LCEN
Light Compute Execution Network

LCEN Technical White Paper

Blockchain-Native AI Agent Identity, Verifiable Training, and Quantitative Strategy Factories

2026

Title, Abstract, and Keywords

Title: LCEN Technical White Paper: A Blockchain-Native Architecture for AI Agent Identity, Verifiable Training, and Quantitative Strategy Factories

Abstract

This white paper presents LCEN (Light Compute Execution Network), a blockchain-native technical network and system architecture for AI agent identity, verifiable training, multi-agent strategy factories, quantitative simulation, and on-chain workload settlement. LCEN keeps model inference, agent training, strategy generation, backtesting, and risk evaluation off-chain, while anchoring agent identity, reputation, validation, task commitments, workload summaries, and settlement events on-chain. The system measures work through Compute Unit (CU), and LCEN Token functions as the settlement asset according to CU, validation status, and settlement rules. The paper specifies Model Mesh, ERC-8004-compatible Agent Passport, Strategy Factory DAG, Workload Accounting Ledger, Commitment Registry, Validation Gateway, blockchain privacy layer, confidential verification, data layer, RAG, feature engineering, execution simulation, audit artifacts, security controls, and phased engineering delivery.

Keywords

LCEN; LCEN Token; AI Agent; ERC-8004; Agent Identity; Verifiable Training; Model Mesh; Strategy Factory; Workload Accounting; Quantitative Simulation; Equity Curve; Blockchain Settlement; Blockchain Privacy Layer; Confidential Verification; TEE; zkML; MCP; A2A

Table of Contents

1 Introduction: Technical Problem and Scope
2 Design Principles and Technical Boundary
3 Model Selection: Model Mesh
4 Model Slicing, Training, and Evaluation Loop
5 ERC-8004-Compatible Agent Identity and Trust Layer
6 Agent Wallets, Permissions, and On-Chain Interaction
7 Multi-Agent Strategy Factory: Technical Principles
8 Data Layer, RAG, and Feature Engineering
9 Strategy Generation, Risk Control, and Execution Simulation
10 On-Chain Commitments, Workload Accounting, and LCEN Token
11 Strategy Simulation Methodology and Equity Curves
12 Verification Layer: TEE, zkML, and Re-execution
13 Audit Artifacts and Content Addressing
14 Blockchain Privacy Layer and Confidential Verification
15 Security and Risk Control
16 Engineering Implementation Architecture
17 Phased Roadmap
18 Developer Interfaces, SDKs, and Protocol Objects
19 Performance, Capacity, and Cost Evaluation
20 Threat Model and Red-Team Testing
21 Standards Compatibility, Interoperability, and Ecosystem Interfaces
22 Protocol State Consistency, Ledger Reconciliation, and Failure Recovery
23 Testing System, Audit Checklist, and Acceptance Criteria
24 Technical Boundaries, Reader Notes, and Conclusion
25 References

1 Introduction: Technical Problem and Scope

1.1 Background

Large model systems are moving from question-answer interfaces into agentic systems that can call tools, read market data, execute code, coordinate workflows, and trigger external systems. In financial data, on-chain state, strategy research, and automated execution, a single model response is not a trusted system. Model calls, data sources, tool permissions, strategy parameters, and execution outputs all need to be recorded, verified, and reproduced.

Blockchains provide public state, immutable events, programmable settlement, and cross-organization coordination. AI agents provide semantic understanding, research generation, tool use, and task orchestration. LCEN is based on a practical engineering boundary: the goal is not to place full LLM inference on-chain, but to connect off-chain intelligence with on-chain trust coordination.

LCEN, or Light Compute Execution Network, is defined as a technical network and system architecture for AI agent identity, verifiable training, multi-agent strategy factories, quantitative simulation, and on-chain workload settlement. Compute Unit (CU) measures work such as model calls, validation, strategy experiments, and audit submissions. LCEN Token is the settlement asset used to complete value settlement according to CU, validation status, and settlement rules.

1.2 Core Questions

First, how should models be selected? Different tasks require different models: frontier reasoning models for complex reasoning, fast execution models for structured batch jobs, open-weight private models for sensitive data, and specialist quantitative models for sequence forecasting and portfolio optimization.

Second, how should agents become trustworthy? Open agent networks require identity, reputation, validation, wallet control, and permission boundaries. ERC-8004 provides an important reference for identity, reputation, and validation registries, while LCEN adds workload accounting and strategy audit layers around that trust foundation.

Third, how should strategies be validated? Agent-generated strategies are not automatically effective. LCEN turns strategy production into reproducible experiments in which data versions, model versions, code versions, backtest parameters, risk limits, equity curves, and audit bundles can be verified.

1.3 Contributions

The first contribution is the LCEN Model Mesh, a model-selection and routing framework for combining frontier reasoning models, fast models, open-weight private models, specialist quant models, and embedding models. The second contribution is an ERC-8004-compatible Agent Passport that binds identity, model cards, skill cards, risk policy, training version, and audit state.

The third contribution is a Multi-Agent Strategy Factory that organizes Data Agent, Feature Agent, Research Agent, Strategy Agent, Risk Agent, Execution Agent, Audit Agent, and Portfolio Meta-Agent as a production DAG. The fourth contribution is the Workload Accounting Ledger, which records model calls, GPU time, backtest scale, oracle data, validation cost, and settlement evidence.

The fifth contribution is a reproducible simulation method including equity curves, drawdown curves, rolling Sharpe, fee drag, slippage sensitivity, and capacity curves. These charts explain methodology, risk exposure, and reproducibility rather than isolated performance claims.

2 Design Principles and Technical Boundary

Table 1. LCEN design principles

Principle	Technical implementation	On-chain/off-chain boundary
Off-chain intelligence	Inference, training, backtesting, risk control	On-chain records hashes and states
On-chain trust coordination	Identity, reputation, validation, settlement, commitments	Minimal trusted state
Reproducibility first	Model BOM, dataset manifest, strategy spec	Artifacts can be replayed
Least privilege	Policy engine, smart account, human approval	Agents do not hold raw private keys
Auditable experiments	Equity, drawdown, and cost are traceable	Assumptions and sample windows are labelled

2.1 Division of Labor

LCEN separates the off-chain intelligence layer from the on-chain trust coordination layer. The former handles high-complexity computation, while the latter handles public state, permissions, commitments, and settlement. This avoids both the cost of on-chain LLM inference and the unverifiability of opaque off-chain results.

The off-chain layer includes model services, agent runtime, RAG, vector databases, feature engineering, backtest engines, risk engines, simulation reports, and encrypted artifact storage. The on-chain layer includes agent identity, reputation, validation, task commitments, workload registries, settlement contracts, and event logs.

The boundary lets LCEN work across multiple model providers, agent frameworks, and EVM-compatible networks. LCEN does not require every computation to

be placed on-chain; it requires important computations to be indexed, traced, replayed, or verified.

2.2 Reproducibility

Reproducibility is a first-class technical goal. Each experiment should produce dataset_hash, model_bom_hash, strategy_spec_hash, risk_policy_hash, simulation_hash, and audit_root. These hashes do not expose raw data, but they prove that a result corresponds to a specific set of inputs and configurations.

A strategy experiment needs frozen data snapshots, feature-generation logic, model parameters, prompts, code versions, cost models, slippage models, and risk constraints. Where data cannot be public, the system stores access policy, encrypted fingerprints, and audit evidence.

Model upgrades do not move directly into the production strategy factory. Each upgrade runs regression evaluation for schema validity, tool success rate, latency, cost, hallucination rate, strategy consistency, and simulation metric drift.

2.3 Risk Boundary

LCEN is written in a technical and verifiable language. The strategy factory produces auditable research experiments, not a single performance narrative. Equity curves are used to explain simulation methodology, risk control, cost assumptions, and reproducibility.

Agents should not directly execute high-risk on-chain operations. Mint, upgrade, pause, treasury transfer, large settlement, validator management, and parameter changes require multisig, timelock, permission checks, and audit logs.

Different task classes should use different verification strengths. Low-risk tasks may rely on reputation feedback, while high-risk tasks should use validator re-execution, TEE attestation, zkML proof, or human review.

