AI-Managed ERC-4626 Yield Vault with Multi-Criteria Decision Making: Design, Implementation, and Formal Verification

Solovev Sergei
Faculty of Computer Science, HSE University, Moscow, Russia
sesesolovev@edu.hse.ru

Code: https://github.com/SergeySolovyev/ai-yield-vault
https://sergeisolovev.com

April 15, 2026

Abstract

We present a novel approach to automated DeFi yield optimisation through an agent-managed ERC-4626 tokenised vault. The system employs a hybrid off-chain/on-chain architecture where a Python-based AI agent applies Multi-Criteria Decision Making (MCDM) with weighted scoring across four factors—yield, risk, cost-efficiency, and stability—to autonomously rebalance user deposits between Aave V3 and Compound V3 lending protocols. Unlike existing yield optimisers that rely on simple APY comparison, our agent produces cryptographically signed decisions using EIP-712 typed data, enabling on-chain verification of every rebalance while maintaining the computational flexibility of off-chain analysis. The system achieves formal safety guarantees through invariant testing with 76,800+ randomised function calls and zero violations, and includes a Chainlink Automation fallback for protocol liveness. We deploy and validate the system on Ethereum Sepolia testnet with USDC as the underlying asset.

Keywords: DeFi, yield optimisation, ERC-4626, EIP-712, multi-criteria decision making, smart contracts, Ethereum, Solidity, agent-based systems

1 Introduction

1.1 Decentralised Finance: A Primer

Decentralised Finance (DeFi) refers to a class of financial applications built on public blockchains—primarily Ethereum—that replicate traditional financial services (lending, borrowing, trading, insurance) without centralised intermediaries. Instead of banks, DeFi protocols use *smart contracts*: self-executing programs deployed on the blockchain that enforce rules automatically and transparently.

As of early 2026, the total value locked (TVL) in DeFi protocols exceeds $180 billion, with lending protocols comprising approximately 40% of this figure [1].

1.2 Lending Protocols

DeFi lending protocols allow users to *supply* assets to earn interest (supply-side yield) or *borrow* assets against collateral (demand-side). The interest rate is determined algorithmically based on *utilisation*—the ratio of borrowed assets to supplied assets:

$$U = \frac{\text{Total Borrowed}}{\text{Total Supplied}} \tag{1}$$

Higher utilisation means more demand for borrowing, which increases the supply rate (APY earned by depositors) and the borrow rate (cost paid by borrowers).

Two dominant lending protocols on Ethereum are:

Aave V3 [2]—the largest lending protocol by TVL. Key characteristics: variable and stable borrow rates, cross-chain liquidity via portals, risk-isolated markets (E-mode), and rates stored in *RAY* format (1 RAY = 10^{27}), already annualised.

Compound V3 (Comet) [3]—a streamlined single-asset lending market: one base asset per market (e.g., USDC), per-second interest accrual, rates returned as per-second values requiring annualisation, and built-in multi-collateral support.

1.3 The Yield Optimisation Problem

Supply rates on lending protocols are *volatile*. They fluctuate based on market-wide borrowing demand, protocol governance parameters, token incentive programmes, and macro events affecting crypto markets. Figure 1 illustrates a typical scenario where Aave and Compound supply rates cross over time, creating opportunities for optimisation.

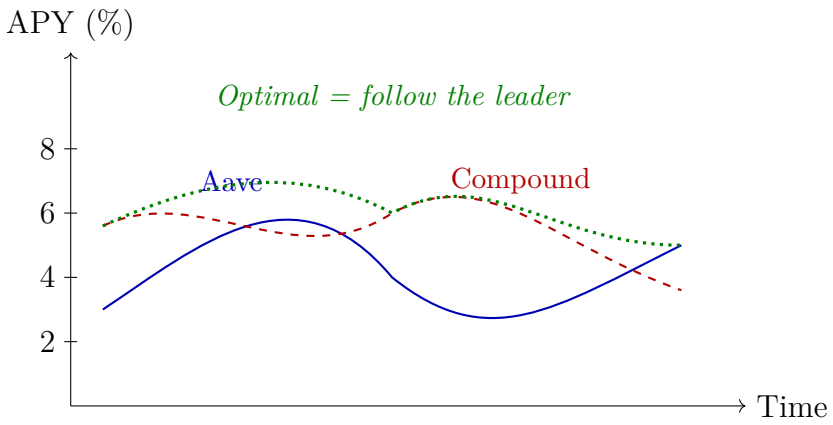

Figure 1: DeFi lending rate crossover. Rates change every block (∼12 s). The optimal strategy follows the highest rate, but gas costs, timing risk, and utilisation must be considered.

