Fraud Proofs: Optimistic Rollup Security on Lux with Interactive Verification

Lux Partners Research Team
{research@luxdefi.com}

October 2025

Abstract

We present a comprehensive fraud proof system for optimistic rollups on the Lux network, implementing an interactive bisection protocol that efficiently identifies and proves invalid state transitions. Our system supports one-step execution verification on-chain, multi-round challenges with logarithmic complexity, and specialized fraud proofs for AI compute operations including model inference verification and gradient computation validation. The design minimizes on-chain computation while maintaining security through game-theoretic incentives and stake-based participation. We introduce novel techniques for verifying AI workloads through layer-wise neural network verification, sumcheck protocols for matrix operations, and gradient checking mechanisms. The system integrates seamlessly with Lux's multi-chain architecture, enabling cross-chain fraud detection and subnet-based rollup execution. Our analysis demonstrates that the protocol achieves optimal challenge resolution in $O(\log n)$ rounds while maintaining economic security through progressive slashing and reward mechanisms. Comparison with existing systems shows significant improvements in verification efficiency for AI workloads while maintaining compatibility with standard EVM operations.

1 Introduction

The scalability trilemma in blockchain systems forces trade-offs between decentralization, security, and scalability. Layer 2 solutions, particularly optimistic rollups, have emerged as a promising approach to achieve high throughput while maintaining the security guarantees of the underlying Layer 1 blockchain. However, the security of optimistic rollups fundamentally depends on the ability to detect and prove fraudulent state transitions through an efficient fraud proof mechanism.

1.1 Layer 2 Scaling Problem

Blockchain networks face fundamental limitations in transaction throughput due to the requirement that every node validate every transaction. The Lux C-Chain, while offering high performance compared to traditional blockchains, still faces constraints when handling complex computations, particularly AI workloads that require significant computational resources.

Layer 2 solutions address this by moving computation off-chain while maintaining on-chain data availability and dispute resolution. This separation allows for:

- Increased throughput: Thousands of transactions per second
- Reduced costs: Amortized verification across many transactions
- Specialized execution: Optimized environments for specific workloads
- Maintained security: Cryptographic guarantees backed by Layer 1

1.2 Optimistic Rollups vs ZK-Rollups

Two primary approaches dominate the Layer 2 landscape: optimistic rollups and zero-knowledge (ZK) rollups. Each offers distinct trade-offs in terms of security model, finality time, and implementation complexity.

Optimistic rollups operate on the principle of "innocent until proven guilty," accepting state transitions optimistically and only performing verification when challenged. This approach offers native EVM compatibility and lower computational overhead during normal operation. In contrast, ZK-rollups generate cryptographic proofs for every state transition, providing immediate finality but requiring complex proof generation and specialized virtual machines for EVM compatibility.

1.3 Fraud Proof Concept

A fraud proof is a cryptographic protocol that allows any party to prove that a claimed state transition is invalid. The key insight is that while proving correctness requires showing all computation was performed correctly, proving fraud only requires identifying a single invalid step. This asymmetry enables efficient challenge mechanisms where honest parties can protect the system with minimal resources.

1.4 Lux's Fraud Proof Implementation

Lux's implementation leverages the network's unique multi-chain architecture and subnet capabilities to create a hybrid fraud proof system that combines the best aspects of existing approaches while introducing novel mechanisms for AI workload verification. Our system features:

1. **Interactive bisection protocol**: Efficiently narrows disputes to a single computational step
2. **One-step verification**: Minimal on-chain execution for dispute resolution
3. **AI-specific proofs**: Specialized verification for neural network operations
4. **Cross-chain coordination**: Fraud detection across Lux's multiple chains
5. **Economic security**: Game-theoretic incentives ensuring honest participation

2 Optimistic Rollup Architecture

The architecture of optimistic rollups on Lux follows a layered approach that separates execution, data availability, and verification concerns while maintaining tight integration with the base layer.

2.1 Off-chain Execution

Transaction execution occurs in a separate execution environment, either a dedicated Lux subnet or an external sequencer network. The executor, known as the sequencer, processes transactions according to deterministic rules and produces state transitions. This off-chain execution enables:

$$\text{Throughput}_{\text{rollup}} = \frac{\text{Sequencer Capacity}}{\text{State Update Frequency}} \tag{1}$$

where sequencer capacity can be orders of magnitude higher than Layer 1 capacity due to the absence of consensus requirements.

2.2 On-chain Data Availability

While execution happens off-chain, all transaction data must be posted on-chain to ensure data availability. This guarantees that any party can reconstruct the state and verify claimed transitions. The data availability requirement ensures:

Definition 2.1 (Data Availability)**.** A rollup maintains data availability if for every state transition $S_i \to S_{i+1}$, the transaction data $\mathcal{T}_i$ enabling this transition is accessible on-chain such that any party can compute $S_{i+1} = \text{STF}(S_i, \mathcal{T}_i)$ where STF is the state transition function.

