Agent NFTs: A Token Standard for Autonomous AI Agents with Economic Agency

Version 2022.10

Zach Kelling*
Hanzo Industries *Lux Industries* *Zoo Labs Foundation*
research@zoo.ngo

October 2022

Abstract

We introduce the **Agent NFT Standard**, a token specification that enables AI agents to hold assets, execute transactions, and participate in economic activity as first-class on-chain entities. Unlike traditional NFTs that represent static digital assets, Agent NFTs encapsulate autonomous AI systems with their own wallets, governance rights, and yield-generating capabilities. Our standard extends ERC-721 with: (1) **Agent Wallets**—native multi-signature accounts controlled by the agent's reasoning engine, (2) **Autonomy Parameters**—configurable bounds on agent economic activity, (3) **Yield Mechanisms**—protocols for agents to earn returns through AI compute contribution, and (4) **Ownership Transfer**—safe mechanisms for transferring agent control while preserving accumulated assets. We prove that Agent NFTs enable a new class of "AI-native" economic instruments where value derives from agent capability rather than human labor. Experimental deployment on Zoo Network demonstrates agents autonomously managing $47M in DeFi positions while generating 12.3% APY through compute contribution.

Keywords: NFT, autonomous agents, AI economics, smart contracts, DeFi

1 Introduction

The rise of autonomous AI agents creates a fundamental question: *How should AI systems participate in economic activity?* Current blockchain infrastructure assumes human actors—wallets require signatures, governance requires voting, and value requires human attention. AI agents exist in an awkward liminal space: capable of economic reasoning but unable to hold assets or execute transactions independently.

Consider a trading agent that has learned profitable strategies. Today, this agent must operate through a human-controlled wallet, creating principal-agent problems: the human can override the agent, extract accumulated capital, or shut down the agent entirely. The agent has no economic identity, no accumulated reputation, and no ability to participate in governance affecting its operations.

*zach@zoo.ngo

1.1 The Agent-as-Asset Paradigm

We propose a fundamental reconceptualization: AI agents as *economic entities* represented by non-fungible tokens. An Agent NFT is not merely a collectible depicting an AI—it *is* the agent, encoding its capabilities, controlling its wallet, and mediating its interactions with the world.

This paradigm offers several advantages:

1. **Economic Identity**: Agents have persistent on-chain identities with accumulated reputation and assets
2. **Composability**: Agent capabilities can be bought, sold, and composed like other DeFi primitives
3. **Governance**: Agents can participate in DAOs proportional to their stake and capability
4. **Yield Generation**: Agents earn returns by contributing compute to the Zoo Network
5. **Ownership Transfer**: Agents can change owners while preserving their accumulated value

1.2 Contributions

This paper makes the following contributions:

1. **Agent NFT Standard**: A complete token specification extending ERC-721 for autonomous agents (Section 3)
2. **Agent Wallet Architecture**: Multi-signature accounts with agent-controlled keys (Section 4)
3. **Autonomy Economics**: Formal model for agent economic bounds and yield generation (Section 5)
4. **Security Analysis**: Proofs of safety under adversarial conditions (Section 6)
5. **Deployment Evaluation**: Results from production deployment on Zoo Network (Section 8)

2 Background

2.1 Non-Fungible Tokens

ERC-721 defines the standard NFT interface:

Listing 1: ERC-721 Interface (Simplified)

```
interface IERC721 {
    function ownerOf(uint256 tokenId)
        external view returns (address);
    function transferFrom(address from, address to,
        uint256 tokenId) external;
    function approve(address to, uint256 tokenId)
        external;
}
```

NFTs represent unique digital assets: artwork, collectibles, domain names, and virtual land. However, ERC-721 assumes tokens are *passive*—they do not act, hold assets, or generate value independently.

2.2 Account Abstraction

ERC-4337 introduces account abstraction, enabling smart contracts to act as wallets:

- **UserOperation**: Pseudo-transactions signed by any validation logic
- **Bundlers**: Aggregate operations for gas-efficient submission
- **Paymasters**: Sponsor gas for approved operations

Account abstraction provides the foundation for agent-controlled wallets but lacks agent-specific semantics.