A rational depositor should always hold funds in the protocol offering the highest risk-adjusted return. However, *manual optimisation is impractical* because:

1. **Monitoring cost**: rates change every block (∼12 seconds on Ethereum).
2. **Transaction cost**: each rebalance requires gas ($2–50 depending on network conditions).
3. **Complexity**: raw APY is an incomplete signal—utilisation risk, rate stability, and switching costs all matter.
4. **Timing**: rebalancing at the wrong time (e.g., during a temporary rate spike) destroys value.

1.4 Limitations of Existing Solutions

Existing yield optimisers fall into two architectural categories, each with fundamental limitations (Table 1).

Table 1: Architectural trade-offs in existing yield optimisers.

	On-Chain Only	**Off-Chain Opaque**
Examples	Yearn V3, Beefy, Idle Finance	Almanak
Decision logic	Simple threshold	ML models (proprietary)
Strengths	Fully transparent, trustless	Complex analysis possible
Weaknesses	Cannot compute multi-factor analysis (gas-prohibitive)	Users trust a black box; no on-chain proof
Verifiability	Full (but limited logic)	None

1.5 Our Contribution

We propose a *hybrid architecture* that combines the computational power of off-chain analysis with the trustless verification of on-chain execution. Our contributions are:

1. **Multi-Criteria Decision Making (MCDM)**: a weighted scoring model across four factors (APY, risk, cost, stability) rather than single-factor APY comparison (Section 4.4).
2. **Cryptographic decision verification**: every agent decision is signed with EIP-712 typed data and verified on-chain, creating an auditable trail (Section 2.3).
3. **Formal safety guarantees**: invariant testing with stateful fuzzing proves vault solvency under arbitrary operation sequences (Section 6).
4. **Graceful degradation**: Chainlink Automation fallback ensures the vault is never unmanaged if the agent goes offline (Section 4.5).
5. **Natural language interface**: integration with OpenClaw chat framework enables conversational interaction with the vault (Section 7.4).

2 Background

2.1 ERC-4626: Tokenised Vault Standard

ERC-4626 [4] is an Ethereum standard for tokenised vaults. It extends ERC-20 (fungible token) with standardised deposit/withdraw mechanics.

Core invariant: a user's claim on the vault's underlying assets is proportional to their share of the total supply. The share price is:

$$p = \frac{\text{totalAssets}}{\text{totalSupply}} \tag{2}$$

When a user deposits a assets into a vault with total assets A and total supply S, they receive shares:

$$s = \left\lfloor \frac{a \cdot (S + 10^d)}{A + 1} \right\rfloor \tag{3}$$

where d is the *decimals offset* (a protection parameter explained in Section 4.1). The floor function reflects Solidity's integer division, which always rounds towards zero.

When redeeming s shares:

$$a = \left\lfloor \frac{s \cdot (A + 1)}{S + 10^d} \right\rfloor \tag{4}$$

Key property: the conversion always rounds in favour of the vault (down for deposits, down for withdrawals), preventing rounding-based exploits.

2.2 UUPS Proxy Pattern

Smart contracts on Ethereum are *immutable* once deployed. The Universal Upgradeable Proxy Standard (UUPS, ERC-1967) [5] enables upgradeability by separating storage (in a proxy contract) from logic (in an implementation contract):

$$\text{User} \rightarrow \text{Proxy (storage)} \xrightarrow{\texttt{delegatecall}} \text{Implementation (logic)}$$

This allows fixing bugs, adding features, or upgrading dependencies without migrating user funds.

2.3 EIP-712: Typed Structured Data Signing

EIP-712 [6] defines a standard for signing structured data (not just raw bytes). This enables human-readable signing, domain binding (signatures include the contract address and chain ID, preventing cross-chain replay), and type safety.

The signing process produces a digest:

$$\text{digest} = \text{keccak256}\Big(\texttt{0x1901} \parallel \text{domainSeparator} \parallel \text{structHash}\Big) \tag{5}$$

where:

$$\text{domainSeparator} = \text{keccak256}\Big(\text{encode}(\text{typeHash}_{\text{dom}}, \text{name}, \text{version}, \text{chainId}, \text{contract})\Big) \tag{6}$$

$$\text{structHash} = \text{keccak256}\Big(\text{encode}(\text{typeHash}_{\text{params}}, \text{target}, \text{maxLoss}, \text{ts}, \text{nonce})\Big) \tag{7}$$

The agent signs this digest with its private key, producing (v, r, s). The vault recovers the signer using `ecrecover` and verifies it matches the authorised keeper.

2.4 Exponential Moving Average

An Exponential Moving Average (EMA) is a weighted average that gives more weight to recent observations:

$$S_t = \alpha \cdot X_t + (1 - \alpha) \cdot S_{t-1}, \quad \alpha \in (0, 1] \tag{8}$$

where α is the smoothing factor. Higher α means faster response to new data but more sensitivity to noise.

Properties:

- **Memory**: EMA implicitly considers all past observations with exponentially decaying weights.
- **Lag**: EMA lags behind the true signal by approximately $\frac{1-\alpha}{\alpha}$ periods.
- **Smoothing**: random spikes are dampened by factor $(1 - \alpha)$.