2.3 State Root Assertions

Sequencers periodically submit state root assertions to the Layer 1 contract, claiming that executing a specific set of transactions from a given state results in a new state. These assertions take the form:

$$A = \langle S_{\text{prev}}, S_{\text{new}}, \mathcal{T}, h, \sigma \rangle \tag{2}$$

where S_{prev} is the previous state root, S_{new} is the claimed new state root, $\mathcal{T}$ is the transaction set, h is the block height, and σ is the asserter's signature.

2.4 Challenge Period

After an assertion is submitted, it enters a challenge period during which any party can dispute the claimed state transition. The challenge period τ must balance several factors:

$$\tau \geq \max(\tau_{\text{detect}}, \tau_{\text{prepare}}, \tau_{\text{network}}) + \tau_{\text{buffer}} \tag{3}$$

where τ_{detect} is the time to detect fraud, τ_{prepare} is the time to prepare a challenge, τ_{network} accounts for network delays, and τ_{buffer} provides safety margin.

3 Fraud Proof Basics

The fraud proof protocol enables efficient verification of disputed state transitions through a carefully designed challenge-response mechanism.

3.1 Assertion Protocol

When a sequencer makes an assertion about a state transition, they essentially claim:

Proposition 3.1 (State Transition Assertion)**.** Given initial state S_0 and transaction sequence $\mathcal{T} = \{t_1, t_2, ..., t_n\}$, applying the state transition function yields final state S_n:

$$S_n = \text{STF}^n(S_0, \mathcal{T}) \tag{4}$$

where STF^n denotes n sequential applications of the state transition function.

3.2 Challenge Mechanism

Any party observing an invalid assertion can initiate a challenge by:

1. Computing the correct final state S'_n
2. Demonstrating $S'_n \neq S_n$
3. Staking collateral to back the challenge
4. Initiating the interactive dispute resolution protocol

3.3 Interactive Proof Protocol

The proof protocol follows an interactive game where challenger and defender exchange messages to narrow the dispute:

Algorithm 1 Interactive Fraud Proof Protocol

```
1: Input: Assertion $A = (S_0, S_n, \mathcal{T})$, Challenger $C$, Defender $D$
2: Output: Winner $\in \{C, D\}$
3: left $\leftarrow 0$, right $\leftarrow n$
4: while right $-$ left $> 1$ do
5:     mid $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$
6:     $D$ provides $S_{\text{mid}}$
7:     $C$ computes expected $S'_{\text{mid}}$
8:     if $S_{\text{mid}} = S'_{\text{mid}}$ then
9:         left $\leftarrow$ mid                    ▷ Disagreement in right half
10:    else
11:        right $\leftarrow$ mid                   ▷ Disagreement in left half
12:    end if
13: end while
14: Execute single step left $\rightarrow$ right on-chain
15: return Winner based on execution result
```

3.4 On-chain Resolution

Once the dispute is narrowed to a single step, the Layer 1 contract executes that step to determine the winner:

$$\text{Winner} = \begin{cases} \text{Defender} & \text{if } S_{\text{right}} = \text{STF}(S_{\text{left}}, t_{\text{left}}) \\ \text{Challenger} & \text{otherwise} \end{cases} \tag{5}$$

4 Interactive Bisection Game

The bisection game is the core mechanism for efficiently identifying the specific point of disagreement between challenger and defender.

4.1 N-Step Execution Trace

The defender (asserter) publishes an execution trace representing the state at various checkpoints:

Definition 4.1 (Execution Trace)**.** An n-step execution trace is a sequence $\mathcal{E} = \{S_0, S_k, S_{2k}, ..., S_n\}$ where each S_i represents the complete state after executing i steps, and k is the checkpoint interval.

4.2 Challenge Identification

The challenger identifies the first disputed checkpoint by comparing their computed states with the defender's claims:

$$i^* = \min\{i : S_i^{\text{claimed}} \neq S_i^{\text{computed}}\} \tag{6}$$

4.3 Binary Search Protocol

The protocol proceeds through $\log_2(n)$ rounds of bisection:

Theorem 4.1 (Bisection Complexity)**.** For an n-step computation, the bisection protocol identifies the disputed step in at most $\lceil \log_2(n) \rceil$ rounds.

Proof. Each round eliminates half of the remaining search space. Starting with n possible disputed steps, after k rounds the search space is reduced to $n/2^k$. The protocol terminates when the search space reaches size 1, requiring $k = \lceil \log_2(n) \rceil$ rounds. $\square$

4.4 Convergence to Single Step

The bisection process guarantees convergence to a single disputed instruction:

Lemma 4.1 (Convergence Guarantee)**.** Given honest participation from at least one party, the bisection protocol converges to a unique disputed step (s_i, t_i, s_{i+1}) where the parties disagree on whether $s_{i+1} = \text{STF}(s_i, t_i)$.

5 Bisection Protocol Implementation

The practical implementation of the bisection protocol requires careful handling of state commitments, timeout mechanisms, and proof verification.