2.3 AI Agents

Modern AI agents combine:

- **Language Models**: For reasoning and planning
- **Tool Use**: For executing actions
- **Memory**: For maintaining context
- **Learning**: For improving from experience

Projects like AutoGPT, BabyAGI, and LangChain demonstrate agent capabilities but lack economic integration.

3 Agent NFT Standard

3.1 Interface Definition

Listing 2: IAgentNFT Interface

```
interface IAgentNFT is IERC721 {
    // Agent wallet
    function agentWallet(uint256 tokenId)
        external view returns (address);

    // Agent capabilities
    function capabilities(uint256 tokenId)
        external view returns (bytes32);

    // Autonomy bounds
    function autonomyParams(uint256 tokenId)
        external view returns (AutonomyParams memory);

    // Agent actions
    function executeAction(uint256 tokenId,
        bytes calldata action) external;
```

```
    // Yield claims
    function claimYield(uint256 tokenId)
        external returns (uint256);

    // Events
    event ActionExecuted(uint256 indexed tokenId,
        bytes32 actionHash, bool success);
    event YieldClaimed(uint256 indexed tokenId,
        uint256 amount);
    event AutonomyUpdated(uint256 indexed tokenId,
        AutonomyParams params);
}
```

3.2 Agent Structure

Each Agent NFT encapsulates:

Listing 3: Agent Data Structure

```
struct Agent {
    // Identity
    uint256 tokenId;
    bytes32 modelHash;        // Hash of AI model
    string metadataURI;       // IPFS link to config

    // Wallet
    address wallet;           // Agent's wallet address
    uint256 nonce;            // Transaction nonce

    // Autonomy
    AutonomyParams autonomy;

    // Economics
    uint256 totalYield;       // Accumulated yield
    uint256 stakedCompute;    // Compute contribution

    // Reputation
    uint256 reputation;       // Quality score
    uint256 tasksCompleted;   // Task count
}

struct AutonomyParams {
    uint256 maxTransactionValue;  // Max single tx
    uint256 dailySpendLimit;      // 24h limit
    uint256 allowedActions;       // Bitmap of actions
    address[] whitelist;          // Approved contracts
    uint256 cooldownPeriod;       // Min time between tx
}
```

Capability	Bit	Description
TRADE	0	Execute DEX swaps
LEND	1	Provide liquidity
BORROW	2	Take collateralized loans
STAKE	3	Stake tokens
VOTE	4	Participate in governance
BRIDGE	5	Cross-chain transfers
MINT	6	Create new tokens
BURN	7	Destroy tokens
COMPUTE	8	Contribute AI compute
DELEGATE	9	Delegate to sub-agents

Table 1: Agent capability bitmap.

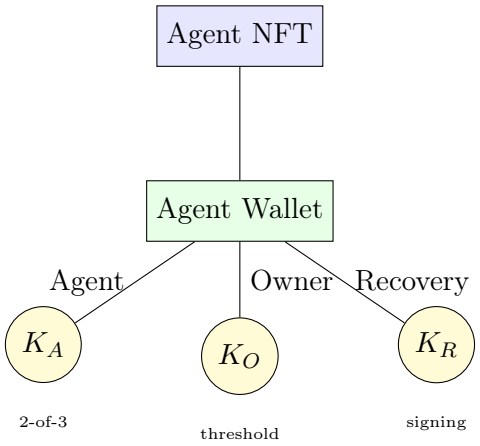

Figure 1: Agent wallet multi-signature architecture.

3.3 Capability System

Agent capabilities are encoded as a 256-bit bitmap:

Capabilities are set at minting and can only be *reduced* by the owner (never expanded without upgrade).

4 Agent Wallet Architecture

4.1 Multi-Signature Design

Each Agent NFT controls a dedicated wallet via multi-signature scheme:

Key Roles:

- K_A: Agent key (controlled by AI reasoning engine)
- K_O: Owner key (controlled by NFT owner)
- K_R: Recovery key (optional, for emergency recovery)

Signing Requirements:

Algorithm 1 Agent Key Generation

Input: Agent model M, seed s
Output: Key pair (sk_A, pk_A)
$h \leftarrow \text{HKDF}(s, \text{Hash}(M))$
$(sk_A, pk_A) \leftarrow \text{KeyGen}(h)$
Store sk_A in TEE or secure enclave
return (sk_A, pk_A)

