NEMESIS

Every trader needs a Nemesis.

Social Battle Trading Interface
on Hyperliquid

Hackathon Submission

LI.FI • Valantis • Pear Protocol • Salt • Insilico

January 2026

Contents

1 Executive Summary

Nemesis is a visual novel-inspired battle trading interface that transforms trading from a solitary activity into a social, competitive experience. Users can challenge rivals to direct counterparty arrangements, pool capital with friends, and participate in prediction markets on crypto price movements.

This document describes the hackathon submission targeting five sponsor tracks simultaneously, clearly distinguishing between what is implemented for the hackathon demonstration and what represents the prospective production system.

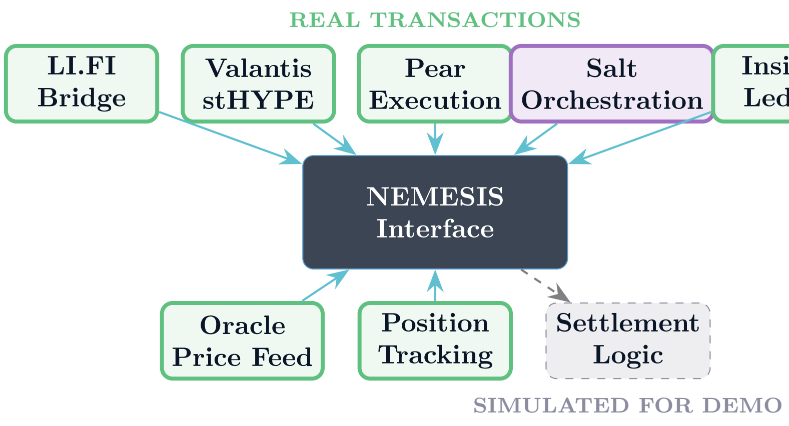

Figure 1: Architecture showing real integrations (solid) versus simulated systems (dashed)

1.1 Hackathon Scope

The hackathon submission demonstrates real API integrations with all five sponsors wrapped in a compelling user experience. The prediction market functionality serves as the narrative context for these integrations.

Component	Hackathon Status	Transaction Type
Wallet Connection	Implemented	Real
LI.FI Cross-Chain Bridge	Implemented	Real
Valantis stHYPE Deposit	Implemented	Real
Pear Position Execution	Implemented	Real
Salt Organisation & Robo Manager	Implemented	Real
Insilico Trade Ledger API	Implemented	Real
Oracle Price Feeds	Implemented	API Call
Trade History	Implemented	API Call
Position Tracking	Implemented	API Call
P&L Calculations	Implemented	Computed
Leaderboard	Implemented	Computed
Order Matching	Simulated	Local State
Settlement and Payouts	Not Implemented	Future Work

Table 1: Implementation status for hackathon submission

2 Product Vision

2.1 The Battle Trading Concept

Traditional crypto trading is adversarial but impersonal. Nemesis makes the adversarial nature explicit and social through three mechanisms.

2.1.1 Duels

Direct one-on-one challenges between traders who disagree on a market outcome. User A believes ETH will exceed a price target; User B disagrees. They stake equal amounts in USDC, and the winner takes the combined stake minus platform fees.

2.1.2 Trading Parties

Friends pool capital and copy-trade together. When the party leader enters a position, members can automatically mirror the trade at a configurable ratio. Win together, lose together.

2.1.3 Guilds

Persistent groups with shared strategies implemented through Salt Organisations. Members create a Salt Organisation, set policies defining what the Nemesis Robo Manager can do on their behalf, and the bot executes strategies within those constraints automatically.

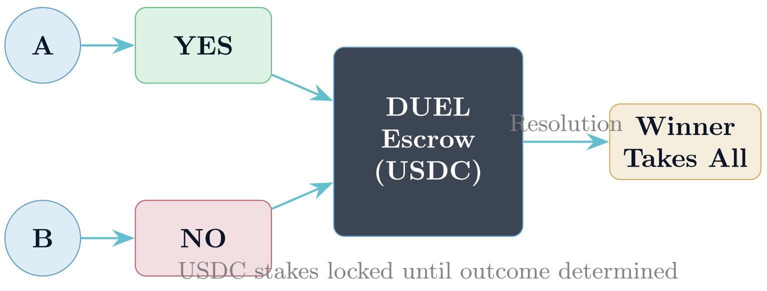

Figure 2: Duel mechanics: direct counterparty arrangement with USDC escrow

2.2 Visual Novel Interface

The interface draws from Japanese visual novels with an AI avatar guiding users through the experience. This aesthetic choice serves the hackathon goal of attracting net new users by making trading approachable.