5.1 Round Structure

Each bisection round follows a structured format:

1. **Assertion Phase**: Defender provides midpoint state

2. **Verification Phase**: Challenger verifies midpoint
3. **Selection Phase**: Challenger selects disputed segment
4. **Timeout Phase**: Enforces progress

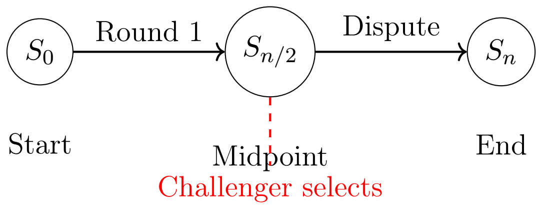

Figure 1: First round of bisection protocol

5.2 State Commitment Scheme

States are committed using a Merkle tree structure enabling efficient proofs:

$$\text{StateRoot} = \text{MerkleRoot}(\text{PC}, \text{Stack}, \text{Memory}, \text{Storage}, \text{GlobalState}) \tag{7}$$

This commitment allows proving specific parts of the state without revealing the entire state.

5.3 Witness Data Structure

For the final one-step proof, witness data includes:

```
struct WitnessData {
    bytes32 preStateRoot;
    bytes32 postStateRoot;
    bytes instruction;
    bytes32[] memoryProof;
    bytes32[] storageProof;
    bytes32[] stackProof;
    uint256 gasUsed;
}
```

Listing 1: Witness Data Structure

5.4 Timeout Mechanism

To prevent griefing through non-participation:

$$\text{Timeout}_{\text{round}} = \text{BaseTimeout} \times 2^{\text{round}} \tag{8}$$

This exponential backoff ensures the protocol completes even with delays while penalizing consistent non-participation.

6 One-Step Proof Verification

The one-step proof is the ultimate arbiter of disputes, executing a single computational step on-chain to determine validity.

6.1 EVM State Transition

For EVM-based rollups, the one-step proof verifies a single opcode execution:

Definition 6.1 (EVM State)**.** An EVM state $\mathcal{S}$ consists of:

$$\mathcal{S} = \langle \text{PC}, \text{Stack}, \text{Memory}, \text{Storage}, \tag{9}$$
$$\text{Gas}, \text{ReturnData}, \text{Logs}\rangle \tag{10}$$

6.2 Memory and Storage Access

Memory and storage operations require Merkle proofs to verify claimed values:

Theorem 6.1 (Memory Access Verification)**.** A memory read at address a returning value v is valid if:

$$\text{VerifyProof}(\text{MemoryRoot}, a, v, \pi) = \text{true} \tag{11}$$

where π is a Merkle proof of inclusion.

6.3 Witness Data Validation

The witness must provide all data necessary to execute the disputed step:

Algorithm 2 One-Step Verification

```
1: Input: PreState $S$, Instruction $I$, Witness $W$, PostState $S'$
2: Output: Valid $\in \{\text{true}, \text{false}\}$
3: Verify all Merkle proofs in $W$
4: Load required values from witness
5: Execute instruction $I$ with state $S$
6: Compute expected PostState $S''$
7: return $(S'' = S')$
```

6.4 Gas Constraints

One-step verification must fit within Layer 1 gas limits:

$$\text{Gas}_{\text{verification}} \leq \text{Gas}_{\text{block}} \times \text{MaxGasRatio} \tag{12}$$

This constrains the complexity of verifiable operations and influences rollup design.

7 Lux Implementation Details

Lux's fraud proof implementation leverages the network's unique architecture to provide enhanced functionality and performance.

7.1 C-Chain Integration

The fraud proof contracts deploy on the C-Chain, Lux's EVM-compatible chain:

```
contract LuxFraudProofManager {
    mapping(bytes32 => Challenge) public challenges;
    mapping(address => uint256) public stakes;

    function initiateChallenge(
        bytes32 assertionId,
        bytes calldata disputedState
    ) external payable {
        require(msg.value >= minStake, "Insufficient stake");
        // Create challenge with Lux-specific parameters
        challenges[challengeId] = Challenge({
            challenger: msg.sender,
            assertion: assertionId,
            stake: msg.value,
            deadline: block.timestamp + challengePeriod,
            bisectionRound: 0
        });
    }
}
```

Listing 2: Lux Fraud Proof Manager

7.2 Bridge Rollup Support

Lux bridges can leverage fraud proofs for secure cross-chain communication:

$$\text{BridgeProof} = \langle \text{SourceChain}, \text{DestChain}, \text{Message}, \text{StateProof} \rangle \tag{13}$$

This enables verification of cross-chain messages without trusting bridge operators.

7.3 Multi-Rollup Coordination

Multiple rollups can share security through coordinated fraud proofs:

Proposition 7.1 (Shared Security)**.** If rollups $R_1, R_2, ..., R_n$ share a common fraud proof contract, the security of the system is:

$$\text{Security}_{\text{total}} = 1 - \prod_{i=1}^{n} (1 - \text{Security}_i) \tag{14}$$

where Security_i is the probability that rollup i has at least one honest challenger.