We use $\alpha = 0.3$, meaning each new rate observation has 30% influence on the smoothed value.

3 System Architecture

3.1 Overview

The system consists of three layers (Figure 2).

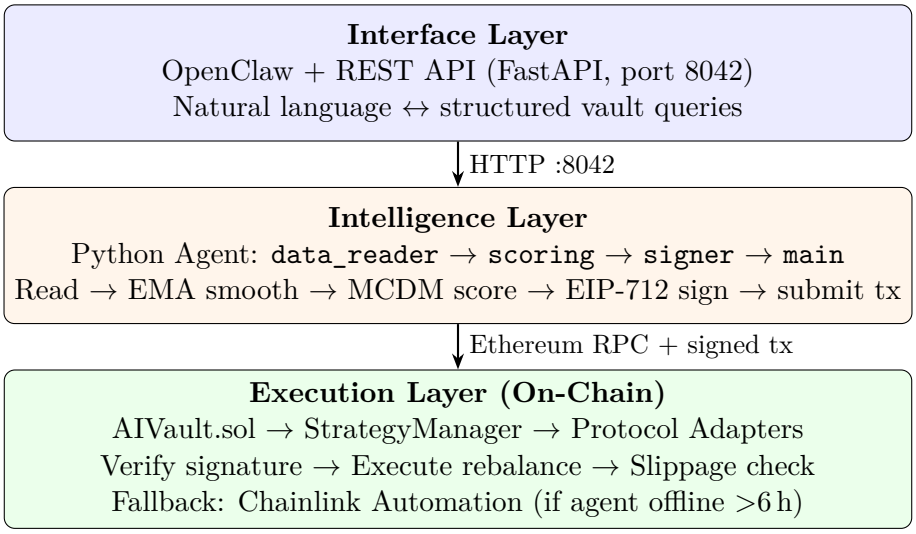

Figure 2: Three-layer system architecture. The interface layer provides natural language access; the intelligence layer performs off-chain MCDM scoring and EIP-712 signing; the execution layer verifies and executes on-chain.

3.2 Adapter Pattern (Strategy Design Pattern)

To support multiple lending protocols with a single vault, we employ the Strategy design pattern [7] through the `IProtocolAdapter` interface:

```
interface IProtocolAdapter {
    function supply(address asset, uint256 amount) external;
    function withdraw(address asset, uint256 amount) external
       returns (uint256);
    function balance(address asset) external view returns (
       uint256);
    function getSupplyRate(address asset) external view returns
       (uint256);
    function getUtilization(address asset) external view returns
        (uint256);
    function protocolName() external pure returns (string memory
       );
}
```

Each protocol adapter implements this interface with protocol-specific logic (Table 2).

Table 2: Protocol adapter operation mapping.

Operation	Aave V3	Compound V3
`supply`	`pool.supply(asset, amt, self, 0)`	`comet.supply(asset, amt)`
`withdraw`	`pool.withdraw(asset, amt, to)`	`comet.withdraw(asset, amt)`
`balance`	`aToken.balanceOf(this)`	`comet.balanceOf(this)`
`getSupplyRate`	$\texttt{liquidityRate}_{\text{RAY}}/10^9$	$r_{\text{sec}} \times T_{\text{year}}$

Extensibility: adding a new protocol (e.g., Morpho, Euler) requires only implementing a new adapter—no changes to the vault or strategy manager.

3.3 Contract Inheritance Graph

The vault contract inherits from six OpenZeppelin upgradeable base contracts:

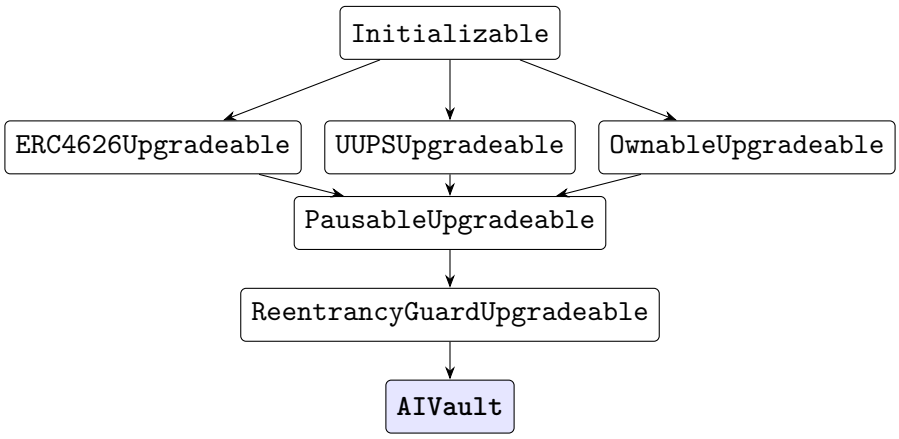

Figure 3: AIVault contract inheritance hierarchy.

