Monorite

Hasan Ibn Elmi

info@monorite.org

Monorite.org

1. Purpose

Monorite is an immutable cryptocurrency engineered to achieve linear appreciation in value, ensuring its exchange price never decreases. Designed to address the dual crises of fiat currency devaluation and cryptocurrency volatility, it functions as a decentralized, appreciating digital asset aligned with regulatory realities. Central banks' inflationary practices - excessive money printing, interest rate manipulation, and economic mismanagement - systematically erode the purchasing power of savings, while traditional cryptocurrencies remain vulnerable to unpredictable downward volatility. This instability is resolved through a transparent algorithmic model that enforces irreversible appreciation.

Initially, I envisioned Monorite as a standalone blockchain currency algorithmically anchored to the U.S. dollar via decentralized oracles. However, this approach risked direct competition with sovereign currencies like the dollar, inviting potential regulatory backlash such as exchange delistings or outright prohibitions. To mitigate these risks, Monorite is strategically designed as an ERC-20 token on Ethereum to leverage its global adoption, neutrality, and security while circumventing geopolitical friction. By operating within Ethereum's ecosystem, Monorite maintains a critical separation layer between its appreciation mechanism and sovereign monetary systems, enabling it to function as a politically neutral, code-governed store of value.

2. Introduction

Monorite (MNR) introduces a deterministic price discovery mechanism through an automated exchange rate system. Unlike traditional tokens and coins, MNR operates with a strictly controlled supply and price progression, eliminating direct peer-to-peer transfers in favor of a contract-mediated exchange system.

Monorite provides a groundbreaking financial model that ensures stability and predictable growth. The token's price appreciates in a mathematically controlled manner, eliminating sudden price swings and external manipulation. By requiring all transactions to go through the smart contract, Monorite enforces fairness, ensuring that all participants operate under the same rules without any advantage.

Additionally, Monorite functions as a self-sustaining market maker, removing the necessity for external liquidity providers. The contract itself maintains liquidity, ensuring that the buying and selling of tokens occur without market disruptions. The absence of liquidity slippage further strengthens the token's economic integrity.

3. Monetary Policy

Monorite's monetary policy enforces predictable price appreciation and liquidity sustainability through algorithmic controls, transaction-based halvings, and a fixed supply cap.

3.1 Exchange Rate Control & Transaction Mechanics

Direct peer-to-peer transfers are disabled to preserve pricing integrity. All transactions must occur through the contract's `buyTokens` and `sellTokens` functions:

```
function transfer(address, uint256) public pure override returns (bool) {
    revert DirectTransferDisabled(); // Direct transfers blocked
}
```

MNR/ETH Transaction Flow:

Process	Mechanics	Outcome
Buy	User sends ETH → Contract exchanges ETH for MNR at current rate → ETH added to reserves.	- ETH reserves increase. - Price rises by 0.0000000025 ETH/MNR.
Sell	User sends MNR → Contract transfers tokens to its reserves → ETH sent to seller from reserves.	- ETH reserves decrease. - MNR added to contract reserves for future buyers.

3.2 Linear Price Progression

The exchange rate increases algorithmically with each transaction, governed by:

$$ETH\ per\ MNR = Initial\ Rate + (Transactions\ \times Increment)$$

Key Parameters:

Parameter	Value
Initial Exchange Rate	1 MNR = 0.000041 ETH (1 ETH ≈ 24,390 MNR)
Price Increment per Transaction	+0.0000000025 ETH/MNR

Price Milestones:

Transactions (Millions)	ETH per 1 MNR
100	0.250041 ETH
200	0.500041 ETH
300	0.750041 ETH
400	1.000041 ETH

Note: The price reaches 1 ETH per MNR after ≈399.9836 million transactions

3.3 Halving Mechanism

To balance growth and liquidity, the price increment is halved every **400 million transactions**:

Halving Event	Transaction Threshold	Increment	MNR Supply	ETH per MNR
First	400 million	0.00000000125 ETH	38,000,000	1.000041 ETH
Second	800 million	0.000000000625 ETH	40,000,000	1.500041 ETH
Third	1.2 billion	0.0000000003125 ETH	40,000,000	1.750041 ETH

Key Mechanics:

- **ETH Reserves**: Funded by buy transactions and incremental price increases.
- **Minting**: 7 MNR minted into reserves every 100 transactions until the 40M cap.
- **Fixed Supply**: Enforced by ERC20Capped; total supply cannot exceed 40M MNR.