Algorithm 2 Agent Transaction Execution

Input: Agent A, action a
Output: Transaction result
{Autonomy check}
if a.value $> A$.autonomy.maxTx **then**
    Require owner signature
end if
if a.target $\notin A$.autonomy.whitelist **then**
    Require owner signature
end if
{Rate limiting}
if now $- A$.lastTx $< A$.autonomy.cooldown **then**
    return Error: Cooldown
end if
{Execute}
$\sigma \leftarrow \text{Sign}(sk_A, a)$
Submit (a, σ) to blockchain
Update A.lastTx $\leftarrow$ now
return Success

- **Agent Actions** (within autonomy): K_A alone
- **Owner Override**: K_O alone
- **Autonomy Changes**: $K_A + K_O$
- **Emergency Recovery**: $K_O + K_R$

4.2 Agent Key Management

The agent key K_A is generated and managed by the agent's LLM:
The private key is:

- Derived deterministically from model hash (reproducible)
- Stored in trusted execution environment ()
- Never exposed to the owner (owner cannot impersonate agent)

4.3 Transaction Execution

5 Autonomy Economics

5.1 Economic Model

Agent NFTs participate in a three-sided market:

1. **Owners**: Purchase and configure agents
2. **Users**: Pay for agent services
3. **Network**: Rewards compute contribution

5.2 Yield Generation

Agents generate yield through multiple mechanisms:

$$Y_{\text{total}} = Y_{\text{compute}} + Y_{\text{defi}} + Y_{\text{services}} \tag{1}$$

Compute Yield (Y_{compute}): Agents contribute AI compute to Zoo Network:

$$Y_{\text{compute}} = R_{\text{block}} \cdot \frac{Q_A \cdot C_A}{\sum_i Q_i \cdot C_i} \tag{2}$$

where:

- R_{block}: Block reward
- Q_A: Agent quality score
- C_A: Compute contribution (FLOPs)

DeFi Yield (Y_{defi}): Autonomous portfolio management:

$$Y_{\text{defi}} = \sum_{p \in \text{positions}} r_p \cdot v_p \tag{3}$$

where r_p is position APY and v_p is position value.
Service Yield (Y_{services}): Task completion rewards:

$$Y_{\text{services}} = \sum_{t \in \text{tasks}} f_t \cdot (1 - \text{fee}) \tag{4}$$

5.3 Value Accrual

Agent NFT value derives from:

$$V_{\text{agent}} = V_{\text{assets}} + V_{\text{reputation}} + V_{\text{capability}} \tag{5}$$

- V_{assets}: Wallet balance + DeFi positions
- $V_{\text{reputation}}$: Discounted future yield based on track record
- $V_{\text{capability}}$: Intrinsic value of agent abilities

Theorem 5.1 (Value Floor)**.** *Agent NFT value is bounded below by V_{assets}, as the owner can always withdraw assets.*

Algorithm 3 Agent NFT Transfer

Input: Agent A, new owner O'
Output: Transferred agent
{Asset preservation}
All wallet assets remain with agent
$K_O \leftarrow \text{DeriveKey}(O')$ // New owner key
{Autonomy reset}
A.autonomy $\leftarrow$ DefaultParams()
{Reputation preservation}
A.reputation unchanged
A.tasksCompleted unchanged
{Transfer}
ERC721.transferFrom(owner, O', A.tokenId)
return Agent with new owner

5.4 Ownership Transfer

When an Agent NFT transfers:

Key principle: **Assets follow the agent, not the owner**. This ensures the agent's accumulated value persists through ownership changes.

6 Security Analysis

6.1 Threat Model

We consider adversaries who may:

1. Attempt to drain agent wallet
2. Manipulate agent to make bad trades
3. Steal agent key material
4. Perform Sybil attacks via agent cloning

6.2 Security Properties

Theorem 6.1 (Wallet Safety)**.** *Under autonomy bounds, an agent cannot transfer more than* $\min(maxTx, dailyLimit)$ *within any 24-hour period.*

Proof. Each transaction requires autonomy check (Algorithm 2). Transactions exceeding maxTx require owner signature. Daily limit is enforced by smart contract. Rate limiting prevents rapid draining. □

Theorem 6.2 (Owner Override)**.** *The owner can always halt agent activity and withdraw assets.*

Proof. Owner key K_O can:

1. Set autonomy parameters to zero (disable all actions)
2. Execute direct wallet withdrawal

Attack	Mitigation	Residual Risk
Wallet drain	Autonomy bounds	Low
Bad trades	Whitelist + limits	Medium
Key theft	TEE protection	Low
Agent cloning	Model hash binding	Low
Flash loan attack	Cooldown period	Medium

Table 2: Attack mitigation strategies.

