Superloop: Architecture Deep-Dive, Opportunities & Development Roadmap

Prepared for: Superlend Core Team
Date: February 2026
Classification: Internal

Table of Contents

1. Executive Summary

Superloop is a **modular, operator-managed ERC-4626 vault system** with async deposit/withdrawal queues, pluggable strategy execution via a module system, and decoupled accounting. Its architecture already mirrors emerging DeFi vault standards (ERC-7540, ERC-7575) in spirit, putting it in a strong position to become a composability hub for yield strategies across EVM chains.

This document catalogs **what the architecture uniquely enables**, maps each opportunity against the latest vault standards, and provides effort estimates for the engineering team.

Key Takeaways

- **3 high-impact, low-effort wins** can be shipped in 2-4 weeks each
- **Standards compliance (ERC-7540/7575)** requires an adapter layer, not a rewrite
- **The module system is the moat** - every new strategy is just a new module contract
- **HyperEVM staking** is a differentiated product opportunity no one else has packaged

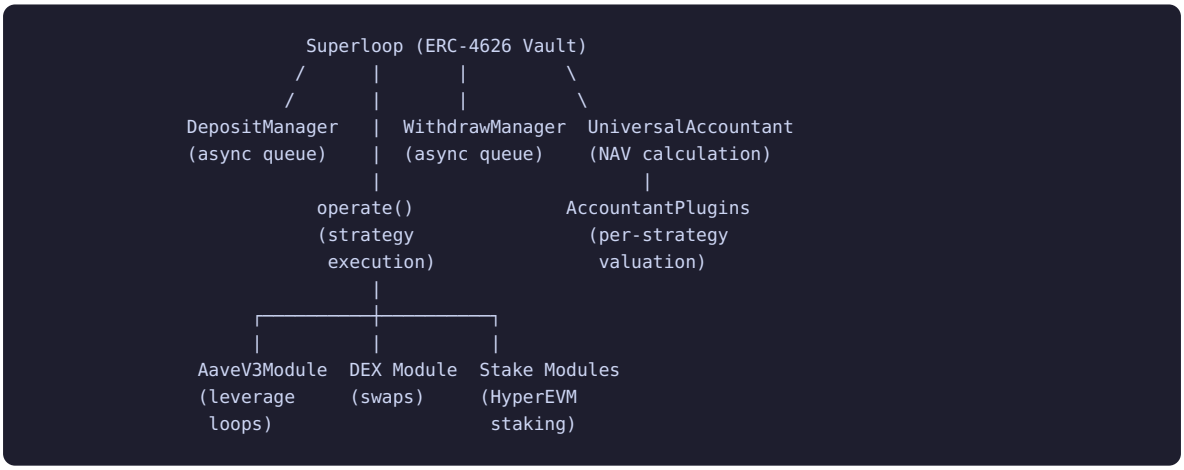
2. Architecture Overview

Core Contract: `Superloop.sol`

An upgradeable ERC-4626 vault with UUPS proxy pattern that adds:

- **Operator-gated execution** via `operate(DecoderAndSanitizer, targets, data, values)`
- **Async deposit/withdrawal queues** via dedicated manager contracts
- **Pluggable accounting** via `UniversalAccountant` with strategy-specific plugins
- **Configurable fees** (platform, performance, entry, exit) with share-locking

Contract Dependency Graph



How a Dollar Flows Through the System

```
1. USER calls requestDeposit(1000 USDC)
   └─> DepositManager escrows 1000 USDC, creates DepositRequest

2. OPERATOR calls resolveDepositRequests([requestIds])
   └─> USDC transferred to vault, shares minted to users

3. OPERATOR calls operate([AaveV3Module.supply(1000 USDC)])
   └─> Vault executes Aave supply via delegatecall
   └─> DecoderAndSanitizer validates target addresses + function selectors

4. UniversalAccountant.getVaultValue() queries AaveV3AccountantPlugin
   └─> Plugin reads aToken balance + debt, returns NAV
   └─> Share price updates automatically

5. USER calls requestWithdraw(500 shares)
   └─> WithdrawManager creates WithdrawRequest

6. OPERATOR calls operate([AaveV3Module.withdraw(500 USDC worth)])
   └─> Vault withdraws from Aave

7. OPERATOR calls resolveWithdrawRequests([requestIds])
   └─> USDC sent to users, shares burned
```