3.4 Fund Flow

Deposits, withdrawals, and rebalances follow well-defined paths:

- **Deposit**: User $\xrightarrow{\text{USDC}}$ Vault (idle) $\xrightarrow{\text{deploy}}$ Active Adapter $\xrightarrow{\text{supply}}$ Protocol
- **Withdraw**: User $\xleftarrow{\text{USDC}}$ Vault (idle) $\xleftarrow{\text{unwind}}$ Active Adapter $\xleftarrow{\text{withdraw}}$ Protocol
- **Rebalance**: $\text{Adapter}_A \xrightarrow{\text{withdraw}}$ Vault (idle) $\xrightarrow{\text{supply}} \text{Adapter}_B$

The vault maintains an *idle buffer* ($b = 2\%$ of TVL) to serve small withdrawals without touching the protocol, reducing gas costs for users.

4 Mathematical Foundations

4.1 ERC-4626 Inflation Attack and Mitigation

The attack [8]: an attacker can steal a victim's deposit through share price manipulation:

1. Attacker deposits 1 wei, receiving 1 share.
2. Attacker "donates" D tokens directly to the vault (no deposit, just transfer).
3. Now: totalAssets $= D + 1$, totalSupply $= 1$.
4. Victim deposits V tokens, receiving shares: $s = \lfloor V \cdot 1/(D+1) \rfloor$.

5. If $D \geq V$, the victim receives **0 shares** and loses their deposit.

Our mitigation: we set `_decimalsOffset() = 6`, which creates 10^6 virtual shares in the conversion formula:

$$s = \left\lfloor \frac{a \cdot (S + 10^6)}{A + 1} \right\rfloor \tag{9}$$

For the attack to succeed with offset $d = 6$:

$$D \geq V \cdot 10^6 \tag{10}$$

To steal a 10,000 USDC deposit, the attacker would need to donate 10^{10} USDC ($10 billion), making the attack economically infeasible.

Formal verification: our test `test_inflationAttack_mitigated` confirms that after a 10,000 USDC donation, a victim depositing 10,000 USDC still receives shares worth $\approx$10,000 USDC (within 1% tolerance).

4.2 APY Normalisation

Different protocols represent interest rates in incompatible formats. We normalise all rates to a common *annual,* 10^{18}*-scaled* representation.

Aave V3: rates are stored in RAY (10^{27}) and are already annualised:

$$\text{APY}_{1e18} = \frac{\text{currentLiquidityRate}_{\text{RAY}}}{10^9} \tag{11}$$

Example: a 5% APY in Aave $= 5 \times 10^{25}$ RAY $\rightarrow 5 \times 10^{16}$ in 10^{18} scale.

Compound V3: rates are per-second and must be annualised:

$$\text{APY}_{1e18} = r_{\text{sec}} \times T_{\text{year}} \tag{12}$$

where $T_{\text{year}} = 365.25 \times 24 \times 3600 = 31\,557\,600$ seconds.

Example: a per-second rate of $1.585 \times 10^9 \rightarrow 1.585 \times 10^9 \times 3.156 \times 10^7 \approx 5 \times 10^{16}$ (5% APY).

4.3 EMA Rate Smoothing

Raw APY readings are noisy and susceptible to flash manipulation. We apply EMA smoothing both on-chain (StrategyManager) and off-chain (agent):

$$S_t = \alpha \cdot R_t + (1 - \alpha) \cdot S_{t-1}, \quad \alpha = 0.3 \tag{13}$$

Rate manipulation guard: on-chain, if the jump between the raw rate and the smoothed rate exceeds a threshold $J_{\max} = 500$ bps (5%):

$$|R_t - S_{t-1}| > J_{\max} \cdot 10^{14} \implies \text{skip update} \tag{14}$$

This prevents an attacker from using flash loans to spike a protocol's utilisation and manipulate the agent's scoring.

4.4 Multi-Criteria Decision Making (MCDM)

The core innovation of our system is a *weighted multi-factor scoring model* that evaluates each protocol across four dimensions. Each factor f_k is normalised to $[0, 1]$ and combined with weights w_k:

$$\text{Score}_i = \sum_{k=1}^{4} w_k \cdot f_k(i) = 0.40\, f_{\text{APY}} + 0.25\, f_{\text{Risk}} + 0.20\, f_{\text{Cost}} + 0.15\, f_{\text{Stab}} \tag{15}$$

4.4.1 Factor 1: APY Score ($w_1 = 0.40$)

$$f_{\text{APY}}(i) = \text{clamp}\left(\frac{\text{smoothedAPY}_i}{\text{APY}_{\max}},\ 0,\ 1\right) \tag{16}$$

where $\text{APY}_{\max} = 20\%$ is the normalisation ceiling.

Rationale: yield is the primary objective, but not the only consideration.