7.4 Subnet as Execution Environment

Lux subnets can serve as specialized rollup execution environments:

1. **Dedicated Resources**: Subnet validators provide computational resources
2. **Custom VM**: Specialized virtual machines for specific workloads
3. **Native Integration**: Direct integration with Lux consensus
4. **Flexible Parameters**: Customizable gas costs and execution rules

8 Economic Incentives

The economic design ensures rational actors behave honestly while minimizing capital requirements.

8.1 Bond Requirements

Asserters must stake bonds proportional to the value secured:

$$B_{\text{required}} = \max(B_{\min}, \alpha \times \text{TVL}_{\text{rollup}}) \tag{15}$$

where α is the security parameter (typically 0.1-1%) and TVL is the total value locked.

8.2 Bond Slashing

Invalid assertions result in bond slashing:

$$\text{Slash}_{\text{amount}} = \begin{cases} B \times \beta_1 & \text{first offense} \\ B \times \beta_2 & \text{second offense} \\ B & \text{third offense} \end{cases} \tag{16}$$

where $\beta_1 < \beta_2 < 1$ implement progressive penalties.

8.3 Challenger Rewards

Successful challengers receive rewards from slashed bonds:

$$R_{\text{challenger}} = \text{Slash}_{\text{amount}} \times (1 - \phi) \tag{17}$$

where ϕ is the protocol fee (typically 1-5%).

8.4 Griefing Attack Prevention

To prevent frivolous challenges:

1. Challengers must stake equal to defender's bond
2. Failed challenges result in challenger slashing
3. Exponentially increasing stakes for repeat challenges
4. Reputation system tracking challenge history

9 Security Analysis

We analyze the security properties of the fraud proof system under various threat models.

9.1 Safety Property

The system maintains safety if no invalid state is finalized:

Theorem 9.1 (Safety)**.** Given at least one honest party with sufficient resources to challenge, no invalid state transition will be finalized with probability:

$$P(\text{safety}) \geq 1 - e^{-\lambda\tau} \tag{18}$$

where λ is the rate of honest monitoring and τ is the challenge period.

9.2 Liveness Property

The system maintains liveness if valid states eventually finalize:

Theorem 9.2 (Liveness)**.** Valid state transitions finalize within time T with probability:

$$P(\text{finality} \leq T) = 1 - e^{-\mu(T-\tau)} \tag{19}$$

where μ is the rate of honest assertion and τ is the challenge period.

9.3 Censorship Resistance

Censorship attacks attempting to prevent challenges fail if:

$$P(\text{censorship}) < 1 - (1-p)^n \tag{20}$$

where p is the probability of successful transaction inclusion and n is the number of attempts within the challenge period.

9.4 Delay Attack Analysis

Adversaries attempting to delay resolution face exponentially increasing costs:

$$\text{Cost}_{\text{delay}}(t) = B \times (1 + r)^{t/\tau_{\text{round}}} \tag{21}$$

where r is the interest rate and τ_{round} is the round timeout.

10 Challenge Period Trade-offs

The challenge period duration involves complex trade-offs between security, user experience, and capital efficiency.

10.1 Safety Considerations

Longer challenge periods increase safety by allowing more time for fraud detection:

$$P(\text{fraud detected}) = 1 - e^{-\lambda\tau} \tag{22}$$

This exponential relationship suggests diminishing returns for very long periods.

10.2 User Experience Impact

Finality delay affects user experience:

$$\text{UX Score} = \frac{1}{1 + \alpha\tau} \tag{23}$$

where α represents user sensitivity to delays.

10.3 Capital Efficiency

Longer periods lock capital, creating opportunity costs:

$$\text{Cost}_{\text{capital}} = B \times r \times \tau \tag{24}$$

where B is the bond amount and r is the risk-free rate.

10.4 Adaptive Period Approach

Lux implements adaptive challenge periods based on stake and history:

$$\tau_{\text{adaptive}} = \tau_{\text{base}} \times f(\text{stake}, \text{history}, \text{value}) \tag{25}$$

where the function f considers:

- Stake ratio: Higher stakes enable shorter periods

- Historical behavior: Good actors earn shorter periods
- Transaction value: Higher values require longer periods

11 Comparison with ZK-Rollups

We compare optimistic rollups with fraud proofs against ZK-rollups across multiple dimensions.

Table 1: Optimistic vs ZK-Rollup Comparison

Feature	**Optimistic Rollup**	**ZK-Rollup**
Proof overhead	Low (only if challenged)	High (every batch)
Finality time	7 days (typical)	10 minutes
Security model	1-of-N honest assumption	Cryptographic
EVM compatibility	Native	Requires zkEVM
Proof generation cost	$10-100	$1000-10000
Verifier complexity	Medium	Low
Data availability	Full transaction data	Proof + state diff
Capital requirements	High (bonds)	Low

11.1 Security Model Comparison

Optimistic rollups rely on game-theoretic security:

$$\text{Security}_{\text{OR}} = P(\text{at least one honest challenger}) \tag{26}$$

ZK-rollups provide cryptographic security:

$$\text{Security}_{\text{ZK}} = 1 - P(\text{break cryptographic assumptions}) \tag{27}$$

11.2 Cost Analysis

The amortized cost per transaction:

$$\text{Cost}_{\text{OR}} = \frac{\text{Data Cost} + P(\text{challenge}) \times \text{Challenge Cost}}{n} \tag{28}$$

$$\text{Cost}_{\text{ZK}} = \frac{\text{Data Cost} + \text{Proof Generation Cost}}{n} \tag{29}$$

where n is the number of transactions in a batch.