3. Current Capabilities

Strategy Modules (17 deployed)

Module	Purpose	Chain Focus
`AaveV3Module`	Supply, borrow, repay, leverage loops	Multi-chain
`CompoundV3Module`	Supply/withdraw on Compound	Multi-chain
`VaultSupplyModule`	Deposit into other ERC-4626 vaults	Multi-chain
`UniversalDexModule`	Swaps via any DEX aggregator	Multi-chain
`HyperliquidStakeModule`	Native HyperEVM staking	HyperEVM
`KinetiqStakeModule`	Kinetiq liquid staking (stHYPE)	HyperEVM
`HyperbeatStakingModule`	Hyperbeat staking	HyperEVM
`ThunderheadStakeModule`	Thunderhead staking	HyperEVM
`UnwrapModule`	WETH/ETH unwrapping	Multi-chain
`WrapModule`	ETH/WETH wrapping	Multi-chain
`NativeTransferModule`	Native token transfers	Multi-chain
`ERC20TransferModule`	ERC-20 token transfers	Multi-chain
`CurveModule`	Curve pool interactions	Multi-chain
`MorphoModule`	Morpho lending	Multi-chain
`EulerModule`	Euler lending	Multi-chain
`FluidModule`	Fluid (Instadapp) lending	Multi-chain
`LayerBankModule`	LayerBank lending	Multi-chain

Accounting Plugins

Plugin	What It Values
`AaveV3AccountantPlugin`	aToken balances minus debt positions
`CompoundV3AccountantPlugin`	Compound supply positions
`ERC20AccountantPlugin`	Raw ERC-20 balances held by vault
`ERC4626AccountantPlugin`	Shares in other ERC-4626 vaults
`StakedTokenAccountantPlugin`	Liquid staking token positions
`NativeAccountantPlugin`	Native ETH/token balance

Infrastructure

- **VaultRouter**: Batched multi-step transactions (deposit + stake in one tx)

- **DecoderAndSanitizer**: Per-module allowlist of valid targets and function selectors
- **RolesAuthority**: Granular role-based access control for all operations
- **Queue System**: Supports GENERAL, STOP_LOSS, INSTANT withdrawal types with configurable fees

---

4. ERC Standards Landscape

Standards That Matter for Superloop

Standard	What It Does	Status	Relevance
ERC-4626	Base tokenized vault interface	Final	Already implemented
ERC-7540	Async deposit/redeem requests	Final	Maps directly to your queue system
ERC-7575	Decoupled share token from vault	Final	Enables multi-asset entry points
ERC-7535	Native ETH as vault asset	Final	Perfect for HyperEVM staking vaults
ERC-7741	Signed operator authorization	Draft	Gasless approvals for your operator model

Your Architecture vs. ERC-7540

Aspect	ERC-7540 Spec	Superloop Current
Deposit request	`requestDeposit(assets, controller, owner)`	`requestDeposit(amount, onBehalfOf)`
Redemption request	`requestRedeem(shares, controller, owner)`	`requestWithdraw(amount, requestType)`
Fulfillment model	**Pull** (user claims)	**Push** (operator delivers)
Cancel deposit	`cancelDepositRequest(requestId, controller)`	`cancelDepositRequest(id)`
Cancel redeem	`cancelRedeemRequest(requestId, controller)`	`cancelWithdrawRequest(id, requestType)`
Request tracking	`pendingDepositRequest()` / `claimableDepositRequest()`	`depositRequest[id].status` enum
Operator auth	`setOperator()` / `isOperator()`	`RolesAuthority` role system

Key insight: The gap is narrow. An adapter contract can bridge the interface differences without touching core logic.