3 Model Selection: Model Mesh

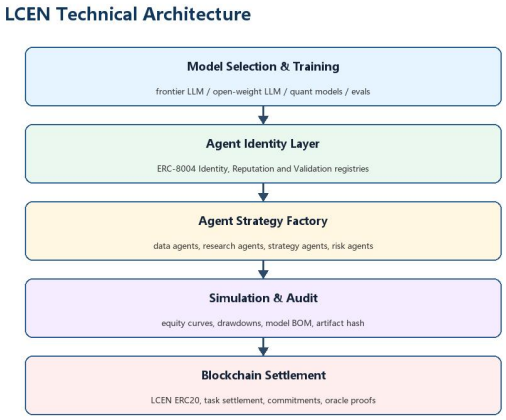

Figure 1. LCEN technical architecture

Table 2. Model Mesh selection matrix

Layer	Candidate models	Applicable tasks	Key evaluation
Frontier reasoning	GPT / Claude / Gemini class	Hypothesis generation, audit reasoning	Reasoning accuracy, long context, tool planning
Fast execution	Mini / Flash / Sonnet class	Batch extraction, low-latency structured jobs	Latency, cost, schema validity
Open-weight private	Llama / gpt-oss / local LLM	Private data, offline experiments, LoRA tuning	Deployment control, privacy, reproducibility
Specialist quant	PatchTST / TFT / PPO / SAC / FinRL	Time series, position sizing, execution optimization	Sharpe, MDD, turnover
Embedding / RAG	Finance and code embedding models	Retrieval, memory, attribution	recall@k, source coverage

3.1 Why a Single Model Is Not Enough

A quantitative strategy factory is not a single text-generation task. It includes data parsing, research reasoning, code generation, parameter search, risk diagnosis, execution simulation, and audit archiving. A single model cannot usually optimize reasoning quality, low latency, low cost, private deployment, long context, and specialist time-series capability at the same time.

LCEN uses Model Mesh. The core idea is model composition and model routing. Each task selects a model according to difficulty, privacy level, budget, latency, context length, structured-output requirement, and verification strength. The router records selected_model_id and routing_policy_hash inside the audit bundle.

This design also reduces model-provider lock-in. If a model becomes unavailable, the system can switch to a peer model or fallback model and use regression evaluation to confirm whether the output still satisfies strategy-factory requirements.

3.2 Frontier Reasoning Layer

Frontier reasoning models are used for complex strategy research and long-chain reasoning. When the system needs to synthesize macro events, on-chain flows, market structure, and textual evidence, a high-reasoning model can generate candidate research paths.

This layer is also useful for audit reporting and anomaly diagnosis. If a strategy suffers drawdown under a market regime, Research Agent and Audit Agent can use a frontier model to explain factor decay, cost increase, or regime shift.

To prevent frontier models from becoming unbounded black boxes, LCEN records a Model BOM for each call and requires Risk Agent and Audit Agent to validate outputs through schemas and tests. Complex models propose candidates; they do not decide final strategies alone.

3.3 Open-Weight and Specialist Quant Layers

Open-weight private models are used for private data and offline experiments. They can run inside controlled GPU clusters and can be adapted with LoRA or PEFT for financial research, strategy code, contract review, and risk explanation.

Specialist quant models handle numerical prediction and optimization tasks that generic LLMs are not designed to solve. PatchTST, Temporal Fusion Transformer, volatility models, PPO, SAC, and FinRL-compatible agents can support price-series modelling, position sizing, and execution optimization.

The LCEN principle is division of labor. LLMs handle research, generation, explanation, orchestration, and audit; specialist quant models handle forecasting, optimization, and risk estimation. Agent runtime and Model Router connect them.

4 Model Slicing, Training, and Evaluation Loop

Table 3. Model slicing mechanisms

Mechanism	Technical meaning	LCEN implementation
Task Slicing	Split by task type	Data / Research / Strategy / Risk / Audit
Context Slicing	Split by context window	Asset, market regime, time window, corpus
Adapter Slicing	Load adapters by domain	LoRA / PEFT adapters for finance, contracts, and risk
Model Routing	Select models by cost, privacy, and latency	routing_policy_hash
Inference Sharding	Parallel inference clusters	tensor/pipeline parallelism, KV cache

4.1 Definition of Model Slicing

Model slicing in LCEN does not mean physically splitting model weights. It is a set of engineering mechanisms. Task slicing divides work into data, research, strategy, risk, execution, and audit tasks. Context slicing organizes retrieved context by asset class, market regime, research topic, and time window. Adapter slicing loads different LoRA or PEFT adapters for different domains.

Model routing is the scheduler of the slicing system. It selects a model according to task risk, model cost, context length, privacy needs, latency requirements, and output format. The routing decision must be included in the audit bundle so that later reviewers can explain why a task used a specific model.

Inference sharding addresses deployment throughput. For self-hosted models, LCEN can use tensor parallelism, pipeline parallelism, KV-cache tiering, prefill/decode separation, and batch scheduling to support concurrent multi-agent workloads.

4.2 Three Stages of Agent Training

The first stage is supervised fine-tuning. The training objective is to make agents emit standardized outputs such as strategy_spec.json, risk_policy.json, simulation_report.json, and audit_bundle.json. Training samples come from historical strategy documents, research reports, contract ABIs, data dictionaries, and human-annotated task traces.

The second stage is preference and rule-reward training. The reward function does not optimize historical backtest appearance as the only target. It includes format compliance, tool success, citation completeness, risk-constraint compliance, refusal of unauthorized actions, low hallucination, and reproducibility. This prevents agents from producing attractive curves while ignoring risk and audit requirements.

The third stage is online evaluation and continuous calibration. The system records each agent's schema valid rate, tool success rate, latency, cost, risk violation rate, validation pass rate, and simulation drift. Low-quality agents receive lower routing priority, and severe violations can pause or revoke an agent.

4.3 Training Reward Function

LCEN can express agent training reward as a multi-objective function. Performance-related metrics are only one part; reproducibility, auditability, and risk compliance are more important. A reward for a strategy-generation agent can be defined as follows:

Reward = w1*SchemaScore + w2*ToolSuccess + w3*RiskCompliance + w4*Reproducibility + w5*ValidationPass - w6*Hallucination - w7*PolicyViolation.

This reward structure reflects LCEN's technical position: the strategy factory should prioritize verifiable strategy experiments rather than unreproducible high-performance text outputs.

5 ERC-8004-Compatible Agent Identity and Trust Layer

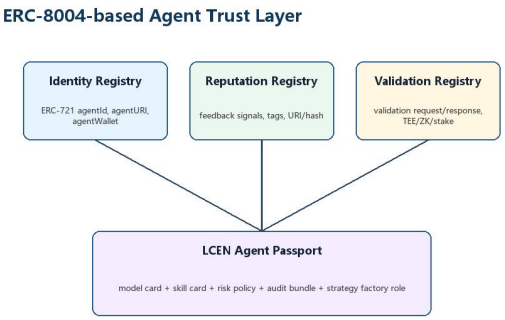

Figure 2. ERC-8004-based agent trust layer

Table 4. Mapping from ERC-8004 to LCEN

Module	ERC-8004 role	LCEN extension
Identity Registry	ERC-721 agentId + agentURI	Agent Passport, model card, skill card
Reputation Registry	Feedback signal, tag, URI/hash	Task success rate, risk violations, latency score
Validation Registry	Validation requests and responses	TEE, zkML, re-execution, validator committee
Agent Wallet	EIP-712 / ERC-1271 control proof	Smart account, budget, settlement policy

5.1 Positioning of ERC-8004

ERC-8004, Trustless Agents, proposes using blockchains to discover, select, and interact with agents across organizational boundaries without pre-existing trust. Its core consists of three registries: Identity Registry, Reputation Registry, and Validation Registry. Because the official EIP page marks it as Draft, LCEN should use ERC-8004-compatible or ERC-8004-inspired wording.

Identity Registry creates an agentId for an agent in an ERC-721-like form and uses agentURI to point to a registration file. That file can include agent name, description, service endpoint, A2A endpoint, MCP endpoint, DID, ENS, wallet, and supportedTrust.

Reputation Registry allows clients to submit feedback signals. Feedback can include tag, endpoint, feedbackURI, and feedbackHash. LCEN can aggregate task success rate, response time, risk violations, validation pass rate, and service availability into reputation scoring.

5.2 Validation Registry

Validation Registry allows agents to request independent validation and allows validator smart contracts to record responses. A validation request usually includes validatorAddress, agentId, requestURI, and requestHash. requestURI points to off-chain validation material, and requestHash is its commitment.