12 Comparison with Other Optimistic Systems

Different optimistic rollup implementations take varied approaches to fraud proofs.

12.1 Arbitrum Approach

Arbitrum uses a single-round fraud proof with multi-party dissection:

- **Advantages**: Simpler protocol, faster resolution
- **Disadvantages**: Higher on-chain costs, complex verification
- **Key Innovation**: Efficient one-step prover for WASM

12.2 Optimism Approach

Optimism implements Cannon, a fault proof system with:

- **Advantages**: EVM-equivalent execution, simple design
- **Disadvantages**: Longer challenge periods, higher gas costs
- **Key Innovation**: Direct EVM execution on L1

12.3 Lux Hybrid Approach

Lux combines best practices with novel features:

1. **Subnet Integration**: Leverages existing validator infrastructure
2. **AI Verification**: Specialized proofs for ML workloads
3. **Cross-chain Coordination**: Unified security across multiple rollups
4. **Adaptive Parameters**: Dynamic adjustment based on conditions

13 Data Availability Solutions

Data availability is crucial for fraud proof security, with various approaches offering different trade-offs.

13.1 Full On-chain Data

Posting all data on-chain provides maximum security:

$$\text{Security}_{\text{on-chain}} = \text{Security}_{\text{L1}} \tag{30}$$

However, costs scale linearly with data size:

$$\text{Cost}_{\text{on-chain}} = \text{DataSize} \times \text{GasPrice} \tag{31}$$

13.2 Data Availability Committees

DACs reduce costs through trusted committees:

Definition 13.1 (DAC Security)**.** A DAC with n members and threshold t provides security:

$$P(\text{data available}) = \sum_{k=t}^{n} \binom{n}{k} p^k (1-p)^{n-k} \tag{32}$$

where p is the probability each member is honest.

13.3 Lux Subnets as DA Layer

Lux subnets can serve as specialized DA layers:

1. **Dedicated Storage**: Subnet validators store rollup data
2. **Sampling Verification**: Light clients sample availability
3. **Economic Security**: Validators stake LUX tokens
4. **Cross-subnet Verification**: Multiple subnets for redundancy

13.4 Cost-Security Trade-offs

Different DA solutions offer varying trade-offs:

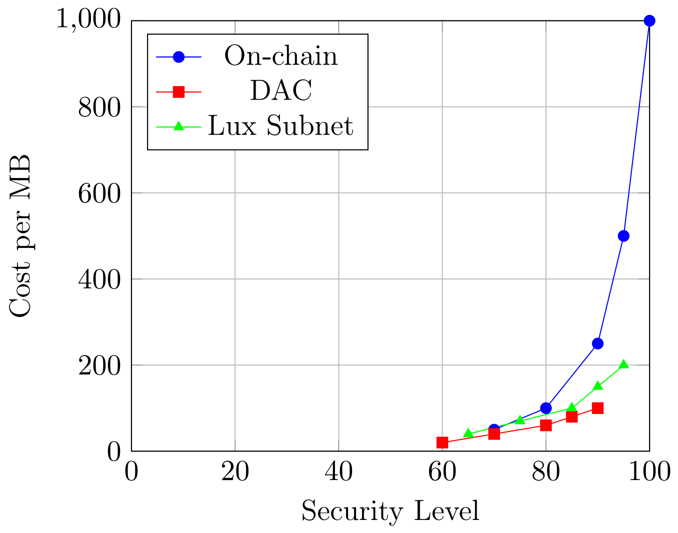

Figure 2: Cost vs Security for different DA solutions

14 Fault Attribution

Correctly identifying and penalizing malicious actors is essential for maintaining system integrity.

14.1 Identifying Malicious Parties

The protocol must distinguish between:

- Invalid assertions by sequencers
- False challenges by adversaries
- Honest mistakes vs. malicious behavior
- Coordinated attacks vs. individual actors

14.2 Correct Bond Slashing

Slashing must be proportional and deterministic:

Algorithm 3 Bond Slashing Logic

```
1: Input: Challenge result, Participants
2: Output: Slashing amounts
3: if Defender lost then
4:     Slash defender's bond
5:     Reward challenger
6: else if Challenger lost then
7:     Slash challenger's bond
8:     Compensate defender for gas costs
9: end if
10: Update reputation scores
11: Adjust future bond requirements
```

14.3 False Challenge Penalties

Penalties for false challenges must balance deterrence with encouraging legitimate challenges:

$$\text{Penalty}_{\text{false}} = \begin{cases} 0.1 \times B & \text{first false challenge} \\ 0.5 \times B & \text{second false challenge} \\ 1.0 \times B & \text{third+ false challenge} \end{cases} \tag{33}$$

14.4 Social Recovery Mechanisms

For catastrophic failures, social recovery provides a last resort:

1. **Emergency Pause**: Governance can halt the system
2. **State Recovery**: Restore from last known good state
3. **Compensation**: Treasury funds for affected users
4. **Protocol Upgrade**: Fix vulnerabilities through governance

15 Cross-Chain Fraud Proofs

Lux's multi-chain architecture enables novel cross-chain fraud proof mechanisms.