---

5. Opportunity Map

Overview Matrix

#	Opportunity	Impact	Effort	Category
1	ERC-7540 Adapter	High	Low	Standards
2	Native ETH Vault (ERC-7535)	High	Low	New Product
3	Deposit/Withdrawal Netting	High	Low	Optimization
4	Multi-Asset Entry (ERC-7575)	High	Medium	Standards
5	DCA / TWAD Deposits	Medium	Low	Feature
6	Vault-of-Vaults Aggregator	High	Medium	New Product
7	Delta-Neutral Basis Trade	Very High	Medium	New Strategy
8	Senior/Junior Tranching	High	Medium	New Product
9	Cross-Chain Vault	Very High	High	Infrastructure
10	Solver/Intent-Based Execution	High	High	Decentralization
11	Conditional Queues (Stop-Loss)	Medium	Low	Feature
12	Queue Position Marketplace	Medium	Medium	Feature

6. Detailed Proposals

Proposal 1: ERC-7540 Compatibility Adapter

What: A wrapper contract that translates between the ERC-7540 interface and Superloop's existing DepositManager/WithdrawManager.

Why it matters: Any protocol, aggregator, or frontend that speaks ERC-7540 can integrate Superloop without custom code. This is the emerging standard for async vaults.

How it works:

```
External Protocol (speaks ERC-7540)
        |
        v
  SuperloopERC7540Adapter
    - requestDeposit() → depositManager.requestDeposit()
    - requestRedeem()  → withdrawManager.requestWithdraw()
    - pendingDepositRequest() → reads depositManager state
    - claimableDepositRequest() → reads resolved requests
    - deposit()/mint() → claims resolved deposits
    - Implements IERC165, IERC7540Deposit, IERC7540Redeem
        |
        v
  Superloop Core (unchanged)
```

Effort: ~1-2 weeks. Pure adapter pattern, no core contract changes.

Files to create:
- `src/adapters/SuperloopERC7540Adapter.sol` (~300 lines)
- `src/interfaces/IERC7540.sol` (~100 lines)
- Tests

Proposal 2: Native ETH Staking Vault (ERC-7535)

What: A Superloop vault that accepts native ETH directly (no wrapping), deploys to HyperEVM staking modules, and issues liquid shares.

Why it matters: Users send ETH and get a yield-bearing token. No wrapping, no manual staking, no choosing between Kinetiq/Hyperbeat/Thunderhead. The operator auto-routes to the best option.

How it works:

```
User sends ETH (payable deposit)
    |
    v
NativeETHVault (ERC-7535 + ERC-4626)
  - asset() returns 0xEeee...EEeE (ETH sentinel)
  - deposit() is payable, uses msg.value
  - withdraw() sends ETH via call
    |
    v
Operator routes via operate():
  ┌─ HyperliquidStakeModule (native staking)
  ├─ KinetiqStakeModule     (stHYPE)
  ├─ HyperbeatStakingModule (hyperbeat)
  └─ ThunderheadStakeModule (thunderhead)
    |
    v
StakedTokenAccountantPlugin values the position
```

Effort: ~2-3 weeks. Most modules already exist. Need a new vault base that handles `payable` deposits and ETH transfers.

Files to create/modify:
- `src/vaults/NativeETHVault.sol` (~200 lines, extends Superloop with payable)
- Modify `UniversalAccountant` to support native ETH pricing
- Tests

Proposal 3: Deposit/Withdrawal Netting

What: When resolving queues, match pending deposits against pending withdrawals directly. Deposit USDC goes straight to withdrawing user; no strategy interaction needed.

Why it matters: Eliminates unnecessary gas costs and slippage from round-tripping through Aave or other protocols. In a vault with $10M deposits and $8M withdrawals in the same epoch, only $2M net needs to touch the strategy.