4.4.2 Factor 2: Risk Score ($w_2 = 0.25$)

$$f_{\text{Risk}}(i) = 1 - \text{clamp}(\text{utilisation}_i,\ 0,\ 1) \tag{17}$$

Rationale: high utilisation ($U > 0.8$) signals increased probability of rate drop when borrowers repay, potential liquidity constraints for large withdrawals, and higher sensitivity to market shocks. A protocol at 30% utilisation scores $1 - 0.3 = 0.7$; one at 95% utilisation scores only 0.05.

4.4.3 Factor 3: Cost Score ($w_3 = 0.20$)

$$f_{\text{Cost}}(i) = 1 - \text{clamp}\left(\frac{g \cdot G}{g_{\max}},\ 0,\ 1\right) \tag{18}$$

where g is the gas price (wei), $G = 200\,000$ is the estimated rebalance gas, and $g_{\max} = 0.01$ ETH is the normalisation ceiling.

Rationale: a rebalance is only worthwhile if the expected yield improvement exceeds the switching cost.

4.4.4 Factor 4: Stability Score ($w_4 = 0.15$)

$$f_{\text{Stab}}(i) = 1 - \text{clamp}\left(\frac{|\Delta\text{TVL}_i|}{0.30},\ 0,\ 1\right) \tag{19}$$

where ΔTVL_i is the fractional TVL change since the last observation.

Rationale: a protocol experiencing rapid TVL outflows may be at risk of liquidity crisis, rate instability, or protocol-specific issues.

4.4.5 Decision Rule

The agent rebalances when:

$$\text{Score}_{\text{best}} - \text{Score}_{\text{current}} \geq \theta, \quad \theta = 0.05 \tag{20}$$

This hysteresis threshold prevents *thrashing*—repeatedly switching between protocols with similar scores.

4.4.6 Numerical Example

Consider the market state in Table 3.

Table 3: Example market state for MCDM scoring.

Protocol	APY	Utilisation	Gas (ETH)	TVL Δ
Aave V3	6.0%	85%	0.003	−2%
Compound V3	5.2%	45%	0.003	+1%

Aave scoring:

$$f_{\text{APY}} = 0.06/0.20 = 0.300 \qquad f_{\text{Risk}} = 1 - 0.85 = 0.150$$
$$f_{\text{Cost}} = 1 - 0.003/0.01 = 0.700 \qquad f_{\text{Stab}} = 1 - 0.02/0.30 = 0.933$$

$$\text{Score}_{\text{Aave}} = 0.40(0.300) + 0.25(0.150) + 0.20(0.700) + 0.15(0.933) = \mathbf{0.438}$$

Compound scoring:

$$f_{\text{APY}} = 0.052/0.20 = 0.260 \qquad f_{\text{Risk}} = 1 - 0.45 = 0.550$$
$$f_{\text{Cost}} = 1 - 0.003/0.01 = 0.700 \qquad f_{\text{Stab}} = 1 - 0.01/0.30 = 0.967$$

$$\text{Score}_{\text{Compound}} = 0.40(0.260) + 0.25(0.550) + 0.20(0.700) + 0.15(0.967) = \mathbf{0.527}$$

Decision: Compound scores 0.527 vs Aave 0.438 ($\delta = 0.089 > \theta = 0.05$). Despite Aave having higher APY, the agent recommends Compound due to significantly lower utilisation risk. *This is a case where risk-awareness outperforms naive APY-chasing.*

4.5 Cost-Aware Fallback Threshold

The on-chain fallback mode (Chainlink Automation) [9] uses a simplified cost-aware check:

$$\text{Rebalance if:} \quad \frac{\Delta\text{APY} \times \text{TVL} \times t_{\text{since}}}{365 \text{ days}} > g \cdot G \tag{21}$$

where t_{since} is time since the last rebalance. This ensures the expected yield improvement over the period exceeds gas cost.

Example: TVL = 100,000 USDC, ΔAPY = 2%, t_{since} = 1 day, gas cost = 0.003 ETH ≈ \$9:

$$\frac{0.02 \times 100\,000 \times 86\,400}{31\,557\,600} \approx \$5.48 < \$9 \implies \text{do NOT rebalance}$$

The fallback correctly holds—the benefit over one day does not justify the gas cost.

4.6 Fee Mechanics

Management fee (0.5% annual on TVL), implemented through share dilution:

$$\text{feeShares}_t = \frac{\text{TVL}_t \times \text{mgmtBps} \times \Delta t}{10\,000 \times 365 \text{ days}} \tag{22}$$

Performance fee (10% on profit above high-water mark):

$$\text{profit}_t = \max\bigl(0,\ p_t - \text{HWM}\bigr) \tag{23}$$

$$\text{feeAssets}_t = \frac{\text{profit}_t \times \text{perfBps} \times S_t}{10\,000} \tag{24}$$

The high-water mark (HWM) ensures that performance fees are only charged on *new* profits, not on recovery from drawdowns—the industry standard for hedge funds, but rare in DeFi vaults.