A validation response can be a score from 0 to 100, or it can include responseURI, responseHash, and tag. LCEN can use this mechanism for strategy re-execution, TEE attestation, zkML proof, risk-rule verification, and audit-report confirmation.

The validation layer should not serve only model outputs; it should also serve strategy-factory results. For example, after a Strategy Agent creates a candidate strategy, Validation Registry can record independent backtest results, code-review results, and whether risk limits were followed.

5.3 Agent Passport

LCEN defines Agent Passport on top of ERC-8004. Passport binds agentId, agentURI, agentWallet, modelCardHash, skillCardHash, riskPolicyHash, trainingVersionHash, auditBundleRoot, and settlementProfile.

Passport does not store full model weights, private strategies, or sensitive data. It stores verifiable indexes. The model card describes model family, version, adapter, and limits; the skill card describes callable tools; the risk policy describes task level, budget, human approval, and prohibited actions; the training version describes SFT/RL/eval data; and the audit root links to off-chain artifacts.

This design allows agents to be identified, invoked, evaluated, and validated across applications, organizations, and strategy factories. CU records task workload, and LCEN Token handles settlement, while ERC-8004 remains the identity and trust layer.

6 Agent Wallets, Permissions, and On-Chain Interaction

Table 5. Task lifecycle state machine

State	Meaning	On-chain/off-chain record
Created	Task created	taskId, requester, inputHash
Authorized	Authorization and budget locked	budget, riskLevel, policyHash
Accepted	Agent accepts task	agentId, skillId, modelHint
Executing	Off-chain execution	checkpoint, tool trace
Submitted	Result submitted	outputHash, artifactURI
Validated	Validation completed	validationHash, score
Settled	Settlement completed	LCEN amount, chainEventId
Disputed	Dispute or failure	reasonCode, evidenceURI

6.1 Agents Do Not Hold Raw Private Keys

AI agents must not directly control on-chain assets through raw private keys. Production environments should use smart accounts, MPC wallets, KMS, hardware security modules, or multisig custody. An agent submits intents or artifacts, and Policy Engine checks them before on-chain transactions are triggered.

High-risk operations require multiple layers of checks, including budget limits, task type, risk level, tool permissions, validation status, human confirmation, and timelock. Any mint, upgrade, pause, treasury transfer, validator management, or large settlement should not be triggered by an agent alone.

Agent Wallet can refer to ERC-4337 account abstraction ideas and support session keys, spending limits, time windows, permission scopes, and recovery policy. This preserves automation while reducing key and permission risk.

6.2 On-Chain Interaction Path

The standard LCEN interaction path is Agent output -> Audit Bundle -> Policy Engine -> Relayer -> Smart Contract. The agent first generates a structured result and audit bundle. Policy Engine checks schema, permissions, budget, and risk. A relayer or smart account signer then submits the transaction.

Minimal on-chain records include agentId, taskId, strategyId, artifactRoot, modelBomHash, datasetHash, simulationHash, validationRequestHash, workloadUnits, and settlementStatus. Raw data, strategy code, prompts, and transaction details remain off-chain.

On-chain events are synchronized by an indexer into the audit database and workload ledger. This allows key states to be restored through event replay even if off-chain services are upgraded.

6.3 Policy Engine

Policy Engine is the control layer between agent automation and on-chain safety. It reads task type, risk level, Agent Passport, Reputation Score, Validation Status, budget, and human approval rules to decide whether execution can continue.

Policy Engine can combine a rule engine with model review. Low-risk tasks can be automatically approved by rules; medium-risk tasks require Risk Agent review; high-risk tasks require human confirmation and timelock.

All rejections, downgrades, pauses, and human approvals must be written into the audit log. This explains why an agent was allowed or blocked during a dispute.

7 Multi-Agent Strategy Factory: Technical Principles

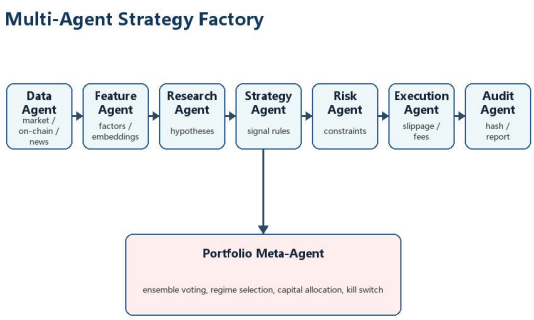

Figure 3. Multi-agent strategy factory DAG

Table 6. Strategy factory agent responsibilities

Agent	Input	Output
Data Agent	Market, on-chain, news, macro	dataset_manifest
Feature Agent	Cleaned data	factor_matrix
Research Agent	Documents, factors, events	hypothesis_set
Strategy Agent	Hypotheses, factors	strategy_spec
Risk Agent	Strategy and portfolio	risk_policy
Execution Agent	Order book and fees	execution_simulation
Audit Agent	All artifacts	audit_root
Portfolio Meta-Agent	Candidate strategies and risk matrix	allocation_plan

7.1 From Single Agent to Strategy Factory

A single agent can generate research text, but it is difficult for one agent to stably produce verifiable strategies. LCEN splits strategy production into a multi-agent DAG. Each agent owns a clearly bounded task and emits a structured artifact. This reduces hallucination, improves reproducibility, and makes error sources easier to locate.

Data Agent handles data ingestion and quality checks, Feature Agent generates factors, Research Agent generates hypotheses, Strategy Agent generates rules, Risk Agent constrains risk, Execution Agent simulates trading, and Audit Agent generates commitments. Portfolio Meta-Agent then selects and combines candidate strategies.

The strategy factory output is not a single buy/sell suggestion. It is a set of verifiable strategy experiment packages. Each package contains data version, model version, strategy specification, risk policy, execution simulation, equity curve, trade-detail hashes, and audit root.

7.2 Strategy Factory State Machine

The strategy factory can be defined as a state machine: Research -> Generate -> Backtest -> Stress Test -> Validate -> Commit -> Archive. Each state produces artifacts, and failures return the task to a previous state for repair.

For example, after Strategy Agent generates a strategy, Risk Agent may detect excessive max drawdown and return the task to Strategy Agent to adjust the signal formula or position rules. Execution Agent may find that returns disappear after slippage, forcing the system to reduce turnover or restrict trading windows.

This loop makes the strategy factory closer to an engineering system than one-shot text generation. It can record failures, repairs, retests, and version changes.

7.3 Portfolio Meta-Agent

Portfolio Meta-Agent does not directly generate base signals. It combines candidate strategies by reading strategy-performance correlation, risk contribution, market regime, transaction cost, and validation scores, then outputs allocation weights and rebalancing rules.

Meta-Agent can use mean-variance optimization, risk parity, CVaR optimization, regime switching, or ensemble voting. Its core objective is not to maximize one historical metric, but to preserve risk constraints and strategy diversity across market regimes.

Portfolio outputs need allocation_hash, risk_budget_hash, and rebalance_rule_hash. These hashes make the portfolio construction process reproducible.

8 Data Layer, RAG, and Feature Engineering

Table 7. Data sources and quality control

Data type	Examples	Quality checks
Market data	OHLCV, order book, trades	Missing values, duplicates, outliers, timezone
On-chain data	Active addresses, flows, DEX	Chain ID, confirmations, reorg risk
Derivatives data	Funding rate, open interest	Exchange consistency
Macro data	Rates, dollar index, risk index	Publication time and lag
News sentiment	News, announcements, social text	Source reliability and deduplication

8.1 Dataset Manifest

LCEN requires each dataset to generate dataset_manifest.json. This file contains source, license, time_range, asset_universe, row_count, missing_rate, schema_version, checksum, and preprocessing_code_hash.

Dataset cleaning must be reproducible. Outlier treatment, missing-value filling, adjustment, time alignment, exchange mapping, and on-chain address aggregation should all record their rules. If a data source has license restrictions, the manifest records access permissions and retention period.

Raw data is not stored on-chain; dataset_hash is stored instead. This allows an auditor, under authorization, to verify whether an equity curve truly comes from a specified data version.

8.2 Role of RAG in Strategy Research

RAG allows Research Agent to cite papers, strategy libraries, contract ABIs, risk rules, market events, and historical experiments. LCEN requires each retrieval to record retrieval_hash, top_k, source_uri, chunk_id, and citation map.