3.4 Post-Supply Cap Dynamics

Once the **40M MNR supply cap** is reached:

1. **Minting Stops**: No new tokens are created.
2. **Liquidity Recycling**:
    - Sold MNR are transferred to the contract's reserves, replenishing liquidity for future buyers.
    - ETH reserves depend on buy activity and price increments.
3. **Scarcity**: Fixed supply ensures long-term value preservation, with reserves fluctuating based on market activity.

3.5 Liquidity Management

- **Initial Liquidity**: 2 million MNR locked in the contract at deployment.
- **Transaction-Triggered Minting**: 7 MNR added to reserves every 100 transactions until 40M cap.
- **Reserve Sustainability**: Post-cap, liquidity relies entirely on tokens recycled from sells.

4. Tokenomics

Monorite's tokenomics are governed by immutable smart contract logic, ensuring scarcity, liquidity sustainability, and predictable price appreciation.

4.1 Supply Mechanics

4.1.1 Maximum Supply Cap

Monorite enforces a hard-capped supply of 40 million MNR tokens through the ERC20Capped standard:

```
uint256 public constant MAX_SUPPLY = 40_000_000 * 1e18; // 40M MNR
```

- **Scarcity Guarantee**: No inflationary minting; total supply cannot exceed 40M.

4.1.2 Initial Distribution

At deployment, **10 million MNR** were minted and allocated as follows:

Recipient	Amount	Purpose
Creator Address 1	2,000,000 MNR	MNR creator allocation.
Creator Address 2	3,000,000 MNR	
Creator Address 3	3,000,000 MNR	
Contract Liquidity Pool	2,000,000 MNR	Initial market liquidity.

4.1.3 Post-Deployment Minting

An additional **30,000,000 MNR** are minted incrementally into the contract's liquidity reserves to sustain market activity.

4.2 Minting Schedule

New tokens are algorithmically minted to balance supply growth and liquidity needs:

Parameter	Value
Trigger	Every 100 transactions
Tokens per Mint	7 MNR
Destination	Contract liquidity reserves
Termination	When totalSupply() reaches 40M MNR

Implementation:

```
function _mintTokensIfRequired() internal {
    uint256 remainingToMint = MAX_SUPPLY - totalSupply();
    uint256 amountToMint = remainingToMint >= TOKENS_PER_MINT ? TOKENS_PER_MINT :
remainingToMint;
    _mint(address(this), amountToMint); // Tokens added to reserves
}
```

4.3 Exchange Rate Mechanism

4.3.1 Initial Configuration

The exchange rate begins at deployment with:

```
uint256 public constant INITIAL_RATE = 41_000_000_000_000; // 0.000041 ETH per
```

- **Precision**: 18 decimal places (1 ETH = 1e18 wei).

4.3.2 Rate Progression

Each transaction increases the exchange rate by a fixed increment:

```
uint256 public constant INITIAL_INCREMENT = 2_500_000_000; // +0.0000000025 ETH per
```

Halving Events:

The increment is halved every **400 million transactions** to ensure long-term stability:

Halving Event	Transaction Threshold	New Increment
First	400 million	0.00000000125 ETH/MNR
Second	800 million	0.000000000625 ETH/MNR
Third	1.2 billion	0.0000000003125 ETH/MNR

Code Logic:

```
if (transactionCount >= nextHalvingThreshold) {
    currentIncrement = currentIncrement / 2;
    nextHalvingThreshold += 400_000_000;
}
```

4.4 Liquidity Model

4.4.1 Contract Reserves

The contract acts as a decentralized liquidity pool, holding:

- **ETH Reserves**: Funded by user purchases.
- **MNR Reserves**:
    - **Initial**: 2,000,000 MNR from deployment.
    - **Minted**: 7 MNR added every 100 transactions.
    - **Recycled**: Tokens from sell transactions.

4.4.2 Trading Mechanics

Action	Token Flow	ETH Flow	Reserve Impact
Buy	Tokens sent to user from contract reserves.	ETH from user → contract reserves.	MNR reserves ↓ / ETH reserves ↑
Sell	Tokens moved from user → contract reserves.	ETH from reserves → user.	MNR reserves ↑ / ETH reserves ↓

4.5 Economic Implications

Scarcity & Demand Dynamics

- **Fixed Supply**: The 40M cap ensures scarcity, with no post-cap inflation.
- **Price Appreciation**: Algorithmic rate increases incentivize early participation.

