123
- ③ **Data URI:** data:application/json;base64,...

Metadata JSON Structure:

- **name** – Token name
- **description** – Human-readable description
- **image** – URI to visual asset
- **attributes** – Array of traits (rarity properties)

Critical Issue: If metadata server goes down, NFT may become unrenderable

Rendering Pipeline:

- ① Wallet/marketplace calls `tokenURI(tokenId)`
- ② Smart contract returns URI string
- ③ Client fetches metadata JSON from URI
- ④ Parse JSON to extract `image` field
- ⑤ Fetch and display image from image URI
- ⑥ Display attributes/traits from metadata

Decentralization Spectrum:

- **Fully On-Chain:** All data in contract (rare, expensive)
- **IPFS/Arweave:** Decentralized storage (common)
- **Centralized Server:** Project-controlled API (risky)

Provenance: Tracking Ownership History

Provenance: Complete ownership history of an NFT

Blockchain Provides:

- Full transaction history via Transfer events
- Verifiable chain of custody from mint to present
- Immutable record (cannot be forged or altered)
- Creator verification (original minting address)

Why Provenance Matters:

- Proves authenticity and origin
- Increases value (celebrity ownership history)
- Detects wash trading (suspicious transfers)
- Validates artist attribution

Typical ERC-721 Contract State:

- `mapping(uint256 => address) private _owners`
 - Maps token ID to owner address
- `mapping(address => uint256) private _balances`
 - Maps owner address to token count
- `mapping(uint256 => address) private _tokenApprovals`
 - Maps token ID to approved spender
- `mapping(address => mapping(address => bool)) private _operatorApprovals`
 - Allows operators to manage all tokens for an owner

Gas Efficiency: Mappings are efficient for sparse data (not all token IDs exist)

Minting: Creating a new NFT token

Typical Minting Flow:

- ① User calls `mint()` function (often with payment)
- ② Contract generates or assigns unique token ID
- ③ Contract sets `_owners[tokenId] = msg.sender`
- ④ Contract increments `_balances[msg.sender]`
- ⑤ Contract emits `Transfer(address(0), msg.sender, tokenId)`
- ⑥ (Optional) Contract sets metadata URI

Minting Patterns:

- **Fixed Supply:** Limited collection (e.g., 10,000 tokens)
- **Open Edition:** Unlimited minting in time window
- **Lazy Minting:** Token created only when first purchased

Transfer Function Implementation:

Key Steps:

- ① Verify sender is owner or approved operator
- ② Check from address owns the token
- ③ Clear any existing approvals for the token
- ④ Update `_owners[tokenId] = to`
- ⑤ Decrement `_balances[from]`
- ⑥ Increment `_balances[to]`
- ⑦ Emit `Transfer(from, to, tokenId)`

Safety Checks:

- Cannot transfer to zero address (burning requires explicit function)
- Recipient must be able to receive NFTs (ERC-721 receiver check)

Common Extensions Beyond Base Standard:

- **ERC-721 Metadata:** Adds `name()`, `symbol()`, `tokenURI()`
- **ERC-721 Enumerable:** Allows iteration over all tokens
 - `totalSupply()` – Total number of tokens
 - `tokenByIndex(index)` – Get token ID by index
 - `tokensOfOwnerByIndex(owner, index)` – Owner's tokens
- **ERC-721 Burnable:** Allows token destruction
 - `burn(tokenId)` – Permanently destroys token
- **ERC-721 Pausable:** Emergency stop mechanism
- **ERC-721 Royalty (ERC-2981):** On-chain royalty information

ERC-1155: Supports both fungible and non-fungible tokens in one contract

Key Differences from ERC-721:

- Single contract can manage multiple token types
- Batch transfers (gas efficient for multiple tokens)
- Semi-fungible tokens (fungible until uniqueness assigned)
- Used heavily in gaming (items, currencies, NFTs)

Example Use Cases:

- Gaming: 100 fungible “gold coins” + unique weapon NFTs
- Event tickets: 500 general admission (fungible) + 10 VIP (non-fungible)

Current NFT Technology Challenges:

- **Off-chain dependency:** Most NFTs rely on external storage
- **Metadata mutability:** Some projects use mutable tokenURI
- **Gas costs:** On-chain storage extremely expensive
- **Interoperability:** Limited cross-chain NFT movement
- **Smart contract bugs:** Exploits can drain entire collections
- **Centralization risks:** Project teams control metadata servers
- **Scalability:** Ethereum mainnet slow and expensive for minting

Layer 2 Solutions: Polygon, Arbitrum, Optimism reduce costs

On-Chain Maximalism: All data stored on blockchain

Examples:

- **Autoglyphs:** Generative art stored as code in contract
- **Bitmap:** Pixel art encoded in contract storage
- **Chain Runners:** SVG generation entirely on-chain

Advantages:

- True permanence (no external dependencies)
- Maximum decentralization
- Provable scarcity of both token and artwork

Disadvantages:

- Extremely high minting costs (gas for storage)
- Limited to simple/generative art (no high-res photos)

Common NFT Smart Contract Vulnerabilities:

- **Reentrancy:** Malicious contracts exploiting callback functions
- **Integer overflow/underflow:** Arithmetic errors in token counting
- **Access control:** Unauthorized minting or burning
- **Front-running:** MEV bots exploiting mint transactions
- **Approval phishing:** Tricking users into approving malicious contracts

Best Practices:

- Use audited libraries (OpenZeppelin)
- External security audits before launch
- Reentrancy guards on state-changing functions
- Limit approval scopes (per-token vs. all tokens)

What are Ordinals?

- NFT-like artifacts directly on Bitcoin blockchain
- Uses Ordinal Theory: assigns numbers to individual satoshis
- “Inscriptions”: Data embedded in Bitcoin transaction witness data
- Enabled by Taproot upgrade (November 2021)

Key Differences from Ethereum NFTs:

- Fully on-chain (no external metadata links)
- No smart contracts (Bitcoin Script limitations)
- Higher permanence guarantees
- Higher inscription costs (\$10-100+ per inscription)

BRCT Tokens:

- Fungible tokens on Bitcoin using inscriptions
- JSON-based token standard (deploy, mint, transfer)
- \$1B+ market cap by late 2023
- Controversy: Bitcoin “bloat” vs innovation debate

Key Takeaways

- ① NFTs are blockchain tokens with unique identifiers governed by smart contracts (ERC-721/ERC-1155)
- ② Most NFT data is off-chain (images, metadata) with on-chain pointers (tokenURI)
- ③ Provenance tracking provides verifiable ownership history and authenticity
- ④ Bitcoin Ordinals (2023): Brought NFTs to Bitcoin via inscriptions
- ⑤ Technical limitations include off-chain dependencies, gas costs, and centralization risks
- ⑥ Fully on-chain NFTs (Ordinals, Art Blocks) solve permanence but sacrifice complexity/cost

Discussion Questions

- ① Should “true” NFTs require all data to be on-chain, or is off-chain storage acceptable?
- ② How does the ERC-721 standard balance gas efficiency with functionality?
- ③ What are the trade-offs between using IPFS vs. centralized servers for NFT metadata?
- ④ How does NFT provenance compare to traditional art provenance verification?
- ⑤ What innovations could solve the scalability and cost issues of on-chain NFTs?

L22: NFT Metadata and IPFS

We will explore:

- JSON metadata format standards
- IPFS content addressing and pinning
- Arweave permanent storage
- Metadata permanence and availability challenges
- Best practices for decentralized NFT storage

Preparation: Review IPFS documentation and explore NFT metadata on OpenSea