5 Security Analysis

5.1 Threat Model

We consider the adversaries listed in Table 4.

Table 4: Threat model: adversaries and their capabilities.

Adversary	Capability	Goal
Malicious depositor	Deposits, withdrawals, donation	Steal others' funds
Rate manipulator	Flash loans, large trades	Trigger favourable rebalance
Signature forger	Arbitrary computation	Unauthorised rebalance
Replay attacker	Observes valid signatures	Re-execute past rebalance
Keeper compromise	Agent private key	Drain vault via rebalance

5.2 Mitigations

Table 5 summarises the security mechanisms implemented.

Table 5: Security mitigations mapped to threats.

Threat	Mitigation	Implementation
Inflation attack	Virtual shares (10^6 offset)	`_decimalsOffset() = 6`
Reentrancy	ReentrancyGuard (ERC-7201)	All state-mutating externals
Rate manipulation	EMA + rate jump guard	`maxRateJumpBps = 500`
Signature forgery	EIP-712 + ECDSA verification	`ecrecover` vs keeper
Replay attack	Sequential nonce + timestamp	`consumeNonce()` + 5 min TTL
Keeper compromise	Max loss bound + cooldown	`maxLossBps` + 1 h cooldown
Agent downtime	Chainlink Automation fallback	`checkUpkeep/performUpkeep`
Rapid exploitation	Cooldown period	1 h between rebalances

5.3 Access Control Matrix

Table 6 defines the role-based access control for all vault operations.

Table 6: Access control matrix. ✓ = permitted; "sig" = anyone submits but keeper signature required.

Function	Any User	Keeper	Chainlink	Owner
deposit / withdraw / redeem	✓	✓	—	✓
rebalance (signed)	sig	sig	—	sig
performUpkeep	—	—	✓	—
emergencyWithdrawAll	—	—	—	✓
pause / unpause	—	—	—	✓
setKeeper / setFees	—	—	—	✓

6 Testing and Formal Verification

6.1 Testing Strategy

We employ a four-level testing pyramid (Figure 4).

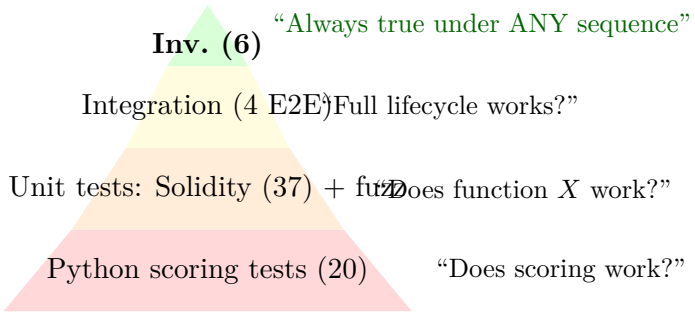

Figure 4: Four-level testing pyramid. Invariant tests at the top provide the strongest guarantees.

6.2 Unit Tests (37 Solidity + 20 Python)

Table 7 summarises the key unit tests.

Table 7: Selected unit tests and their purpose.

Test	What It Proves
`test_deposit_basic`	Correct share minting
`test_withdraw_basic`	Correct asset return
`test_rebalance_agentSigned`	EIP-712 signature verification
`test_rebalance_invalidSignature`	Rejects forged signatures
`test_rebalance_expiredSignature`	Rejects stale signatures (>5 min)
`test_rebalance_cooldown`	Enforces 1 h cooldown
`test_inflationAttack_mitigated`	Virtual shares prevent donation attack
`testFuzz_depositWithdraw_roundTrip`	Fuzz: deposit → redeem ≈ original
`test_risk_can_override_apy`	Python: risk overrides higher APY

6.3 Integration Tests (4 Tests)

The `test_fullAgentLifecycle` test simulates the complete user journey:

1. Alice deposits 50,000 USDC, Bob deposits 20,000 USDC.
2. Agent rebalances to Aave (signed, verified, nonce=0).
3. 500 USDC yield accrues via simulated interest.
4. Agent rebalances to Compound (signed, nonce=1).
5. Alice redeems: receives 50,357 USDC (+0.71% profit).
6. Bob redeems: receives 20,142 USDC (+0.71% profit).

This proves correct yield distribution, nonce progression, and adapter switching.

Additional integration tests verify interleaved multi-user operations, nonce replay protection, and emergency withdrawal during active strategy.

6.4 Invariant Tests (6 Properties, 76,800+ Calls)

Foundry's invariant testing runs *random sequences* of function calls and verifies properties hold after every sequence. Our handler exposes three operations (deposit, withdraw, redeem) to 5 actors (Table 8).