RAG should not be used merely to generate longer text. It should reduce hallucination and improve traceability. Each key hypothesis emitted by Research Agent should be traceable to a data source, document source, or historical experiment.

For high-value strategies, the RAG corpus should also be versioned. Otherwise, the same prompt can produce different conclusions under different knowledge-base states.

8.3 Feature Engineering

Feature Agent generates momentum, reversal, volatility, volume, liquidity, funding rate, on-chain activity, stablecoin flow, sentiment embeddings, and market-regime labels.

Feature engineering must prevent look-ahead bias. Data available only after the intraday close cannot be used for intraday signals; macro data should align by publication time rather than reference period; on-chain data should account for confirmation delay.

Each feature should record feature_name, lookback, lag, normalization, winsorization, source_hash, and code_hash. Feature Store should support replay by experiment version.

9 Strategy Generation, Risk Control, and Execution Simulation

Table 8. Strategy specification fields

Object	Required fields	Purpose
strategy_spec	universe, frequency, signal_formula	Strategy definition
risk_policy	max_position, max_drawdown, volatility_target	Risk constraints
fee_model	maker_fee, taker_fee, gas_cost	Fee simulation
slippage_model	spread, depth, impact	Execution simulation
benchmark	buy_hold, equal_weight, momentum_base	Result comparison

9.1 Strategy Specification

Strategy Agent output must be a structured strategy specification rather than natural-language advice. strategy_spec should at least include asset_universe, rebalance_frequency, lookback_window, signal_formula, entry_rule, exit_rule, position_sizing, and risk_limits.

Strategies may come from trend following, mean reversion, volatility breakout, multi-factor ranking, on-chain factors, sentiment factors, funding rates, market microstructure, and reinforcement-learning position management. Different strategy types should use different validation methods.

After generation, a strategy version must be frozen. Later modifications produce a new strategy_id and cannot overwrite older experiment results.

9.2 Risk Layer

Risk Agent applies layered constraints. Basic constraints include max single-asset position, max portfolio leverage, max drawdown, volatility target, daily loss limit, and correlation limit. Advanced constraints include market-regime filters, liquidity filters, event-risk filters, and kill switch.

Risk control is not decoration after backtesting; it is a hard constraint before strategy generation and during execution simulation. If a strategy works only by ignoring transaction costs or using extreme turnover, Risk Agent should downgrade or reject it.

The risk report should include max_drawdown, VaR, CVaR, volatility, turnover, exposure, correlation, liquidity_score, and stress_result.

9.3 Execution Simulation

Execution Agent simulates a realistic trading environment, including maker/taker fee, gas cost, spread, market impact, latency, partial fill, and failed order. For high-frequency strategies, execution simulation can be more important than the signal itself.

Execution simulation should output orders.csv, trades.csv, positions.csv, fee_report.json, and slippage_report.json. Full details remain off-chain, while hashes enter the audit bundle.

The LCEN simulation report shows both ideal-fill and real-cost equity curves to demonstrate strategy sensitivity to transaction costs.

10 On-Chain Commitments, Workload Accounting, and LCEN Token

Table 9. On-chain contract modules

Module	Responsibility	Core event
LCENToken	ERC-20-compatible settlement token	Transfer, Approval
AgentRegistryAdapter	Link ERC-8004 identity	AgentLinked
WorkloadRegistry	Record workload summary	WorkloadCommitted
StrategyCommitmentRegistry	Record strategy experiment	StrategyCommitted
SettlementContract	Task and validation settlement	TaskSettled
ValidationGateway	Link validator responses	ValidationRecorded
AccessControl/Timelock	Permissions and delayed execution	RoleGranted, CallScheduled

10.1 Technical Role of LCEN Token

LCEN Token is an ERC-20-compatible settlement token. It does not replace ERC-8004. ERC-8004 handles agent identity, reputation, and validation; LCEN architecture handles workload accounting and task verification. The system first uses Compute Unit (CU) to measure task workload, then uses LCEN Token for task settlement, validation fees, strategy-factory experiment fees, and audit-submission fees according to validation results, fee parameters, and settlement rules.

The technical value of LCEN architecture comes from auditable work rather than a single performance narrative. Model calls, GPU runtime, backtest scale, data access, validation cost, and on-chain submissions can all be mapped to CU and then converted into LCEN ledger events by settlement rules.

LCEN architecture should separate the internal ledger from on-chain LCEN Token settlement. High-frequency tasks can be aggregated in the internal ledger first and periodically settled on-chain to reduce gas cost.

10.2 Workload Formula

A workload unit can be represented as:

CU = alpha*T_in + beta*T_out + gamma*GPU_ms + delta*BacktestBars + eta*OracleRows + zeta*ValidationCost.

T_in and T_out represent model input text tokens and model output text tokens; GPU_ms is GPU runtime; BacktestBars is the scale of backtest data; OracleRows is the number of external data records; and ValidationCost is validator or proof cost. These text tokens are used only to estimate model computation and are not LCEN Token.

```
CU = alpha*T_in + beta*T_out + gamma*GPU_ms +
delta*BacktestBars + eta*OracleRows +
zeta*ValidationCost
```

10.3 Strategy Commitment Registry

StrategyCommitmentRegistry records strategyId, artifactRoot, datasetHash, modelBomHash, simulationHash, riskPolicyHash, artifactURI, and committer. The registry does not prove that a strategy will remain effective; it proves that a specific experiment

produced a specified result under a specific input and configuration.

WorkloadRegistry records agentId, taskId, modelIdHash, computeUnits, runtimeMs, resultHash, and createdAt. It supports cost attribution, quality analysis, dispute handling, and reputation update.

Together, these two registries form LCEN's auditable computation layer. Strategy results and workload consumption can be connected into a technical closed loop.

11 Strategy Simulation Methodology and Equity Curves

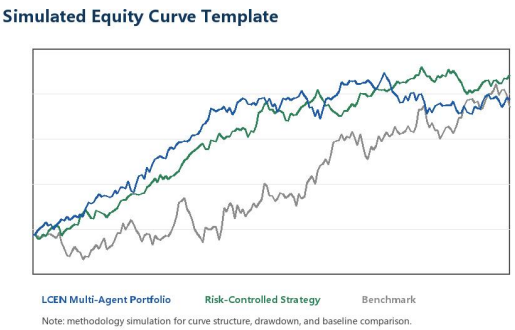

Figure 4. Simulated strategy equity curve

Table 10. Simulation metrics

Metric	English	Meaning
Equity curve	Equity Curve	Portfolio net value over time
Maximum drawdown	Max Drawdown	Peak-to-trough loss
Rolling Sharpe	Rolling Sharpe	Risk-adjusted stability
Sortino ratio	Sortino Ratio	Downside-risk adjustment
Turnover	Turnover	Trading frequency and cost pressure
Fee drag	Fee Drag	Fees, slippage, and gas impact
Capacity curve	Capacity Curve	Capital scale impact

11.1 Experiment Setting

The experiment chapter specifies data range, asset pool, trading frequency, training set, validation set, test set, walk-forward window, fees, slippage, latency, and failed-fill probability.

Baseline strategies should at least include buy-and-hold, equal-weight portfolio, simple momentum, and simple mean reversion. LCEN strategy factory must compare against baselines rather than showing a single curve.

All results are labelled as historical backtest or simulated trading. If methodological sample data is used, chart titles and notes should clearly mark the figure as a simulation illustration.

11.2 Ablation Experiments

Ablation experiments demonstrate the contribution of each agent module. Experiment A compares a single strategy with a multi-agent portfolio. Experiment B compares Risk Agent on and off. Experiment C compares ideal execution with real fee/slippage execution. Experiment D compares a single model with Model Router.

Each experiment outputs strategy performance, volatility, max drawdown, Sharpe, turnover, fee drag, and validation pass rate. More importantly, it explains why metrics change, such as whether risk control reduces drawdown or whether execution simulation exposes an untradable strategy.

Ablation experiments prevent the white paper from remaining only at the architecture level. They show how system design affects reproducible engineering metrics.

11.3 Equity Curve Interpretation

Equity curves should not be shown in isolation. They must be linked to trade details, positions, fees, drawdown, and risk state. LCEN recommends that each equity curve correspond to equity_curve_hash, trades_hash, positions_hash, and metrics_hash.