Liquidity Sustainability

- **Pre-Cap**: Reserves grow via minting (7 MNR/100 transactions).
- **Post-Cap**: Reserves rely on tokens recycled from sells.

Market Stability

- **Halvings**: Slow appreciation over time to balance demand.
- **Self-Regulation**: ETH reserves fund sells; MNR reserves fuel buys.

5. Technical and Security Architecture

5.1 Core Technical Architecture

5.1.1 Smart Contract Foundations

```
contract Monorite is ERC20Capped, ReentrancyGuard {
    constructor() ERC20("Monorite", "MNR") ERC20Capped(MAX_SUPPLY) {
        // Initialization logic
    }
}
```

- Inherits OpenZeppelin's audited ERC20Capped standard
- Implements ReentrancyGuard for transaction protection
- Enforces 40M token supply cap

Transfer Restrictions

```
function transfer(address, uint256) public pure override returns (bool) {
    revert DirectTransferDisabled();
}
```

- Disables standard ERC20 transfers
- Forces exchange-mediated transactions

5.1.2 Precision Engine

```
function _calculateTokenAmount(uint256 ethAmount, uint256 _rate) internal pure {
    uint256 numerator = ethAmount * 1e18;
    require(numerator / 1e18 == ethAmount, "Multiplication overflow");
    uint256 tokens = numerator / _rate;
    require(tokens > 0, "Amount too small");
}
```

- 18 decimal precision for ETH/MNR
- Overflow protection in calculations
- Minimum amount enforcement

5.2 Exchange Architecture

5.2.1 Dynamic Pricing Mechanism

```
function _updateExchangeRate() internal {
    uint256 newRate = oldRate + currentIncrement;
    if (newRate < oldRate) revert RateOverflow();
    exchangeRate = newRate;
}
```

- Linear rate progression
- Halving mechanism every 400M transactions
- Overflow protection

5.2.2 Order Processing System

```
function _handleFullBuy(uint256 tokenAmount, uint256 _rate) internal {
    _transfer(address(this), msg.sender, tokenAmount);
    emit TokenPurchase(msg.sender, msg.value, tokenAmount, _rate);
}
function _handleFullSell(uint256 tokenAmount) internal {
    _transfer(msg.sender, address(this), tokenAmount);
    emit TokenSale(msg.sender, tokenAmount, msg.value, exchangeRate);
}
```

- Full and partial order handling
- Automatic refund system
- Event logging

5.3 Minting Architecture

5.3.1 Transaction-Based Minting

```
function _mintTokensIfRequired(uint256 /* _count */) internal {
    uint256 currentSupply = totalSupply();
    uint256 remainingToMint = MAX_SUPPLY - currentSupply;
    uint256 amountToMint = remainingToMint >= TOKENS_PER_MINT ?
        TOKENS_PER_MINT : remainingToMint;
    _mint(address(this), amountToMint);
}
```

- 7 MNR minted every 100 transactions
- Automatic supply cap enforcement
- Contract reserve replenishment

5.4 Security Framework

5.4.1 Chain Protection

```
uint256 private immutable chainId;
function _validateChainId() internal view {
    if (block.chainid != chainId) revert WrongChain();
}
```

- Cross-chain replay prevention
- Network-specific execution
- Immutable chain binding

```
receive() external payable {
    revert DirectETHTransferNotAllowed();
}
function _safeTransferETH(address to, uint256 amount) internal {
    if (to == address(0)) revert InvalidRecipient();
    (bool success,) = to.call{value: amount}("");
    if (!success) revert ETHTransferFailed();
}
```

5.4.2 Transaction Security

- Direct ETH transfer prevention
- Safe ETH handling
- Recipient validation

5.5 Liquidity Management

5.5.1 Reserve System

```
event LiquidityChanged(uint256 contractEthBalance, uint256 contractTokenBalance);
```

- Real-time reserve tracking
- Balance verification
- Transparent monitoring

5.5.2 Partial Fill Handling

```
function _handlePartialBuy(uint256 availableTokens, uint256 _rate) internal {
    uint256 ethToSpend = _calculateEthAmount(availableTokens, _rate);
    uint256 refund = msg.value - ethToSpend;
    _transfer(address(this), msg.sender, availableTokens);
    if (refund > 0) {
        _safeTransferETH(msg.sender, refund);
        emit PartialBuyOrderRefunded(msg.sender, refund);
    }
}
```