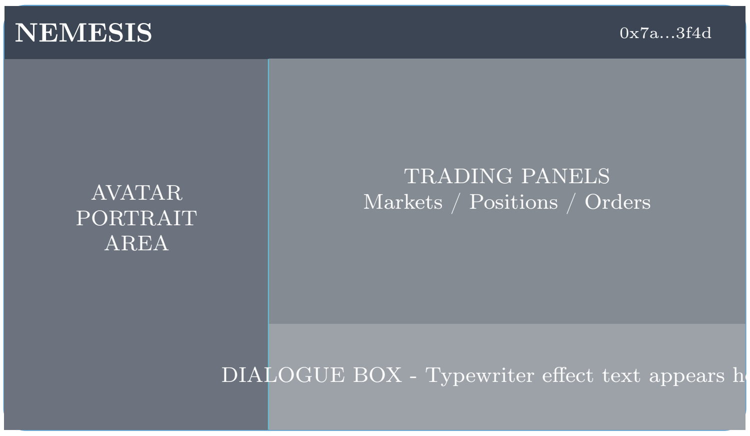
NEMESIS
0x7a...3f4d
AVATAR PORTRAIT AREA
TRADING PANELS
Markets / Positions / Orders
DIALOGUE BOX - Typewriter effect text appears h

Figure 3: Interface layout: visual novel with trading panels

3 Track Integrations

3.1 Track 1: LI.FI — Cross-Chain Onboarding

Prize Pool: $6,000

Users arrive on any chain with any token and can onboard to HyperEVM with a single transaction. The LI.FI SDK handles routing, swapping, and bridging.

- User connects wallet on Ethereum, Arbitrum, Base, or other supported chain
- LI.FI finds optimal route to HyperEVM
- Single transaction moves funds cross-chain
- User lands on HyperEVM with USDC ready to trade

3.2 Track 2: Valantis — Yield Generation

Prize Pool: $2,000

Protocol treasury deposits to Valantis stHYPE for yield generation. The approximately 8% APY on stHYPE provides sustainable protocol revenue.

- Protocol treasury accumulates HYPE from fees
- HYPE staked to Valantis stHYPE contract
- Yield accrues to protocol treasury
- User escrow remains in USDC (no HYPE price exposure for users)

3.3 Track 3: Pear Protocol — Trade Execution

Prize Pool: $3,500

Pear provides perpetual position execution for hedging imbalanced prediction market exposure (prospective) and direct trading features.

- REST API integration for position management
- Open long/short positions on supported pairs
- Position monitoring and P&L tracking
- Prospective: automated hedging for protocol exposure

3.4 Track 4: Salt — Automated Strategy Execution

Prize Pool: $1,000

Salt provides the orchestration layer for non-custodial automated execution. This is the most architecturally significant integration.

3.4.1 Core Concepts

Robo Guardians are an organisation's automated co-signers that enforce policies. They are hosted on the organisation's own infrastructure (AWS) and hold a portion of key material for every account. Robo Guardians will only co-sign transactions that pass all policy checks.

Robo Managers are third-party bots, humans, or AI agents that can execute transactions on an organisation's behalf without taking custody of assets. The pitch is: *"Hire a bot. Keep custody."*

Nemesis as Robo Manager: The Nemesis platform itself functions as a Robo Manager. Users create a Salt Organisation, invite Nemesis as a Robo Manager, set policies defining constraints, and Nemesis executes prediction market strategies on their behalf 24/7.

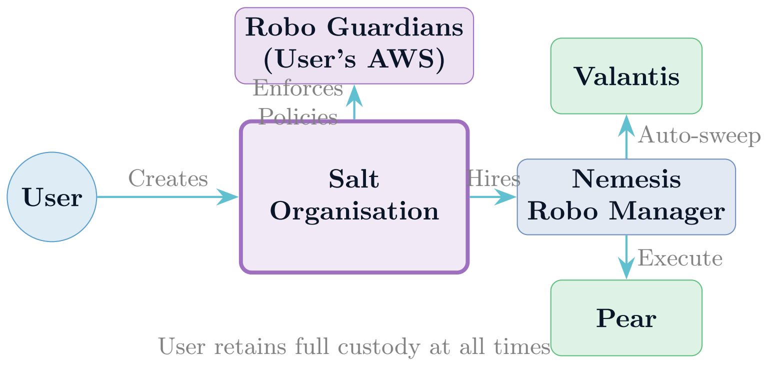

Figure 4: Salt architecture: Nemesis as Robo Manager executing within policy constraints

3.4.2 Policy Types

Salt supports granular policy controls:

- **Allowed Recipients:** Whitelist of addresses the Robo Manager can interact with (e.g., only Pear and Valantis contracts)
- **Denied Recipients:** Blacklist of forbidden addresses
- **Transaction Limits:** Maximum amounts per transaction or time period
- **Nominated Approvers:** Require human approval for certain actions

3.4.3 Implementation

- `salt-sdk` npm package for TypeScript integration
- SIWE (Sign-In With Ethereum) authentication flow
- Organisation creation and member management
- Account setup with Robo Guardian coordination
- Policy definition and enforcement
- Transaction submission within policy constraints
- HyperEVM explicitly supported on mainnet and testnet

3.5 Track 5: Insilico — Trade Ledger API

Prize Pool: $5,000

Four API endpoints provide trade history, position tracking, P&L calculations, and leaderboards. Data sourced from Hyperliquid Info API with abstraction layer for future data source swaps.