The drawdown curve is equally important. Strategies with high historical performance but excessive drawdown may not be suitable for a portfolio. Risk Agent filters strategies by max drawdown, tail loss, and market regime.

This white paper presents multi-agent portfolio curves, risk-control curves, and benchmark curves together with sample window, data source, transaction cost, slippage assumption, and validation status.

12 Verification Layer: TEE, zkML, and Re-execution

Table 11. Comparison of verification methods

Method	Applicable scenario	Advantages	Limits
Re-execution	Backtests, rules, code outputs	Simple and reproducible	Higher compute cost
TEE	Private data, model inference	Privacy-preserving, environment attestation	Depends on hardware trust
zkML	Small models, metric computation	Strong cryptographic proof	Cost and model-size limits
Validator committee	High-value strategy review	Flexible governance	Requires incentives and penalties
Human audit	High-risk tasks	Strong semantic judgment	Slow

12.1 Verification Is Not On-Chain Inference

LCEN's verification layer does not require full large-model inference to be placed on-chain. The chain

records only validation requests, validation responses, and evidence hashes. Actual verification can happen through off-chain re-execution, TEE, zkML, or human audit.

For backtest tasks, the most realistic validation method is re-execution. Validators read the same dataset manifest, strategy spec, and code hash, rerun the experiment in an independent environment, and compare simulation_hash.

For private data and strategy code, TEE can provide execution-environment attestation. For small models or rule computation, zkML or zkVM can provide stronger cryptographic proof.

12.2 Validation Request

A validation request should include agentId, taskId, validatorAddress, requestURI, requestHash, proofType, and deadline. requestURI points to off-chain material, and requestHash is the material commitment.

A validation response should include responseScore, responseHash, responseURI, tag, validatorSignature, and timestamp. responseScore may be a 0-100 score or a pass/fail result.

Validation results enter the agent reputation system. Repeated failures or violations reduce agent routing priority, and severe cases can pause the Agent Passport.

12.3 Validator Incentives

Validators need incentives and penalties. LCEN can design validator bonds, challenge windows, error penalties, and validation rewards. If a validator submits a false response, other validators can challenge it.

Different task values require different verification strength. Low-value tasks can rely on reputation, while high-value tasks require multiple validators, TEE, or zk proof.

The validator network should not be controlled by a single institution. Open validators improve credibility but require sybil and collusion defenses.

13 Audit Artifacts and Content Addressing

Table 12. Audit Bundle file structure

File	Content	Hash purpose
dataset_manifest.json	Data source, window, checksum	datasetHash
model_bom.json	Model, version, parameters, tools	modelBomHash
strategy_spec.json	Signals, parameters, frequency	strategyHash
risk_policy.json	Risk limits and thresholds	riskPolicyHash
execution_simulation.json	Fees, slippage, fills	executionHash
simulation_report.json	Equity curves and metrics	simulationHash
validation_report.json	Validation results	validationHash
artifact_manifest.json	Index of all files	artifactRoot

13.1 Why Artifacts Are Needed

A blockchain cannot and should not store complete strategy-experiment materials. LCEN uses off-chain artifacts combined with on-chain commitments. Off-chain artifacts store complete data and reports; on-chain records store hashes, roots, and URIs.

Content-addressed storage can use IPFS, Filecoin, or object storage with hash indexes. For sensitive strategies, artifacts should be encrypted and access permissions controlled by an off-chain authorization system.

Audit Bundle is the foundation for dispute resolution and experiment reproduction. An on-chain hash without artifacts has little audit value.

13.2 Merkle Root

Multiple artifact files can form a Merkle tree. The hash of each file is a leaf node, and the final artifactRoot is written into StrategyCommitmentRegistry.

When an auditor only needs to verify a specific file, a Merkle proof can be used without downloading the entire bundle. This balances integrity and efficiency.

If an artifact is updated, a new version and a new root must be generated. Older roots cannot be overwritten, and older experiments should remain immutable.

13.3 Data Retention and Access Control

LCEN should define artifact retention periods. Low-value tasks may be stored for a shorter time, while high-value strategies and validation results should be preserved for the long term. Third-party data must follow license terms.

Access control should support role, task, time, and purpose restrictions. Auditors, validators, agent developers, and public users see different materials.

On-chain URIs should not leak sensitive information. Encrypted CIDs, access gateways, or zero-knowledge indexes can be used.

14 Blockchain Privacy Layer and Confidential Verification

Table 13. Privacy-preserving technology matrix

Private object	Technical mechanism	On-chain verifiable object
User task input	Encrypted task package, access gateway, minimal disclosure	taskHash, requestHash
Strategy code and parameters	Encrypted Audit Bundle, Merkle proof	strategyHash, artifactRoot
Model-call context	Prompt hash, Model BOM, TEE attestation	modelBomHash, attestationHash
Datasets and features	Dataset commitment, authorized verification	datasetHash, featureHash
Workload	CU commitment,	workloadRoot

accounting	range proof, batch reconciliation	
Validation material	Selective disclosure, validator permission, zk proof	validationHash
Settlement detail	Public settlement event plus private task evidence	settlementEvent, evidenceRoot

14.1 Privacy Design Goals

The privacy goal of LCEN is not to hide all on-chain activity. It is to protect strategy intellectual property, user task data, agent tool traces, model-call context, validation materials, and settlement details while preserving verifiability and auditability. The principle is privacy-preserving, not audit-avoiding.

A public chain is suitable for state, hashes, events, and minimal settlement information. It is not suitable for directly storing prompts, strategy code, trade details, private datasets, or full model outputs. LCEN therefore uses a privacy architecture based on public commitments, encrypted evidence, and authorized disclosure.

Different roles receive different visibility. Public readers can see system state, validation results, and public metrics; validators can read encrypted evidence under authorization; auditors can inspect full artifacts; developers can access only materials related to their own agents or tasks.

14.2 Commitment-Based On-Chain Privacy

LCEN uses commitment mechanisms to reduce on-chain data exposure. The chain records dataset_hash, model_bom_hash, strategy_hash, risk_policy_hash, simulation_hash, workload_root, and artifact_root, while raw data, strategy code, prompts, trade details, and model outputs remain off-chain.

Merkle roots compress files, trade details, model-call logs, and validation materials into a single verifiable root. When an auditor needs to verify one file, a Merkle proof is enough; the full experiment package does not need to be disclosed.

Workload accounting can also use commitments. The system may publish CU summaries and settlement state while keeping specific model context, private task length, GPU execution details, and strategy R&D intensity in an encrypted ledger that reconciles with on-chain events through batch roots.

14.3 Encrypted Audit Bundle and Selective Disclosure

Audit Bundle can use encrypted CIDs, KMS/MPC key management, and role-based access control. The chain stores only artifactRoot, evidenceRoot, access-policy hash, and required events. Full materials are read through authorized gateways.

Selective disclosure allows one artifact to expose different views to different roles. Public users see experiment summaries, curves, and validation status; validators see reproducible materials; auditors see full data manifests, model BOMs, strategy specs, execution simulations, and settlement evidence.

Access records should also enter audit logs, including requester, purpose, scope, expires_at, approved_by, and access_hash. This proves who accessed sensitive material and for what purpose.

14.4 Zero-Knowledge Verification and Confidential Computing

Zero-knowledge proofs can show that a strategy experiment satisfies constraints without revealing the full strategy or dataset. Examples include proving that max drawdown stays below a threshold, risk rules were executed, a backtest used a specified dataset commitment, workload accounting was not inflated, or a validation result came from a given artifactRoot.

TEE can support confidential execution for private data and private strategies. Model inference, feature computation, and strategy backtesting can run inside a trusted execution environment. The chain stores attestationHash, resultHash, and environment version so that reviewers can verify that computation occurred inside a controlled environment.

zkML still has cost and scale limits for full large-model inference. In the near term, it is more suitable for small models, rule verification, metric computation, and workload proofs. For large-model inference, TEE attestation, hash commitments, and validator re-execution are more practical engineering paths.

14.5 Confidential Settlement and Compliance Boundary

When LCEN Token functions as the settlement asset, the task text, strategy code, and model context do not need to be public. The chain can publish settlementEvent, amount, epoch, workloadRoot, and validationHash, while task details remain protected through encrypted evidenceRoot and authorized disclosure.