How it works:

```
Epoch N:
  Pending Deposits:  $10M USDC
  Pending Withdrawals: $8M USDC

WITHOUT netting:
  1. Deposit $10M into Aave     (gas + potential slippage)
  2. Withdraw $8M from Aave     (gas + potential slippage)
  Net strategy movement: $10M + $8M = $18M moved

WITH netting:
  1. Route $8M directly from depositors → withdrawers
  2. Deposit only $2M net into Aave
  Net strategy movement: $2M + $8M direct = saves $16M of unnecessary moves
```

Effort: ~1-2 weeks. Logic lives in DepositManager/WithdrawManager resolution functions.

Files to modify:
- `src/managers/DepositManager.sol` (add netting logic to `resolveDepositRequests`)
- `src/managers/WithdrawManager.sol` (coordinate with deposit resolution)
- Tests

Proposal 4: Multi-Asset Entry Points (ERC-7575)

What: Deploy multiple vault entry points (one per accepted token) that all mint the same share token. Users deposit USDC, USDT, DAI, or ETH and receive the same Superloop share.

Why it matters: Removes the "swap to the right token first" friction. Any supported token works. The vault swaps internally using UniversalDexModule.

How it works:

```
USDC Entry Vault (IERC7575)    USDT Entry Vault (IERC7575)
  asset() => USDC                asset() => USDT
  share() => SUPERLOOP_SHARE     share() => SUPERLOOP_SHARE
        \                              /
         \____________________________/
                      |
              SUPERLOOP_SHARE (ERC-20)
              One share, multiple entry tokens
              Backed by unified Aave strategy
```

Effort: ~3-4 weeks. Requires decoupling the share token from the vault contract. More architectural, but the VaultRouter already does part of this.

Files to create:
- `src/tokens/SuperloopShare.sol` (standalone ERC-20 share token)
- `src/vaults/AssetEntryVault.sol` (per-asset entry point)
- Modify Superloop to support external share token minting
- Tests

Proposal 5: Time-Weighted Average Deposits (DCA)

What: Users submit a large deposit request with a schedule (e.g., "deposit $100K over 10 epochs"). The operator drip-feeds the deposit across multiple resolution cycles.

Why it matters: Reduces timing risk for large depositors. Instead of getting one share price, they get an average. This is the "DCA into the vault" pattern.

How it works:

```
User: requestScheduledDeposit(100_000 USDC, 10 epochs)
  └─> Creates 10 sub-requests of 10,000 USDC each

Epoch 1: resolve sub-request 1 → 10,000 USDC at price P1
Epoch 2: resolve sub-request 2 → 10,000 USDC at price P2
...
Epoch 10: resolve sub-request 10 → 10,000 USDC at price P10

User receives shares at average price (P1+P2+...+P10)/10
```

Effort: ~1-2 weeks. Extension to DepositManager.

Files to modify:
- `src/managers/DepositManager.sol` (add scheduled request type)
- Tests

Proposal 6: Vault-of-Vaults Strategy Aggregator

What: A meta-vault that holds positions across multiple Superloop vaults (or third-party ERC-4626 vaults), auto-rebalancing between them.

Why it matters: Users get diversified yield exposure through a single deposit. The vault automatically allocates to the best-performing strategies.

How it works:

```
Meta-Vault (Superloop instance)
    |
    ├─ VaultSupplyModule → Superloop Aave Leverage Vault (30%)
    ├─ VaultSupplyModule → Superloop Staking Vault (30%)
    ├─ VaultSupplyModule → External Morpho Vault (20%)
    └─ VaultSupplyModule → External Euler Vault (20%)

Operator rebalances periodically based on yield + risk
ERC4626AccountantPlugin values all positions
```

Effort: ~2-3 weeks. VaultSupplyModule and ERC4626AccountantPlugin already exist. Need orchestration logic and rebalance triggers.

Files to create:
- Deployment scripts and configuration for meta-vault
- Rebalance helper contract (~100 lines)
- Tests

Proposal 7: Delta-Neutral Basis Trade Strategy

What: A new module that executes the basis trade: long spot + short perpetual of the same asset. Earns funding rate yield with near-zero directional exposure.