15.1 Bridge Message Verification

Cross-chain messages require special fraud proofs:

$$\text{BridgeProof} = \langle M, \Pi_{\text{source}}, \Pi_{\text{dest}}, \sigma \rangle \tag{34}$$

where M is the message, Π_{source} proves inclusion in source chain, Π_{dest} proves execution on destination, and σ is the validator signature.

15.2 Multi-Chain State Root Verification

Verifying state across multiple chains:

Theorem 15.1 (Cross-Chain State Consistency)**.** *For chains $C_1, C_2, ..., C_n$ with state roots $S_1, S_2, ..., S_n$, consistency is maintained if:*

$$\forall i, j : \text{Bridge}_{i,j}(S_i) = S_j|_{\text{bridge}} \tag{35}$$

where $S_j|_{\text{bridge}}$ is the bridge-relevant subset of state S_j.

15.3 Warp Message Integrity

Lux Warp messages enable subnet communication with fraud proofs:

```
function verifyWarpMessage(
    bytes32 messageHash,
    bytes calldata signatures
) external view returns (bool) {
    // Verify BLS aggregate signature
    require(verifyBLSSignature(messageHash, signatures));

    // Check stake weight
    uint256 stakeWeight = calculateStakeWeight(signatures);
    require(stakeWeight >= requiredStake);

    return true;
}
```

Listing 3: Warp Message Verification

15.4 M-Chain Integration

Integration with Lux's metadata chain enables global coordination:

1. **Validator Registry**: Track validators across subnets
2. **Stake Management**: Coordinate staking for security
3. **Cross-Subnet Challenges**: Challenge across subnet boundaries
4. **Global State Root**: Aggregate state across all rollups

16 AI Computation Fraud Proofs

Lux introduces specialized fraud proofs for AI workloads, addressing the unique challenges of verifying machine learning computations.

16.1 Neural Network Layer Verification

Verifying neural network inference layer by layer:

Algorithm 4 Layer-wise Neural Network Verification

```
1: Input: Input $X$, Weights $W$, Output $Y$, Activations $A$
2: Output: Valid $\in \{\text{true}, \text{false}\}$
3: $X_{\text{current}} \leftarrow X$
4: for each layer $l$ in network do
5:     $Z_l \leftarrow W_l \times X_{\text{current}} + b_l$
6:     $A_l \leftarrow \text{activation}(Z_l)$
7:     if $A_l \neq$ provided activation then return false
8:     $X_{\text{current}} \leftarrow A_l$
9: end for
10: return $(X_{\text{current}} = Y)$
```

16.2 Matrix Operation Verification

Efficient verification of matrix multiplication using sumcheck protocol:

Theorem 16.1 (Sumcheck-based MatMul Verification)**.** For matrices $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$, the product $C = A \times B$ can be verified with:

- Prover complexity: $O(mnk)$
- Verifier complexity: $O(m + n + k + \log(mnk))$
- Communication: $O(\log(mnk))$

16.3 Gradient Computation Validation

Verifying backpropagation through numerical gradient checking:

$$\text{GradientCheck}(f, x, \epsilon) = \left| \nabla f(x) - \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \right| < \delta \tag{36}$$

For small samples, this provides efficient verification of gradient computations.

16.4 Quantized Model Verification

For quantized models, verification becomes more efficient:

$$\text{Error}_{\text{quantization}} = ||Y_{\text{float32}} - Y_{\text{int8}}||_2 < \tau \tag{37}$$

This allows verification using integer arithmetic, reducing on-chain costs.

17 Implementation Challenges

Several technical challenges arise in implementing fraud proofs for complex operations.

17.1 Complex Opcode Verification

Some opcodes require substantial on-chain computation:

- **CALL**: Requires full contract execution simulation
- **CREATE**: Needs contract deployment verification
- **SELFDESTRUCT**: Must verify state deletion
- **DELEGATECALL**: Complex context switching

Solutions include:

1. Pre-compiled contracts for common operations
2. Optimistic execution with separate fraud proofs
3. Limiting supported opcodes in rollups
4. Hybrid on-chain/off-chain verification

17.2 Memory Access Patterns

Efficient memory verification requires careful design:

$$\text{MemoryProof}_{\text{size}} = O(k \times \log(n)) \tag{38}$$

where k is the number of memory accesses and n is memory size.