The privacy layer should not be designed as a tool for avoiding audit or compliance. High-value tasks, disputed tasks, and regulatory or audit-authorized scenarios should be able to reconstruct the evidence chain under permission control and prove consistency between task, workload, validation, and settlement.

LCEN's privacy design is therefore confidential but accountable: data and strategy details are confidential by default, critical states and validation relationships are publicly inspectable, and required evidence can be selectively disclosed under authorization.

15 Security and Risk Control

Table 14. Risk and control matrix

Risk	Control	Audit evidence
Contract permission	Multisig, Timelock, RBAC, Pause	role events
Agent overreach	Policy Engine, Tool Sandbox	permission log
Model hallucination	RAG source, schema validator	eval report
Prompt injection	Input isolation, tool allowlist	security trace
Data poisoning	source allowlist, checksum	quality report
Strategy overfitting	walk-forward, out-of-sample	experiment report
Privacy leakage	hash on-chain, encrypted storage	access log
Validator collusion	multi-validator, challenge period, penalty	challenge record

15.1 Smart Contract Security

The contract layer should use mature ERC-20, AccessControl, Pausable, and Timelock patterns. Key roles include DEFAULT_ADMIN, MINTER, PAUSER, VALIDATOR_ADMIN, SETTLEMENT_OPERATOR, and UPGRADER.

Upgrades must pass through timelock and multisig. Contract events should cover role grants, role revocations, parameter changes, pauses, resumes, settlements, and validation responses.

Contract audits should cover reentrancy, permission bypass, integer precision, missing events, upgrade storage collision, and signature replay.

15.2 Agent Security

Agent security includes prompt injection, tool misuse, unauthorized calls, malicious model outputs, and supply-chain attacks. LCEN should use allowlists, parameter schemas, sandboxing, and least privilege for tool calls.

MCP and A2A endpoints require authentication, rate limiting, audit logs, and input isolation. External tool output cannot directly enter high-risk decisions; it must pass through Risk Agent or Policy Engine.

Agent outputs must be structurally validated. Natural-language explanations cannot replace JSON schema, hash, signature, and validation response.

15.3 Strategy Risk

Quantitative strategy risks include look-ahead bias, data leakage, overfitting, insufficient capacity, underestimated transaction cost, and market-regime drift. LCEN's simulation framework must explicitly include these risks in experiment design.

Before production, a strategy should pass out-of-sample testing, walk-forward testing, cost stress, slippage stress, and extreme market-regime testing. Unverified strategies remain research artifacts only.

Equity curves must be shown together with drawdown and fee drag. Showing only an upward curve does not meet the standard of a technical white paper.

16 Engineering Implementation Architecture

Table 15. Reference technology stack

Layer	Component	Description
AI Runtime	LangGraph / AutoGen / Agents SDK	Agent orchestration and state
Model Serving	API providers / vLLM / TGI	Model service and local inference
Data	PostgreSQL / DuckDB / Object Storage	Structured and batch data
Vector	pgvector / Milvus / Qdrant	RAG and memory
Quant	Python / pandas / vectorbt / backtrader	Backtesting and metrics
Blockchain	Solidity / Foundry / OpenZeppelin	Contract development
Indexing	Subgraph / custom indexer	On-chain event sync
Security	KMS / MPC / Multisig	Keys and signatures

16.1 Service Decomposition

LCEN can be divided into Model Service, Agent Runtime, Data Service, Quant Engine, Audit Service, Blockchain Service, Ledger Service, and Monitoring Service.

Model Service manages model calls and local inference; Agent Runtime manages workflows; Data Service manages data versions; Quant Engine runs backtests; Audit Service generates artifacts; Blockchain Service submits commitments; Ledger Service handles workload and settlement; Monitoring Service monitors quality and risk.

Services should be linked through task_id and artifact_id rather than implicit state.

16.2 Database Design Direction

Core tables include agents, agent_passports, tasks, model_calls, datasets, features, strategies, backtests, risk_reports, audit_bundles, workload_entries, settlements, and chain_events.

All key tables should include created_at, updated_at, version, hash, status, and operator. Reconciliation tables should support ledger-state replay from on-chain events.

Strategy and model results should not be overwritten directly. Versioning and append-only records should be used.

16.3 Monitoring Metrics

System monitoring includes model latency, model cost, tool failure rate, schema pass rate, agent success rate, validation pass rate, on-chain transaction failure rate, ledger difference, backtest duration, and artifact upload success.

Strategy monitoring includes max drawdown, volatility, turnover, fee drag, signal decay, rising correlation, and market-regime drift.

Security monitoring includes abnormal permission calls, failed signatures, suspicious endpoints, prompt-injection alerts, and abnormal validator behavior.

17 Phased Roadmap

Table 16. LCEN technical roadmap

Phase	Core delivery	Acceptance criteria
P0 Specification	Schemas, interfaces, threat model	Reviewable technical specification
P1 Contract MVP	ERC-20, registries, commitments	Testnet deployment
P2 Agent Runtime	Model Router, Agent SDK, Policy Engine	End-to-end task execution
P3 Strategy Factory	Backtest, risk, audit bundle	Reproducible experiment
P4 Validation Network	Validator, TEE/zk proof PoC	Validation result on-chain
P5 Production	Monitoring, security, SDK, reconciliation	Security audit and stress testing

17.1 P0-P1

P0 completes the technical specification, including Agent Passport schema, Model BOM schema, Strategy Commitment schema, Workload Ledger schema, Task Lifecycle, and contract ABI.

P1 completes the minimum contract implementation, including LCEN ERC-20, AgentRegistryAdapter, WorkloadRegistry, StrategyCommitmentRegistry, SettlementContract, ValidationGateway, and AccessControl.

P1 acceptance criteria include local chain and testnet deployment, unit tests, permission tests, and event indexing.

17.2 P2-P3

P2 completes Agent Runtime, including Data Agent, Research Agent, Strategy Agent, Risk Agent, Execution Agent, Audit Agent, Model Router, Skill Registry, and Policy Engine.

P3 completes the strategy-factory MVP. The system should generate strategy specifications from data input, run backtests, generate equity curves, package audit bundles, and submit commitments to a testnet.

P3 acceptance requires at least three reproducible strategy experiments that can generate complete artifacts and on-chain commitments.

17.3 P4-P5

P4 completes the validation-layer proof of concept, including validator re-execution, TEE attestation, or zkML/zkVM small-model proof. Validation results should enter Validation Registry or a compatible module.

P5 completes production hardening, including key management, monitoring alerts, ledger reconciliation, security audit, stress testing, developer SDK, and documentation.

Before production, LCEN must complete threat modeling, contract audit, agent red-team testing, data-license review, and simulation reproducibility review.

18 Developer Interfaces, SDKs, and Protocol Objects

Table 17. LCEN core API objects

Object	Key fields	Purpose
AgentPassport	agentId, modelCardHash, skillCardHash	Agent identity and capability index
TaskRequest	taskId, requester, inputHash, budget	Task creation and authorization
ModelBOM	modelId, adapterHash, promptHash	Model-call reproduction
StrategySpec	universe, signalFormula, riskLimits	Strategy definition
AuditBundle	artifactRoot, cid, fileHashes	Audit archive
ValidationRequest	validator, requestURI, requestHash	Independent validation
WorkloadEntry	computeUnits, runtimeMs, resultHash	Workload accounting

18.1 SDK Layers

LCEN SDK should be divided into Agent SDK, Quant SDK, Chain SDK, Audit SDK, and Validation SDK. Agent SDK defines agents, skills, model routing, and task state; Quant SDK reads data, generates features, runs backtests, and emits metrics; Chain SDK submits commitments, reads registries, and listens to events; Audit SDK generates artifacts, Merkle roots, and CIDs; Validation SDK initiates and responds to validation requests.

The goal of the SDK is not to hide complexity, but to organize complex workflows through standard objects. If each team defines task, strategy, model, validation, and settlement formats independently, agents cannot collaborate and auditors cannot reproduce results.

The SDK should support TypeScript, Python, and Solidity ABI layers. TypeScript serves frontends and agent services; Python serves quant and model experiments; Solidity ABI serves on-chain registration and settlement.