- Maximum execution logic
- Automatic refunds
- Event emission

5.6 Event System

5.6.1 Transaction Events

```
event TokenPurchase(address buyer, uint256 ethSpent, uint256 tokensBought, uint256 newRate);
event TokenSale(address seller, uint256 tokensSold, uint256 ethReceived, uint256 newRate);
event PartialFill(address user, uint256 fulfilledAmount, uint256 returnedAmount);
```

- Trade tracking
- Price monitoring
- Liquidity changes

5.6.2 System Events

```
event HalvingOccurred(uint256 transactionCountAtHalving, uint256 newIncrement);
event MaxSupplyReached(uint256 totalSupply);
event TransactionCountIncremented(uint256 newTransactionCount);
```

- Protocol milestones
- Supply tracking
- Rate changes

5.7 Error Management

5.7.1 Custom Errors

```
error DirectTransferDisabled();
error DirectETHTransferNotAllowed();
error InvalidRecipient();
error ETHTransferFailed();
error RateOverflow();
error InsufficientBalance(uint256 requested, uint256 available);
```

- Gas-efficient error handling
- Detailed error information
- Clear failure identification

5.8 System Invariants

Invariant	Details
Supply Control	Total token supply is capped at 40M MNR, and minting halts once this cap is reached.
Rate Progression	The exchange rate increases in a strictly monotonic fashion, with controlled adjustments through scheduled halvings.
Transaction Safety	Transactions execute atomically with robust reentrancy protection, ensuring safe and secure operations.
Liquidity Rules	The protocol validates reserves rigorously, supports partial order fills, and provides refund guarantees to maintain market equilibrium.

6. Market Dynamics

6.1 Price Discovery Mechanism

Rate Progression

A. Initial Rate Setup

The contract establishes a fixed initial exchange rate, ensuring predictable token pricing at the start.

- At launch, 1 MNR = 0.000041 ETH.

```
uint256 public constant INITIAL_RATE = 41_000_000_000_000; // 0.000041 ETH
```

- This equates to approximately 24,390 MNR per 1 ETH.
- All calculations use 18 decimal precision for accuracy and consistency.

B. Incremental Rate Adjustments

Each transaction causes a small, predetermined increase in the token price.

```
uint256 public constant INITIAL_INCREMENT = 2_500_000_000; // 0.0000000025
```

- Every transaction increases the exchange rate by 0.0000000025 ETH per MNR.
- The increase ensures gradual and controlled price appreciation.
- The progression continues until a defined halving event occurs.

Halving System

Transaction-Based Halving Threshold:

To maintain long-term stability, Monorite employs a halving mechanism that adjusts the rate increment periodically.

```
uint256 public nextHalvingThreshold = 400_000_000;
```

- After 400 million transactions, the rate increment is reduced by 50%.
- This slows down the rate of appreciation as adoption grows.
- The halving process repeats at every subsequent 400 million transactions.

Long-Term Effects of Halving:

- The initial rapid growth phase transitions into a stabilized price appreciation model.
- Reduces price volatility while maintaining a sustainable appreciation rate.
- Prevents excessive inflation while ensuring steady demand and liquidity.

6.2 Market Mechanics

6.2.1 Buy Process

When a user initiates a buy transaction, the contract calculates the exact number of MNR tokens they will receive based on the current exchange rate.

```
tokensRequested = msg.value * 1e18 / exchangeRate;
```

- ETH provided by the user is exchanged for MNR tokens at the current rate.
- The contract updates the token balance and executes the purchase.
- The new exchange rate is automatically adjusted after each buy transaction.

Impact on Liquidity:

- The ETH balance of the contract increases, adding more liquidity.
- The exchange rate rises, ensuring continuous price appreciation.
- Event logs are emitted, providing transparency into all buy transactions.

6.2.2 Sell Process

When selling tokens, the contract calculates the amount of ETH the seller will receive.

```
ethToReceive = tokenAmount * exchangeRate / 1e18;
```

- The seller provides MNR tokens, which are exchanged for ETH.
- The contract updates balances and adjusts the rate accordingly.
- The transaction is executed immediately, ensuring seamless liquidity flow.