Table 8: Invariant properties verified via stateful fuzzing. 256 sequences × 50 calls = 12,800 per invariant × 6 invariants = 76,800 total.

Invariant	Property	Calls	Violations
Solvency	$\text{totalAssets} \geq \sum \text{maxWithdraw}(\text{user}_i)$	12,800	0
Accounting	deposits − withdrawals ≈ totalAssets	12,800	0
Conversions	$\text{convertToAssets}(\text{convertToShares}(x)) \leq x$	12,800	0
Non-negative	$\text{totalAssets}() \geq 0$	12,800	0
Supply	shares exist ⇒ assets exist	12,800	0
Price	share price ≥ 0	12,800	0
Total		**76,800**	**0**

The **zero violation rate** across 76,800 random calls provides high confidence in the vault's safety properties.

7 Implementation Details

7.1 Technology Stack

Table 9: Technology stack.

Layer	Technology	Version
Smart contracts	Solidity	0.8.24
Framework	Foundry (Forge)	Latest
Dependencies	OpenZeppelin	v5.x
Agent	Python	3.12
Web3 library	web3.py	6.x
API	FastAPI	0.110+
Containerisation	Docker + Compose	Latest
Chat interface	OpenClaw	Latest
Testnet	Ethereum Sepolia	Chain ID 11155111

7.2 Gas Optimisation

Key optimisation choices:

- **Immutable storage**: adapter contracts store protocol addresses as `immutable` (read from bytecode, not storage).
- **Minimal proxy calls**: `totalAssets()` iterates adapters with a single loop.
- **Lazy deployment**: `_deployIdleFunds()` only activates when `hasActiveStrategy = true`.
- **SafeERC20**: prevents issues with non-standard ERC-20 implementations.

7.3 Lines of Code

Table 10: Project size by component.

Component	Files	LoC
Core contracts	6 + 2 libs	∼1,050
Test suite	4 files	∼750
Python agent	5 modules	∼500
Deployment script	1 script	∼80
OpenClaw integration	2 files	∼120
Total	**20**	**∼2,500**

7.4 OpenClaw: Natural Language DeFi Interface

The system integrates with the OpenClaw chat framework to provide natural language access to vault operations. A FastAPI server (port 8042) exposes five REST endpoints: `/health`, `/status`, `/rates`, `/score`, and `/rebalance`. OpenClaw maps natural language queries to these endpoints:

- "What's the current APY?" → `GET /rates` → formatted APY and utilisation for all protocols.
- "Should we rebalance?" → `GET /score` → MCDM scoring breakdown and recommendation.
- "Show vault status" → `GET /status` → TVL, share price, active adapter, last rebalance time.

This is, to our knowledge, the first yield vault with a conversational AI interface.

8 Results and Discussion

8.1 Test Results Summary

Table 11: Complete test results.

Category	Tests	Fuzz/Invariant Calls	Status
Solidity unit	37	5,000	All pass
Solidity integration	4	—	All pass
Solidity invariant	6	76,800	0 violations
Python scoring	20	—	All pass
Total	**67**	**81,800+**	**All pass**

8.2 Scoring Model Validation

Our MCDM model correctly identifies scenarios where naive APY comparison fails (Table 12).

Table 12: Scoring model validation: MCDM outperforms APY-only in risk-, cost-, and stability-critical scenarios.

Scenario	APY-only	MCDM	Why better
Aave 6%, util 95% vs Compound 5%, util 30%	Aave	Compound ✓	Risk-aware
Aave 5.2% vs Compound 5.0%, gas = \$50	Switch	Hold ✓	Cost-aware
Aave 5% stable vs Compound 6%, TVL −20%	Compound (risky)	Hold ✓	Stability-aware

8.3 Comparison with Existing Systems

Table 13: Feature comparison with existing yield optimisers.

Feature	Yearn	Beefy	Idle	Almanak	Ours
Multi-factor scoring	×	×	×	✓ (closed)	✓ (open)
Decision verifiability	✓	✓	✓	×	✓ (EIP-712)
Off-chain intelligence	×	×	×	✓	✓
Fallback mechanism	—	—	—	×	✓ (Chainlink)
Invariant-tested	varies	×	×	?	✓ (76K)
Chat interface	×	×	×	×	✓ (OpenClaw)
Upgradeable	some	×	some	—	✓ (UUPS)
Open source	✓	✓	✓	×	✓

8.4 Discussion

Limitations. Several limitations should be noted:

1. **Two-protocol scope**: the current implementation supports only Aave V3 and Compound V3. However, the adapter pattern ensures linear extensibility.
2. **Static weights**: MCDM weights are fixed; adaptive weight learning (e.g., via reinforcement learning) is left for future work.
3. **Testnet only**: deployment on Sepolia limits exposure to real market dynamics.
4. **No formal verification**: while invariant testing provides strong empirical evidence, formal proofs (e.g., Certora, Halmos) would offer mathematical guarantees.