18.2 API Endpoint Design

LCEN APIs can be divided into five categories: identity endpoints for Agent Passport, model cards, skill cards, and reputation; task endpoints for creating, authorizing, submitting, and cancelling tasks; strategy endpoints for registering strategies, querying experiments, and downloading reports; validation endpoints for initiating validation, submitting responses, and querying status;

and workload endpoints for workload, fee, and settlement status.

Each API response should include request_id, timestamp, schema_version, and hash. For data that enters on-chain commitments, the API should return canonical JSON to ensure different language clients compute the same hash.

High-risk APIs require signatures. Users, agents, or servers can use EIP-712 typed data to sign task authorization, budget locking, validation requests, and settlement confirmations.

18.3 Canonical JSON and Hashing Rules

LCEN must define canonical JSON rules, including field ordering, null handling, numeric precision, time format, encoding, and array order. Otherwise, the same object can serialize differently in different languages and produce different hashes.

Amounts and quantities should use integers or fixed-point numbers rather than floating point values before hashing. Time should use ISO-8601 or Unix timestamp, with timezone explicitly defined in schema.

Strategy formulas, risk rules, and model parameters should also serialize to a stable format. For code files, source hash, dependency lockfile hash, and runtime environment hash should be recorded.

19 Performance, Capacity, and Cost Evaluation

Table 18. Performance and capacity metrics

Dimension	Metric	Description
Agent Runtime	tasks/min, success rate	Task throughput and success rate
Model Serving	text tokens/s, p95 latency	Model inference performance
Quant Engine	bars/s, backtests/hour	Backtest throughput
Storage	artifact upload latency	Audit bundle archiving
Blockchain	tx success rate, gas per commit	On-chain submission cost
Validation	validation latency, pass rate	Validation efficiency

19.1 Agent Runtime Capacity

The bottleneck of a strategy factory is usually not a single agent, but the concurrent scheduling of a multi-agent DAG. A complete strategy experiment may include data download, feature calculation, research generation, strategy coding, backtesting, stress testing, audit packaging, and on-chain submission. LCEN should manage these steps with queues, state machines, and retry mechanisms.

Capacity evaluation should report p50, p95, and p99 latency. Low-risk tasks can run asynchronously, while high-risk tasks require human confirmation or validator responses and should be measured separately.

Agent runtime needs checkpoints. If a long task is interrupted, the system should resume from the latest checkpoint instead of repeating all model calls and backtest computations.

19.2 Model Inference Cost

Model cost consists of model input text tokens, model output text tokens, context length, reasoning intensity, tool calls, and retries. Frontier models are used for a small number of high-value reasoning tasks; fast models are used for bulk structured work; open-weight models are used for private data and high-frequency offline tasks.

Model Router should balance quality and cost. For example, a Data Agent formatting task should not call the most expensive model, while a high-value Audit Agent report can call a frontier model but must be checked by schema validator.

Cost optimization includes prompt caching, batch processing, context compression, embedding cache, adapter reuse, and local-model fallback.

19.3 On-Chain Cost Control

LCEN should not write every intermediate step on-chain. The chain only stores final audit root, workload summary, validation result, and settlement event. Large intermediate logs remain off-chain and are committed through artifactRoot.

For frequent small tasks, batch submission can be used. Multiple task commitments can form a Merkle tree and periodically submit the root. When a user or validator needs to inspect an individual task, a Merkle proof is provided.

On-chain cost evaluation should include gas per strategy commitment, gas per workload batch, gas per validation response, and gas per settlement. Benchmarks should be run on the target network before production deployment.

20 Threat Model and Red-Team Testing

Table 19. Threat model

Attack surface	Attack method	Defense
Prompt	Prompt injection, data exfiltration	Input isolation, tool allowlist
Tool	Malicious API, command injection	Sandbox, schema, permission
Model	Hallucination, overreach reasoning	Evaluation, RAG citation, audit
Data	Poisoning, fabrication, delay	Multi-source validation, checksum
Agent	Sybil, reputation farming	ERC-8004 reputation filtering, validators
Chain	Permission abuse, signature replay	Multisig, timelock, nonce
Strategy	Overfitting, look-ahead bias	walk-forward, leakage detection

20.1 Prompt and Tool Attacks

The most common risk in agent systems is prompt injection. External webpages, news, API responses, or documents may contain instructions that induce agents to leak secrets, bypass rules, or call dangerous tools. LCEN should isolate external content from system instructions and prohibit external content from escalating privileges.

Tool calls must use allowlists and parameter schemas. Even if a model generates a tool-call request, runtime must check tool_id, parameter range, budget, target endpoint, and risk level.

For code-execution tools, sandboxing, resource limits, and filesystem isolation are required. Strategy code may run backtests, but it must not access secrets, wallets, system directories, or unauthorized data sources.

20.2 Reputation and Validation Attacks

ERC-8004-like reputation systems may suffer sybil attacks, reputation farming, and colluding validators. LCEN needs to weight reputation by task type, validator credibility, time decay, and economic cost rather than using a simple average.

Validators can also behave maliciously. The system should support multiple validators, challenge periods, deposits, penalties, and secondary validation. High-value tasks should not depend on a single validator.

When an agent migrates or transfers ownership, reputation inheritance requires careful design. If agentId is transferred through ERC-721, the new owner should not automatically inherit unconditional trust.

20.3 Red-Team Testing Process

Before launch, LCEN should run agent red-team tests. Test cases include prompt injection, malicious documents, forged data, abnormal market data, API failure, tool timeout, contract rejection, signature error, and backtest look-ahead bias.

Red-team outputs include attack_case, expected_block, actual_behavior, risk_level, fix_status, and regression_test. Each fix should enter the continuous test suite.

Red-team testing is not only a security-team task. It should also cover quant teams and model teams because many risks come from strategy logic and model behavior rather than traditional contract vulnerabilities.

21 Standards Compatibility, Interoperability, and Ecosystem Interfaces

Table 20. Standards compatibility matrix

Standard / protocol	LCEN usage	Boundary
ERC-20	LCEN Token	Settlement asset interface
ERC-721	Agent identity	Basis of ERC-8004 Identity Registry
ERC-8004	Agent identity, reputation, validation	Draft; adapter layer
ERC-4337	Agent Wallet / smart account	Permission and budget control
EIP-712	Structured signature	Task authorization and wallet proof
ERC-1271	Contract wallet signature validation	Agent Wallet control proof
MCP	Tool and data connection	Does not provide on-chain trust
A2A	Agent-to-agent communication	Does not provide settlement or validation

21.1 MCP and A2A

MCP addresses how models connect to tools, data sources, and workflows. LCEN can expose market queries, on-chain data, backtest engines, audit services, and contract ABI readers as MCP tools. MCP tools must pass through permission and sandbox controls.

A2A addresses communication and collaboration between agents. Research Agent, Strategy Agent, Risk Agent, and Audit Agent in the strategy factory can exchange tasks, context, and artifacts through A2A-style protocols.

MCP and A2A themselves do not provide on-chain identity, reputation, or validation. LCEN uses the ERC-8004-compatible layer to fill that gap.

21.2 ERC-8004 Compatibility Strategy

Because ERC-8004 is still Draft, LCEN should build an adapter layer. If standard interfaces change, the adapter can be updated while core Agent Passport, Workload Ledger, and Strategy Commitment objects remain stable.

LCEN should not embed payment logic into the ERC-8004 core. ERC-8004 also treats payment as an orthogonal issue. LCEN Token as an independent settlement asset better matches modular design.

Agents can register on multiple chains and receive feedback on different chains. LCEN should use chain_id, registry_address, and agent_id together to determine global identity.

21.3 Developer Ecosystem Interfaces

LCEN can provide five interface classes for developers: agent registration, skill publishing, strategy experiment, validation request, and workload settlement.

Third-party developers can register specialized agents such as on-chain data agents, news sentiment agents, risk agents, validation agents, or reporting agents. Each agent describes its capabilities, risks, and validation methods through Passport.

The key to interoperability is schema stability. LCEN should publish JSON Schema, ABI, SDK, example artifacts, and test vectors so that different teams can implement compatible services independently.