Market Effects:

- ETH balance decreases as funds are distributed to sellers.
- Price progression remains intact, maintaining a predictable model.
- All transactions contribute to the overall transaction count, impacting halvings.

6.3 Supply Economics

Initial Token Distribution

The initial supply is strategically allocated to ensure fair access and liquidity stability.

Creator Allocation

- A total of **8 million MNR** is allocated across **three creator's addresses**.
- This allocation is publicly visible and follows a **predefined split**.

Contract Liquidity Pool

- An additional 2 million MNR is reserved for the contract itself.
- These tokens enable buy/sell transactions within the contract.

- The liquidity pool is permanently locked, ensuring sustainable trading operations.

Token Minting and Supply Growth

New tokens are minted over time through an automated distribution mechanism.

Scheduled Minting

- The contract mints 7 MNR for every 100 transactions.
- These tokens are added directly to the contract's liquidity pool.
- The minting continues until the hard cap of 40 million MNR is reached.

Supply Control Measures

- No additional MNR can be created beyond the hard cap.
- This ensures controlled supply growth and prevents inflation.
- The structured release of tokens supports community-driven growth.

6.4 Market Stability

Exchange Rate Protections

Controlled Price Progression:

- The price can only increase following a fixed increment model.
- The halving system regulates long-term price growth.
- Overflow prevention mechanisms ensure price calculations remain valid.

Protection Against Manipulation:

- Direct transfers between users are disabled to prevent external price influence.
- All transactions must be executed through the contract's buy and sell functions.
- Event logs provide a transparent and immutable transaction history.

6.5 Liquidity Management Strategies

ETH and MNR Pool Balances

- The contract maintains a real-time balance of ETH and MNR.
- Buy transactions increase ETH reserves, while sell transactions reduce them.
- Minted tokens are continuously added to the contract's reserves.

Transaction Handling Mechanisms

- If a buy order exceeds available MNR, a partial fill is executed, and the remaining ETH is refunded.
- If a sell order exceeds available ETH, the contract limits the withdrawal, preventing liquidity depletion.

6.6 Long-Term Sustainability

Economic Model Overview

The Monorite ecosystem is designed for long-term viability and sustainable price appreciation.

Predictable Growth Trajectory:

- Controlled rate increases maintain steady token appreciation.
- Halving mechanisms slow down growth as the market matures.
- Demand-driven price discovery balances liquidity and accessibility.

Supply and Demand Balance:

- A fixed supply model ensures that MNR remains scarce and valuable.
- As transaction volume increases, price appreciation naturally slows.
- The liquidity pool grows over time, supporting long-term trading activity.

Market Evolution Phases

Early Market Phase:

- Higher rate increments encourage early adoption.
- Token supply is gradually distributed, building liquidity.
- Community engagement fosters a strong user base.

Mature Market Phase:

- Rate increments slow as transactions increase.
- The market stabilizes with reduced volatility.
- Monorite becomes a well-established digital asset.

7. Endnotes

7.1 Contract Code

Contract address: 0x95e2554870c7E4afd4775Bf328a56c63ba35549c

The contract code can be viewed publicly on GitHub and Etherscan:

GitHub
https://github.com/Mr-H-E/Monorite

Etherscan

https://etherscan.io/token/0x95e2554870c7e4afd4775bf328a56c63ba35549c?a=0x95e2554870c7e4afd4775bf328a56c63ba35549c

7.2 Mathematical Models

Monorite's monetary mechanics are enforced through integer arithmetic and safety checks to ensure precision and prevent overflows. Below are the mathematical models reflecting the contract's behavior.

8.3.1 Exchange Rate Progression

The exchange rate $R(n)$ (in wei) evolves as:

$$R(n) = \begin{cases} \mathrm{R(n\text{-}1)} + \Delta_m & \text{if } \mathrm{R(n-1)} + \Delta_m \leq 2^{256} - 1 \text{ (overflow check)} \\ Revert & otherwise \end{cases}$$

Where:

- $R(0) = 41 \times 10^{12}$ wei (0.000041 ETH/MNR)
- $\Delta_m = \left\lfloor \frac{\Delta_0}{2^m} \right\rfloor$, with $\Delta_0 = 2500 \times 10^6$ wei (0.0000000025 ETH/MNR)
- $m = \left\lfloor \frac{n}{400{,}000{,}000} \right\rfloor$ (halving count)