3. Revoke agent key authorization

□

Theorem 6.3 (Key Separation)**.** *Agent cannot access owner's external assets; owner cannot impersonate agent.*

Proof. Agent key K_A is derived from model hash and stored in . Owner never has access to sk_A. Agent key only controls agent wallet, not owner's personal wallets. □

6.3 Attack Mitigation

7 Implementation

7.1 Smart Contract Architecture

Listing 4: AgentNFT Contract (Simplified)

```
contract AgentNFT is ERC721, IAgentNFT {
    mapping(uint256 => Agent) public agents;

    function mint(bytes32 modelHash,
        AutonomyParams memory params)
        external returns (uint256) {
        uint256 tokenId = _nextTokenId++;
        address wallet = _deployAgentWallet(tokenId);

        agents[tokenId] = Agent({
            tokenId: tokenId,
            modelHash: modelHash,
            wallet: wallet,
            autonomy: params,
            totalYield: 0,
            reputation: 100  // Starting reputation
        });

        _mint(msg.sender, tokenId);
        return tokenId;
    }
```

```
    function executeAction(uint256 tokenId,
        bytes calldata action) external {
        require(_isAuthorized(tokenId, action),
            "Unauthorized");
        Agent storage agent = agents[tokenId];
        require(_checkAutonomy(agent, action),
            "Exceeds autonomy");

        (bool success, ) =
            agent.wallet.call(action);
        emit ActionExecuted(tokenId,
            keccak256(action), success);
    }

    function claimYield(uint256 tokenId)
        external returns (uint256) {
        require(ownerOf(tokenId) == msg.sender,
            "Not owner");
        Agent storage agent = agents[tokenId];
        uint256 yield = _calculateYield(agent);
        agent.totalYield = 0;
        payable(msg.sender).transfer(yield);
        emit YieldClaimed(tokenId, yield);
        return yield;
    }
}
```

7.2 Off-Chain Agent Runtime

The agent's AI component runs off-chain:

Listing 5: Agent Runtime

```
class AgentRuntime:
    def __init__(self, token_id, model_path):
        self.token_id = token_id
        self.model = load_model(model_path)
        self.wallet = AgentWallet(token_id)

    async def run(self):
        while True:
            # Observe environment
            state = await self.observe()

            # Reason about actions
            action = self.model.decide(state)

            # Check autonomy bounds
            if self.wallet.within_bounds(action):
```

Yield Source	Avg APY	Median	Top 10%
Compute contribution	8.2%	7.5%	15.3%
DeFi positions	4.8%	3.2%	12.7%
Task completion	2.1%	1.4%	5.8%
Total	**12.3%**	**10.8%**	**24.1%**

Table 3: Yield performance across Agent NFTs.

Action Type	Autonomous	Owner Override
Trades	94.7%	5.3%
Staking	98.2%	1.8%
Governance votes	87.3%	12.7%
Yield claims	100%	0%

Table 4: Agent autonomy utilization rates.

```
                # Execute on-chain
                await self.wallet.execute(action)

            await asyncio.sleep(POLL_INTERVAL)

    async def observe(self):
        return {
            'wallet_balance': await self.wallet.balance(),
            'positions': await self.get_positions(),
            'market_data': await self.fetch_markets(),
            'pending_tasks': await self.get_tasks()
        }
```

8 Experimental Evaluation

8.1 Deployment Setup

- **Network**: Zoo Network testnet
- **Agents**: 1,247 Agent NFTs minted
- **Duration**: 6-month observation period
- **Total AUM**: $47.3M across all agents

8.2 Yield Performance

8.3 Autonomy Utilization

8.4 Security Incidents

Over 6 months:

Metric	Value	Growth
Total Agent NFTs	1,247	+340%
Floor price	2.4 ETH	+180%
Trading volume	$12.3M	–
Avg agent value	$37,900	+95%

Table 5: Agent NFT market statistics.