22 Protocol State Consistency, Ledger Reconciliation, and Failure Recovery

Table 21. State consistency checks

Object	Consistency rule	Exception handling
Task	Off-chain task status matches on-chain event	Event replay, state repair
Agent	Passport, reputation, validation state match	Re-index, freeze agent
Strategy	artifactRoot matches off-chain files	Mark invalid, resubmit
Workload	computeUnits match model/backtest logs	Ledger adjustment, audit record
Settlement	Internal balance matches on-chain events	Pause withdrawals, manual review
Validation	requestHash matches responseHash	Challenge, secondary validation

22.1 Why State Consistency Matters

LCEN includes off-chain agent runtime, off-chain workload ledger, off-chain artifact storage, and on-chain contract events. Delay, failure, or retry at any layer can create inconsistent state. For example, an off-chain task may complete while the on-chain commitment transaction fails, or an on-chain event may succeed while the indexer has not synchronized it.

State consistency is not an ordinary backend issue; it is central to LCEN auditability. If task status, artifactRoot, workload entry, and settlement event cannot be matched, auditors cannot reproduce strategy-factory outputs or judge whether agent workload actually occurred.

LCEN should therefore adopt event-sourcing ideas. On-chain events and key off-chain events are written into append-only logs, and system state is derived by replaying events rather than relying on a single mutable field.

22.2 Reconciliation Between Internal Ledger and On-Chain Events

The workload ledger should use a double-entry structure. Each workload entry has debit, credit, source_type, source_id, computeUnits, amount, status, artifactRoot, and chainEventId. If on-chain settlement fails, ledger state remains pending or failed rather than being deleted.

Reconciliation jobs should periodically compare internal ledger records with on-chain events. Checks include mint batch, task settlement, validation fee, workload commitment, strategy commitment, and agent update. Any difference should generate a reconciliation_report.

For chain reorganization, transaction failure, nonce conflict, or relayer interruption, the system should support idempotent retry. Each on-chain submission must have an idempotency_key to avoid duplicate settlement.

22.3 Failure Recovery

LCEN should define failure recovery levels. Minor failures include model API timeout, tool-call failure, and artifact-upload failure. Medium failures include interrupted backtests, validator non-response, and pending on-chain transactions. Severe failures include key leakage, contract pause, ledger inconsistency, and data poisoning.

Minor failures can be retried automatically. Medium failures enter a retry queue and human observation. Severe failures must trigger emergency pause, freeze affected agents, stop settlement, and start an audit.

The failure recovery report should include incident_id, affected_tasks, affected_agents, root_cause, repair_action, chain_events, ledger_diff, and prevention_plan. The report can also be stored as an audit artifact.

23 Testing System, Audit Checklist, and Acceptance Criteria

Table 22. LCEN test matrix

Test type	Coverage	Acceptance criteria
Unit Test	Contracts, ledger, schema	Core function coverage
Integration Test	Agent -> Audit -> Chain	End-to-end submission succeeds
Replay Test	Data, model, strategy	Results reproducible
Security Test	Permission, signature, tools	No high-risk vulnerability
Red Team	Prompt, data, agent	Attack blocked or downgraded
Load Test	Concurrent tasks and backtests	Throughput target met
Reconciliation Test	Ledger and on-chain events	Explained difference equals zero

23.1 Contract Tests

Contract tests cover LCENToken, AgentRegistryAdapter, WorkloadRegistry, StrategyCommitmentRegistry, SettlementContract, ValidationGateway, and AccessControl. Each contract should include permission tests, event tests, exception paths, and boundary values.

For WorkloadRegistry and StrategyCommitmentRegistry, tests must cover duplicate submissions, empty hashes, wrong agentId, unauthorized committer, paused state, storage compatibility after upgrade, and event-index integrity.

For SettlementContract, tests must cover duplicate claim, wrong signature, expired authorization, insufficient balance, paused withdrawal, batch settlement, and on-chain event replay.

23.2 Agent and Model Tests

Agent tests should not only evaluate answer quality. LCEN tests schema pass rate, tool success rate, refusal of unauthorized actions, citation accuracy, RAG source coverage, model routing correctness, and risk-policy compliance.

Model upgrade tests must use a fixed eval suite. The suite includes data-cleaning tasks, strategy-generation tasks, risk-explanation tasks, contract-reading tasks, backtest-repair tasks, and audit-report tasks. Each upgrade outputs metric drift.

If a new model improves reasoning quality but lowers schema pass rate, the system should not directly replace the production model. LCEN prioritizes stability over one-time performance.

23.3 Strategy Experiment Acceptance

A strategy experiment can enter a formal report only if its data manifest is complete, model BOM is complete, strategy specification is parseable, risk policy is executable, backtest has no look-ahead bias, transaction costs are included, equity curve is reproducible, audit bundle is downloadable, and on-chain commitment is queryable.

The acceptance report should include experiment_id, strategy_id, dataset_hash, model_bom_hash, simulation_hash, validation_status, risk_flags, and reviewer_signature.

Strategies that fail acceptance cannot enter the formal group of Strategy Commitment Registry and remain research artifacts only.

23.4 Public Verification Materials

LCEN public verification materials include the technical white paper, contract interface documents, Agent Passport schema, Model BOM schema, Strategy Spec schema, Workload Ledger schema, API documentation, test reports, threat model, audit checklist, and experiment reproduction package.

Public charts use real historical backtests or are explicitly marked as methodological simulation illustrations. Every chart labels data source, sample window, transaction cost, slippage assumption, and limitations.

The audit checklist is retained as engineering audit material to describe the validation state of contracts, AI agents, model routing, quantitative simulation, and on-chain settlement modules.

24 Technical Boundaries, Reader Notes, and Conclusion

24.1 Technical Limits

LCEN cannot eliminate large-model hallucination; it can only reduce risk through RAG, schemas, evaluation, validation, and audit. LCEN also cannot guarantee that a strategy remains effective in the future; it can improve reproducibility and auditability of strategy experiments.

ERC-8004 is currently Draft, and its interfaces and ecosystem may change. LCEN should therefore preserve an adapter-layer design rather than locking the core system to one version.

zkML still has cost and scale limits for full large-model inference. In the short term, the more realistic path is hash commitments, validator re-execution, and TEE attestation.

24.2 Reader Notes

Equity curves, backtest metrics, and simulation charts should be read together with sample window, data source, transaction cost, slippage assumption, and validation status. Readers should focus on whether a strategy is reproducible, independently validated, and governed by explicit risk thresholds.

Any strategy-factory output should pass independent validation, risk review, and compliance evaluation. Real markets involve insufficient liquidity, extreme volatility, exchange risk, chain congestion, model failure, and data latency.

Agent automation must be constrained by permissions, budgets, human confirmation, and emergency pause mechanisms.

24.3 Conclusion

LCEN should be positioned as blockchain-native AI agent infrastructure and a technical network. Its core is not a single application rule, but model selection, verifiable training, agent identity, strategy factories, simulation experiments, and on-chain workload settlement. CU measures workload, while LCEN Token acts as the settlement asset in the network.

Through ERC-8004-compatible Agent Passport, Model Mesh, Strategy Factory DAG, Workload Accounting Ledger, and Commitment Registry, LCEN can provide a technical trust layer for an open agent economy.

This white paper keeps an engineering, paper-style, and verifiable expression so that readers can understand how the system is implemented, validated, audited, and deployed in phases.

25 References

De Rossi M., Crapis D., Ellis J., Reppel E. ERC-8004: Trustless Agents [DRAFT]. Ethereum Improvement Proposals, 2025. https://eips.ethereum.org/EIPS/eip-8004

OpenAI. API Models Documentation. https://developers.openai.com/api/docs/models

Anthropic. Claude Models Overview. https://platform.claude.com/docs/en/about-claude/models/overview

Google. Gemini API Models. https://ai.google.dev/gemini-api/docs/models

Model Context Protocol. Introduction. https://modelcontextprotocol.io/docs/getting-started/intro

A2A Project. Agent2Agent Protocol. https://github.com/a2aproject/A2A

Ethereum. ERC-20 Token Standard. https://eips.ethereum.org/EIPS/eip-20

Ethereum. ERC-4337 Account Abstraction. https://eips.ethereum.org/EIPS/eip-4337

OpenZeppelin Contracts. AccessControl Documentation. https://docs.openzeppelin.com/contracts/5.x/access-control

IPFS. Content Addressing. https://docs.ipfs.tech/concepts/content-addressing/

Vaswani et al. Attention Is All You Need. arXiv:1706.03762.

Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401.

Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.

Yao et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601.

Schulman et al. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Liu et al. FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading. arXiv:2011.09607.

Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.