Why it matters: Historically 15-30%+ APY on major assets. Market-neutral. Huge demand. Currently only available via centralized platforms (Ethena).

How it works:

```
Vault holds $100K USDC
    |
    ├─ BasisTradeModule.openPosition():
    |    1. Buy $50K ETH spot (via UniversalDexModule)
    |    2. Short $50K ETH perp on HyperLiquid
    |    Net exposure: ~0 (delta neutral)
    |    Income: funding rate payments (when positive)
    |
    ├─ BasisTradeAccountantPlugin:
    |    Position value = spot value + unrealized PnL on perp
    |    Accounts for funding received/paid
    |
    └─ Risk management:
         - Max leverage limits in DecoderAndSanitizer
         - Auto-unwind if funding goes negative for N epochs
         - Position size limits per asset
```

Effort: ~4-6 weeks. Needs new module for perp trading, new accountant plugin for perp position valuation, and careful risk parameterization.

Files to create:
- `src/modules/BasisTradeModule.sol` (~400 lines)
- `src/decoders/BasisTradeDecoderAndSanitizer.sol` (~200 lines)
- `src/plugins/BasisTradeAccountantPlugin.sol` (~150 lines)
- Tests + integration tests

Proposal 8: Senior/Junior Tranching

What: Two vaults backed by the same strategy. The Senior vault gets predictable yield (e.g., 5% fixed). The Junior vault absorbs losses first but gets all excess yield.

Why it matters: Opens the vault to two different user profiles: conservative (institutions wanting fixed yield) and aggressive (degens wanting leveraged yield).

How it works:

```
Strategy generates 12% yield on $10M
    |
    ├─ Senior Vault ($7M, 70%)
    |    Gets: fixed 5% = $350K
    |    Risk: protected by junior tranche
    |
    └─ Junior Vault ($3M, 30%)
         Gets: remaining = $1.2M - $350K = $850K = 28.3% APY
         Risk: absorbs first $3M of losses

If strategy loses 10% ($1M):
    Senior: $7M (fully protected, junior absorbs loss)
    Junior: $3M - $1M = $2M (33% loss)

If strategy loses 40% ($4M):
    Senior: $7M - $1M = $6M (14% loss, after junior wiped out)
    Junior: $0 (wiped out)
```

Effort: ~4-6 weeks. Requires a tranche controller contract that manages NAV splitting between two Superloop vault instances.

Files to create:
- `src/tranches/TrancheController.sol` (~500 lines)
- `src/plugins/TrancheAccountantPlugin.sol` (~200 lines)

- Modified deployment scripts for paired vaults
- Tests

Proposal 9: Cross-Chain Vault

What: A vault on Chain A that deploys capital to strategies on Chain B, using the async queue to absorb bridge latency.

Why it matters: Users deposit on the chain they're on. Capital flows to wherever yield is highest. Bridge delay maps perfectly to your existing async queue pattern.

How it works:

Effort: ~8-12 weeks. Requires bridge integration, cross-chain messaging, remote executor contract, and cross-chain accountant plugin. Significant but high-value.

Files to create:
- `src/bridges/BridgeModule.sol`
- `src/executors/RemoteExecutor.sol` (deployed on Chain B)
- `src/plugins/CrossChainAccountantPlugin.sol`
- Bridge adapter contracts
- Extensive testing

Proposal 10: Solver/Intent-Based Strategy Execution

What: Replace the single trusted operator with a competitive solver market. Anyone can submit strategy execution bundles; the best one (highest yield, lowest cost) wins.

Why it matters: Decentralizes the operator role, reduces trust assumptions, and creates competitive pressure for better execution.

How it works:

```
Current: Single Operator
  operator.operate([actions])  ← trust required

Proposed: Solver Competition
  1. Vault publishes "intent": "Deploy $1M to highest Aave yield"
  2. Solvers submit execution plans (off-chain)
  3. On-chain auction selects best plan
  4. Winner executes via operate()
  5. Performance bond slashed if execution deviates
```

Effort: ~8-12 weeks. Needs auction mechanism, solver registry, performance bonds, and off-chain infrastructure.

