L21: NFT Technology Deep Dive

Module C: NFTs & Digital Assets

Blockchain & Cryptocurrency Course

December 2025

Learning Objectives

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 $\neq$ 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

NFT Rendering: From Contract to Display

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

ERC-721 Internals: State Variables

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 Process

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 Mechanics

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)

ERC-721 Extensions

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
  - `tokenOfOwnerByIndex(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: Multi-Token Standard

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)

Technical Limitations of NFTs

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

Fully On-Chain NFTs

On-Chain Maximalism: All data stored on blockchain

Examples:

- **Autoglyphs:** Generative art stored as code in contract
- **Blitmap:** 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)

Security Considerations

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)

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. ERC-721 uses mappings to track ownership, balances, and approvals efficiently
5. Technical limitations include off-chain dependencies, gas costs, and centralization risks
6. Fully on-chain NFTs solve permanence but sacrifice complexity and 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?

Next Lesson Preview

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