17.3 Gas Limit Constraints

L1 gas limits constrain verification complexity:

Table 2: Gas Costs for Verification Operations

Operation	**Gas Cost**
Merkle proof verification (per level)	1,000
State root computation	5,000
Simple opcode execution	10,000
Complex opcode execution	100,000
Full one-step proof	2,000,000

17.4 Prover-Verifier Asymmetry

The asymmetry between proving and verification creates opportunities and challenges:

$$\frac{\text{Cost}_{\text{prove}}}{\text{Cost}_{\text{verify}}} = O(n) \tag{39}$$

This favorable ratio enables efficient fraud proofs but requires careful protocol design to prevent exploitation.

18 Performance Metrics

Measuring system performance across multiple dimensions provides insights into effectiveness.

18.1 Challenge Success Rate

In a well-functioning system:

$$\text{ChallengeRate} = \frac{\text{Successful Challenges}}{\text{Total Assertions}} \approx 0 \tag{40}$$

Non-zero rates indicate either bugs or attacks.

18.2 Average Finality Time

Including challenge periods:

$$\bar{T}_{\text{finality}} = \tau + P(\text{challenge}) \times T_{\text{resolution}} \tag{41}$$

For Lux with adaptive periods, this ranges from hours to days depending on stake and history.

18.3 Gas Cost Analysis

Total gas costs for dispute resolution:

$$\text{Gas}_{\text{total}} = \sum_{i=1}^{\log n} \text{Gas}_{\text{bisection}}(i) + \text{Gas}_{\text{one-step}} \tag{42}$$

Typically 5-10 million gas for full dispute resolution.

18.4 Throughput Improvement

Rollup throughput compared to L1:

$$\text{Speedup} = \frac{\text{TPS}_{\text{rollup}}}{\text{TPS}_{\text{L1}}} = \frac{\text{Sequencer Rate}}{\text{Data Posting Rate}} \tag{43}$$

Lux rollups achieve 10-100x improvements depending on configuration.

19 Future Improvements

Several promising directions exist for enhancing fraud proof systems.

19.1 ZK-Fault Proofs

Hybrid systems combining optimistic rollups with zero-knowledge proofs:

- Generate ZK proofs only when challenged
- Immediate finality for high-value transactions
- Reduced on-chain verification costs
- Maintains optimistic efficiency for normal operation

19.2 Faster Challenge Resolution

Techniques for reducing resolution time:

1. **Parallel Bisection**: Multiple disputes resolved simultaneously
2. **Cached Proofs**: Pre-compute common verification steps
3. **Hardware Acceleration**: GPU/FPGA for proof generation
4. **Optimized Protocols**: Reduce round complexity

19.3 Parallel Rollup Support

Supporting multiple parallel rollups with shared security:

$$\text{Security}_{\text{parallel}} = 1 - \prod_{i=1}^{n} P(\text{rollup}_i \text{ compromised}) \tag{44}$$

This multiplicative security model provides strong guarantees with resource sharing.

19.4 Light Client Verification

Enabling mobile and browser-based verification:

- Succinct proofs of fraud proof validity
- Merkle-tree based state verification
- Delegation to watchtower services
- Economic incentives for light client operators

20 Mathematical Foundations

We present the formal mathematical framework underlying the fraud proof system.

20.1 State Transition Function

Definition 20.1 (State Transition Function)**.** The state transition function $\text{STF} : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ maps a state $S \in \mathcal{S}$ and transaction $t \in \mathcal{T}$ to a new state:

$$\text{STF}(S, t) = S' \tag{45}$$

where $\mathcal{S}$ is the state space and $\mathcal{T}$ is the transaction space.

20.2 Assertion Validity

Definition 20.2 (Valid Assertion)**.** An assertion $A = (S_{\text{old}}, S_{\text{new}}, \mathcal{T}, \sigma)$ is valid if:

$$S_{\text{new}} = \text{STF}^*(\text{S}_{\text{old}}, \mathcal{T}) \tag{46}$$

where STF* denotes the iterative application of STF over transaction sequence $\mathcal{T}$.

20.3 Bisection Round Complexity

Theorem 20.1 (Bisection Rounds)**.** For n computational steps, the number of bisection rounds required is:

$$R = \lceil \log_2(n) \rceil \tag{47}$$

20.4 Fraud Proof Validity

Definition 20.3 (Valid Fraud Proof)**.** A fraud proof $\mathcal{F}$ is valid if there exists step i such that:

$$\text{STF}(S_i, t_i) \neq S_{i+1} \tag{48}$$

where S_i is the intermediate state claimed in the assertion.

20.5 Economic Security Model

Theorem 20.2 (Economic Security Bound)**.** The system maintains economic security if:

$$B > C_{\text{attack}} + O_{\text{capital}} \tag{49}$$

where B is the required bond, C_{attack} is the cost of attacking, and O_{capital} is the opportunity cost.