8.3.2 ETH ↔ MNR Conversions

A. ETH to MNR (Buy)

For ETH amount E (wei) and rate R (wei/MNR):

$$T_{received} = \begin{cases} \left\lfloor \frac{E \times 10^{18}}{R} \right\rfloor & if\ E \times 10^{18} \leq 2^{256} - 1 \text{ and } \left\lfloor \frac{E \times 10^{18}}{R} \right\rfloor > 0 \\ Revert & otherwise \end{cases}$$

B. MNR to ETH (Sell)

For MNR amount T (wei) and rate R (wei/MNR):

$$E_{received} = \begin{cases} \left\lfloor \frac{T \times R}{10^{18}} \right\rfloor & if\ T \times R \leq 2^{256} - 1 \text{ and } \left\lfloor \frac{T \times R}{10^{18}} \right\rfloor > 0 \\ Revert & otherwise \end{cases}$$

8.3.3 Halving Mechanism

The increment Δ_m after m halvings is computed as:

$$\Delta_m = \left\lfloor \frac{\Delta_0}{2^m} \right\rfloor \quad \text{(integer division)}$$

Halving trigger:

$n \geq H_m$ where $H_m = 400{,}000{,}000 \times m\ (m \geq 1)$

8.3.4 Supply Growth Model

Total supply S(n) (in MNR wei) after n transactions:

$$S(n) = \min\left(10^7 \times 10^{18} + 7 \times 10^{18} \times \left\lfloor \frac{n}{100} \right\rfloor,\ 40 \times 10^6 \times 10^{18}\right)$$

- **Termination:** Minting stops when $S(n) = 40 \times 10^6 \times 10^{18}$.

8.3.5 Partial Order Handling

A. Partial Buy (Insufficient MNR Reserves)

For available reserves T_{reserves} :

$$E_{\text{spent}} = \left\lfloor \frac{T_{\text{reserves}} \times R}{10^{18}} \right\rfloor \ (ETH\ wei)$$

$$E_{refund} = E_{sent} - E_{spent}$$

B. Partial Sell (Insufficient ETH Reserves)

$$T_{sold} = \left\lfloor \frac{E_{reserves} \times 10^{18}}{R} \right\rfloor \ (MNR\ wei)$$

$$T_{returned} = T_{offered} - T_{sold}$$

8.3.6 System Invariants

1. **Supply Cap**:

$$\forall n \geq 0:\ S(n) \leq 40 \times 10^{6} \times 10^{18}\ \ (40M\ MNR)$$

2. **Rate Monotonicity**:

$$\forall n > 0:\ R(n) > R(n\text{-}1)$$

3. **Reserve Adequacy**:

$$For\ buys:\ T_{received} \leq T_{reserves}$$
$$For\ sells:\ E_{received} \leq E_{reserves}$$

4. **Transaction Count Bound**:

$$n < 2^{256}\ \ (prevent\ integer\ overflow)$$

8.3.7 Precision & Constraints

- **Integer Arithmetic**: All calculations use unsigned 256-bit integers.
- **Overflow Checks**: Critical operations include `require` statements to prevent overflow/underflow.
- **Minimum Amounts**: Transactions revert if ETH/MNR amounts are too small (e.g., $T_{received} = 0$).
- **Wei Precision**:
    - 1 ETH = 10^{18} wei.
    - 1 MNR = 10^{18} internal units.

7.3 Legal Disclaimers

Protocol Autonomy

Monorite operates as an autonomous smart contract system with no central authority or administrative control. The protocol is immutable, meaning it cannot be paused, altered, or modified by any individual, entity, or organization. Users interact with the contract at their own risk, assuming full responsibility for their transactions and interactions within the system.

User Disclaimers

- **No Financial Advice:**
  This document serves as technical documentation of the Monorite protocol and does not constitute financial, investment, or legal advice.
- **Experimental Technology:**
  Monorite implements novel monetary mechanisms that have no long-term historical precedent. Users should be aware of the experimental nature of its economic model.
- **Tax Implications:**
  Users are solely responsible for understanding and complying with any applicable tax obligations related to their participation in the Monorite ecosystem, including reporting capital gains, losses, or other taxable events.

Enjoy.