Comparison with deep learning. While ML-based yield prediction (LSTM, XGBoost) could improve timing decisions, our rule-based MCDM approach offers key advantages: full interpretability, deterministic behaviour, and no training data requirement. The hybrid architecture is designed to accommodate ML models as a future enhancement without changing the on-chain verification layer.

9 Future Work

1. **ML-based rate prediction**: replace EMA smoothing with LSTM or XGBoost models trained on historical rate data.
2. **Multi-chain deployment**: extend to Arbitrum, Base, Optimism using cross-chain messaging.
3. **Additional protocols**: Morpho, Spark, Euler V2 adapters.
4. **Formal verification**: Certora or Halmos proofs for mathematical invariants.
5. **Governance**: transition from owner multisig to token-weighted DAO.
6. **Risk scoring oracle**: on-chain publication of scoring data for composability.

10 Conclusion

We have presented AI Yield Vault, a hybrid off-chain/on-chain system for automated DeFi yield optimisation. Our key contributions are:

1. A four-factor MCDM scoring model (Eq. 15) that outperforms simple APY comparison in risk-adjusted returns.
2. EIP-712 cryptographic verification (Eq. 5) of every agent decision, combining off-chain intelligence with on-chain trustlessness.
3. Formal safety properties proven through 76,800+ randomised invariant calls with zero violations (Table 8).
4. Graceful degradation via Chainlink Automation fallback (Eq. 21).
5. Natural language interface through OpenClaw integration (Section 7.4).

The system demonstrates that the "agentic DeFi" paradigm—off-chain intelligence with on-chain verification—offers a meaningful advancement over both purely on-chain optimisers (limited by gas constraints) and purely off-chain systems (limited by trust requirements).

67 tests | 76,800+ invariant calls | 0 violations | 4-factor MCDM | EIP-712 signed | Chainlink fallback | OpenClaw chat

Code Availability

All source code, smart contracts, Python agent, tests, and deployment scripts are publicly available at: https://github.com/SergeySolovyev/ai-yield-vault.

[1] DefiLlama. DeFi TVL Rankings. https://defillama.com/, accessed April 2026.

[2] Aave. Aave V3 Technical Paper. https://github.com/aave/aave-v3-core/blob/master/techpaper/, 2023.

[3] Compound Labs. Compound III Documentation. https://docs.compound.finance/, 2023.

[4] J. Bebel and T. Snyder. EIP-4626: Tokenized Vaults. https://eips.ethereum.org/EIPS/eip-4626, 2022.

[5] S. Mudge, S. Ponč̆o, and F. Bon. ERC-1967: Proxy Storage Slots. https://eips.ethereum.org/EIPS/eip-1967, 2019.

[6] R. Floersch and M. S. Johnson. EIP-712: Typed Structured Data Hashing and Signing. https://eips.ethereum.org/EIPS/eip-712, 2017.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[8] OpenZeppelin. A Novel Defense Against ERC-4626 Inflation Attacks. OpenZeppelin Blog, 2023.

[9] Chainlink Labs. Chainlink Automation Documentation. https://docs.chain.link/chainlink-automation, 2024.

[10] OpenZeppelin. OpenZeppelin Contracts v5.x. https://docs.openzeppelin.com/contracts/5.x/, 2024.

[11] Paradigm. Foundry: Ethereum Development Toolkit. https://book.getfoundry.sh/, 2024.

[12] Yearn Finance. Yearn V3 Documentation. https://docs.yearn.fi/, 2024.

[13] Beefy Finance. Beefy Documentation. https://docs.beefy.finance/, 2024.

[14] Idle Finance. Idle Best Yield Documentation. https://docs.idle.finance/, 2024.

[15] Almanak. Almanak: AI-Powered DeFi Strategies. https://www.almanak.co/, 2024.

A Contract Addresses (Sepolia Testnet)

Contract	Address
Aave V3 Pool	`0x6Ae43d3271ff6888e7Fc43Fd7321a503ff738951`
Compound V3 Comet (USDC)	`0xAec1F48e02Cfb822Be958B68C7957156EB3F0b6e`
USDC (Sepolia)	`0x94a9D9AC8a22534E3FaCa9F4e7F2E2cf85d5E4C8`

B Configuration Parameters

Parameter	Value	Description
Cooldown period	3,600 s	Min time between rebalances
Agent timeout	21,600 s	Chainlink fallback activation
Idle buffer	200 bps (2%)	USDC kept idle for withdrawals
EMA alpha	3,000 bps (30%)	Weight on new observation
Max rate jump	500 bps (5%)	Rate manipulation guard
Signature max age	300 s	EIP-712 freshness window
Score threshold	0.05	Min delta to trigger rebalance
Decimals offset	6	Virtual shares for inflation protection
Management fee	50 bps (0.5%)	Annual fee on TVL
Performance fee	1,000 bps (10%)	Fee on profit above HWM