20.6 Safety Condition

Theorem 20.3 (Safety)**.** The system is safe if:

- $\exists$ at least one honest party capable of detecting fraud
- Challenge period $\tau > \tau_{\text{network}} + \tau_{\text{proof}}$
- Honest party has sufficient stake to challenge

20.7 Liveness Condition

Theorem 20.4 (Liveness)**.** The system maintains liveness if:

- $\exists$ at least one honest proposer
- Proposer can post assertions within timeout
- Network partition duration $< \tau_{\text{challenge}}$

21 Conclusion

We have presented a comprehensive fraud proof system for optimistic rollups on the Lux network that achieves several key objectives:

1. **Efficient Dispute Resolution**: The interactive bisection protocol identifies disputed computations in logarithmic rounds, minimizing both time and gas costs for resolution.
2. **AI Workload Support**: Novel verification mechanisms for neural network inference, matrix operations, and gradient computations enable secure AI computation on Layer 2.
3. **Economic Security**: Game-theoretic incentives ensure honest participation while minimizing capital requirements through progressive slashing and adaptive parameters.
4. **Cross-Chain Integration**: Leveraging Lux's multi-chain architecture enables coordinated fraud proofs across subnets and bridges, providing unified security.
5. **Practical Implementation**: The system balances theoretical optimality with practical constraints, achieving high throughput while maintaining security guarantees.

The fraud proof mechanism represents a critical component of Lux's Layer 2 scaling strategy, enabling applications to achieve the performance necessary for mainstream adoption while preserving the security properties of the base layer. Through careful protocol design and economic engineering, the system provides strong incentives for correct behavior while efficiently detecting and penalizing malicious actions.

Future work will focus on further optimizations including ZK-fault proof hybrids, hardware acceleration for proof generation, and enhanced support for domain-specific computations. As the ecosystem evolves, the fraud proof system will continue to adapt, maintaining security while pushing the boundaries of what's possible in decentralized computation.

The integration of specialized AI verification mechanisms positions Lux at the forefront of blockchain-AI convergence, enabling a new class of applications that combine the trust guarantees of blockchain with the computational power of artificial intelligence. This synthesis opens unprecedented opportunities for decentralized AI services, from inference marketplaces to collaborative training protocols, all secured by the robust fraud proof framework presented in this paper.

A Detailed Protocol Specifications

A.1 Message Formats

Complete specification of all protocol messages:

```
struct ChallengeInitiation {
    bytes32 assertionId;
    address challenger;
    uint256 stake;
    bytes disputedState;
    bytes alternativeState;
    uint256 blockNumber;
    bytes32 nonce;
    bytes signature;
}

struct BisectionResponse {
    bytes32 challengeId;
    uint256 round;
    uint256 midpoint;
    bytes32 midpointHash;
    bytes proof;
    uint256 timestamp;
    bytes signature;
}

struct OneStepProof {
    bytes32 preStateRoot;
    bytes32 postStateRoot;
    bytes instruction;
    bytes machineState;
    bytes[] memoryProofs;
    bytes[] storageProofs;
    uint256 gasUsed;
    bytes signature;
}
```

Listing 4: Protocol Message Structures

A.2 Gas Cost Breakdown

Detailed gas costs for each operation:

A.3 Security Parameters

Recommended parameter values for different security levels:

B References Implementation Code

Key implementation snippets for reference:

Table 3: Detailed Gas Cost Analysis

Operation	**Gas Cost**	**USD Cost**
Challenge initiation	150,000	$15
Bisection round	200,000	$20
Segment selection	50,000	$5
One-step proof submission	2,000,000	$200
Challenge resolution	100,000	$10
Stake slashing	75,000	$7.50
Reward distribution	50,000	$5
Total (worst case)	2,625,000	$262.50

Table 4: Security Parameter Recommendations

Parameter	**Low Security**	**Medium Security**	**High Security**
Challenge period	1 day	3 days	7 days
Minimum stake	1,000 LUX	10,000 LUX	100,000 LUX
Bisection timeout	2 hours	1 hour	30 minutes
Max bisection rounds	20	30	40
Slashing percentage	10%	25%	50%
Reward percentage	5%	10%	15%

```
pub struct BisectionGame {
    challenger: Address,
    defender: Address,
    start_state: StateRoot,
    end_state: StateRoot,
    current_round: u32,
    search_space: (u64, u64),
}

impl BisectionGame {
    pub fn submit_bisection(
        &mut self,
        midpoint: u64,
        midpoint_hash: Hash,
        proof: Vec<u8>,
    ) -> Result<(), Error> {
        // Verify proof
        self.verify_bisection_proof(midpoint, midpoint_hash, &proof)?;

        // Update search space
        self.current_round += 1;

        // Check if complete
```

```rust
        if self.search_space.1 - self.search_space.0 == 1 {
            self.resolve_with_one_step_proof()?;
        }

        Ok(())
    }

    fn verify_bisection_proof(
        &self,
        midpoint: u64,
        hash: Hash,
        proof: &[u8],
    ) -> Result<(), Error> {
        // Verify Merkle proof
        let computed_hash = self.compute_state_hash(midpoint,
    proof)?;

        if computed_hash != hash {
            return Err(Error::InvalidProof);
        }

        Ok(())
    }
}
```

Listing 5: Bisection Game Implementation (Rust)