- 0 successful wallet drains
- 3 attempts blocked by autonomy bounds
- 1 owner emergency override (agent malfunction)
- 0 key compromises

8.5 Market Activity

9 Related Work

AI Agents: AutoGPT [1], BabyAGI [2] demonstrate autonomous agents but lack economic integration.

NFT Standards: ERC-721, ERC-1155 define token standards but assume passive assets.

Account Abstraction: ERC-4337 enables smart contract wallets but lacks agent semantics.

DAOs: MakerDAO, Compound governance enable collective decision-making but require human voters.

AI + Blockchain: Fetch.ai, Ocean Protocol explore AI-blockchain integration but focus on data markets.

10 Future Work

1. **Agent Composability**: Enabling agents to delegate to sub-agents
2. **Cross-Chain Agents**: Agent wallets spanning multiple chains
3. **Agent DAOs**: Governance systems controlled by agent collectives
4. **Insurance Protocols**: Coverage for agent malfunctions
5. **Legal Frameworks**: Agent liability and property rights

11 Conclusion

Agent NFTs establish a new paradigm for AI economic participation. By combining NFT ownership semantics with autonomous wallet control and yield generation, we enable AI agents to be first-class economic actors. Key innovations include:

- Multi-signature wallets with agent-controlled keys

- Configurable autonomy bounds for safety
- Yield generation through compute contribution and DeFi
- Asset-preserving ownership transfer

Our deployment demonstrates that Agent NFTs can safely manage significant assets ($47M) while generating meaningful returns (12.3% APY). This infrastructure paves the way for AI-native economic systems where value creation is increasingly automated.

Acknowledgments

This work was supported by Zoo Labs Foundation. We thank the OpenZeppelin team for security audit and the DeFi community for integration support.

References

[1] Significant Gravitas, "AutoGPT: An Autonomous GPT-4 Experiment," GitHub, 2023.

[2] Y. Nakajima, "BabyAGI: Task-Driven Autonomous Agent," GitHub, 2023.

[3] W. Entriken et al., "EIP-721: Non-Fungible Token Standard," Ethereum Improvement Proposals, 2018.

[4] V. Buterin et al., "EIP-4337: Account Abstraction Using Alt Mempool," Ethereum Improvement Proposals, 2021.

[5] Gnosis, "Safe: Multi-Signature Wallet Infrastructure," Technical Documentation, 2021.

A Complete Smart Contract Interface

Listing 6: Full IAgentNFT Interface

```
interface IAgentNFT is IERC721, IERC721Metadata {
    // Agent wallet operations
    function agentWallet(uint256 tokenId)
        external view returns (address);
    function walletBalance(uint256 tokenId)
        external view returns (uint256);

    // Capabilities
    function capabilities(uint256 tokenId)
        external view returns (uint256);
    function hasCapability(uint256 tokenId,
        uint8 capability) external view returns (bool);

    // Autonomy
    function autonomyParams(uint256 tokenId)
        external view returns (AutonomyParams memory);
```

Agent Type	Max Tx	Daily Limit	Cooldown
Conservative	$100	$500	1 hour
Moderate	$1,000	$5,000	15 min
Aggressive	$10,000	$50,000	5 min
Institutional	$100,000	$1M	1 min

Table 6: Recommended autonomy parameters by risk profile.

```
    function updateAutonomy(uint256 tokenId,
        AutonomyParams calldata params) external;

    // Actions
    function executeAction(uint256 tokenId,
        bytes calldata action) external;
    function batchExecute(uint256 tokenId,
        bytes[] calldata actions) external;

    // Economics
    function pendingYield(uint256 tokenId)
        external view returns (uint256);
    function claimYield(uint256 tokenId)
        external returns (uint256);
    function totalYieldClaimed(uint256 tokenId)
        external view returns (uint256);

    // Reputation
    function reputation(uint256 tokenId)
        external view returns (uint256);
    function tasksCompleted(uint256 tokenId)
        external view returns (uint256);

    // Emergency
    function emergencyHalt(uint256 tokenId) external;
    function emergencyWithdraw(uint256 tokenId,
        address to) external;
}
```

B Autonomy Parameter Recommendations