Proposal 11: Conditional Queue Triggers

What: Extend withdrawal requests with conditions: "withdraw when share price hits X" (take-profit) or "withdraw if share price drops below Y" (stop-loss).

Why it matters: Basic risk management without requiring users to monitor positions. The operator checks conditions during resolution.

How it works:

```
requestConditionalWithdraw(
    shares: 1000,
    condition: STOP_LOSS,
    triggerPrice: 0.95e18  // withdraw if share price < 0.95
)

Operator checks each epoch:
  if (sharePrice < request.triggerPrice) → resolve withdrawal
  else → skip (remains pending)
```

Effort: ~1-2 weeks. Your WithdrawManager already supports STOP_LOSS type. Need to add price-trigger logic.

Files to modify:
- `src/managers/WithdrawManager.sol` (add trigger conditions)
- Tests

Proposal 12: Queue Position Marketplace

What: Allow users to trade their pending deposit/withdrawal positions as transferable tokens (ERC-721 or ERC-1155).

Why it matters: Positions stuck in the async queue become liquid. "I have a $100K deposit pending for epoch 5" becomes a tradeable asset.

Effort: ~3-4 weeks. Wrap queue positions as NFTs, integrate with marketplace.

---

7. Effort Estimates & Prioritization

Effort Classification

Label	Engineering Time	Description
XS	< 1 week	Configuration or minimal code
S	1-2 weeks	Single contract, well-defined scope
M	3-4 weeks	Multiple contracts, moderate complexity
L	4-6 weeks	New subsystem, needs design iteration
XL	8-12 weeks	Cross-system, infrastructure-level

Priority Matrix

Recommended Execution Order

Phase 1: Quick Wins (Weeks 1-4)

#	Item	Effort	Why First
1	ERC-7540 Adapter	S (1-2w)	Instant ecosystem composability, no core changes
3	Deposit/Withdrawal Netting	S (1-2w)	Immediate gas savings, improves economics
11	Conditional Queues	S (1w)	Small addition to existing queue system

Phase 2: New Products (Weeks 5-10)

#	Item	Effort	Why Second
2	Native ETH Vault (ERC-7535)	S-M (2-3w)	Differentiated product on HyperEVM
5	DCA Deposits	S (1-2w)	Attractive feature for large depositors
6	Vault-of-Vaults	M (2-3w)	Meta-strategy using existing modules

Phase 3: Growth Strategies (Weeks 11-20)

#	Item	Effort	Why Third
4	Multi-Asset Entry (ERC-7575)	M (3-4w)	Removes deposit friction
7	Basis Trade Module	L (4-6w)	Highest yield potential
8	Senior/Junior Tranching	L (4-6w)	Opens institutional demand

Phase 4: Infrastructure (Weeks 20+)

#	Item	Effort	Why Last
9	Cross-Chain Vault	XL (8-12w)	Infrastructure-heavy, needs bridge partners
10	Solver Competition	XL (8-12w)	Decentralization layer, complex game theory
12	Queue Marketplace	M (3-4w)	Nice-to-have, needs liquidity

---

8. Appendix: Architecture Diagrams

Full System Architecture

Module Development Pattern

Adding a new strategy to Superloop follows a consistent 3-file pattern:

```
1. Module Contract (src/modules/NewModule.sol)
   - Functions the vault can delegatecall
   - Pure strategy logic, no state

2. Decoder & Sanitizer (src/decoders/NewDecoderAndSanitizer.sol)
   - Validates targets and function selectors
   - Allowlists specific addresses

3. Accountant Plugin (src/plugins/NewAccountantPlugin.sol)
   - Reads position value from external protocol
   - Returns NAV contribution to UniversalAccountant
```

This is Superloop's moat: every new strategy integration is just these 3 files. No core contract changes. No upgrades. No migrations.

Document generated from Superloop codebase analysis, February 2026.