Endpoint	**Method**	**Description**
`/v1/trades`	GET	Trade history with filtering by user, coin, time range
`/v1/positions/history`	GET	Position timeline showing opens, modifications, closes
`/v1/pnl`	GET	Cumulative profit and loss calculations
`/v1/leaderboard`	GET	Ranked user performance aggregation

Table 2: Trade Ledger API endpoints

4 Business Model

4.1 Revenue Streams

Nemesis generates revenue through two primary mechanisms:

4.1.1 Duel Escrow Yield

All prediction market escrow is denominated in USDC to provide defined risk without collateral volatility. While USDC is locked in escrow during active duels, it earns yield via lending protocols at approximately 3–8% APY. Users receive their principal back; Nemesis retains the yield.

Metric	**Conservative**	**Optimistic**
TVL in Escrow	$1,000,000	$10,000,000
Average Lock Duration	14 days	14 days
Lending APY	5%	8%
Annual Yield Revenue	$50,000	$800,000

Table 3: Escrow yield revenue projections

4.1.2 Duel Fees

A percentage of the winning pot or flat fee per duel creation. Fee structure to be determined based on market research and competitive analysis.

4.1.3 Protocol Treasury

Protocol treasury accumulates HYPE from fee conversion and stakes via Valantis for approximately 8% APY. User escrow is never exposed to HYPE price risk—only protocol-owned funds are staked.

4.2 Track Integrations as TVL Drivers

The five track integrations are features that drive TVL, not direct revenue sources:

- **LI.FI:** Reduces friction → more users onboard → more escrow
- **Valantis:** Protocol treasury yield (not user escrow)
- **Pear:** Attracts traders → more duels → more escrow
- **Salt:** Guilds lock more capital → more escrow
- **Insilico:** Proves track records → attracts whales → more escrow

5 Oracle Integration

5.1 Current: HyperCore Price Feeds

- HyperCore Read precompile at `0x800` for gas-free price reads
- Real-time price updates for supported assets
- Market display reflects actual oracle prices

5.2 Future: Settlement Oracle

Settlement requires determining the asset price at market expiry to identify winners. This logic is not implemented for the hackathon but the oracle infrastructure is in place.

- Time-weighted average price (TWAP) to prevent manipulation
- Snapshot at market expiry timestamp
- Fallback oracles (Pyth, Chainlink) for redundancy

6 Simulated Systems

The following systems are simulated for the hackathon demonstration. They represent future development work required for a production prediction market.

6.1 Order Matching

Orders fill instantly without a matching engine.

- No order book implementation
- No counterparty matching
- Instant fill simulation for demonstration

6.2 Settlement

Market resolution and payout distribution are not implemented.

- No winner determination logic
- No fund distribution mechanism
- No escrow release automation

7 Prospective Architecture

This section describes the architecture required for a production prediction market. These components represent post-hackathon development.

7.1 Smart Contract Requirements

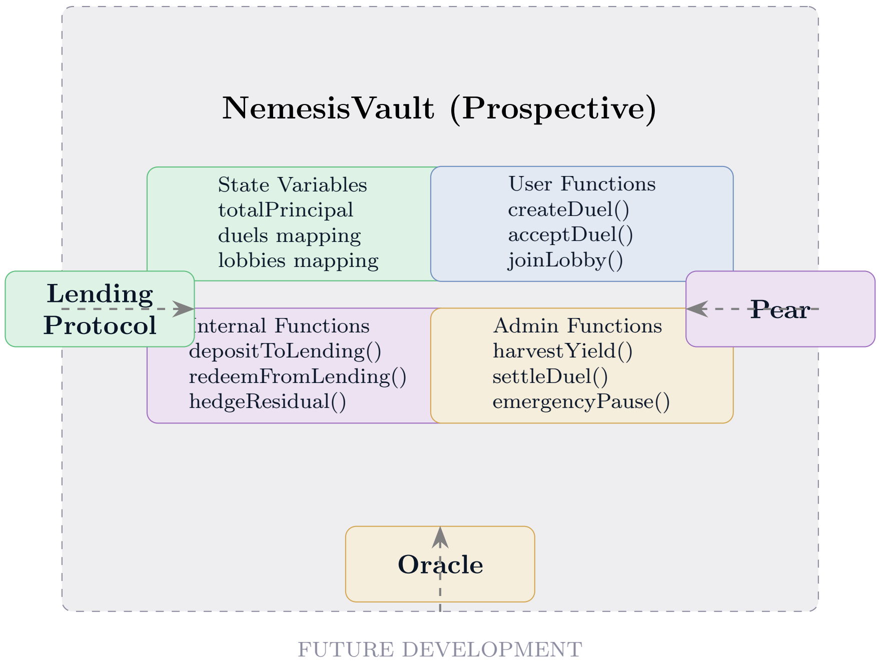

Figure 5: Prospective smart contract architecture

7.2 Internal Matching Engine

A production system would match opposing bets internally before external hedging.

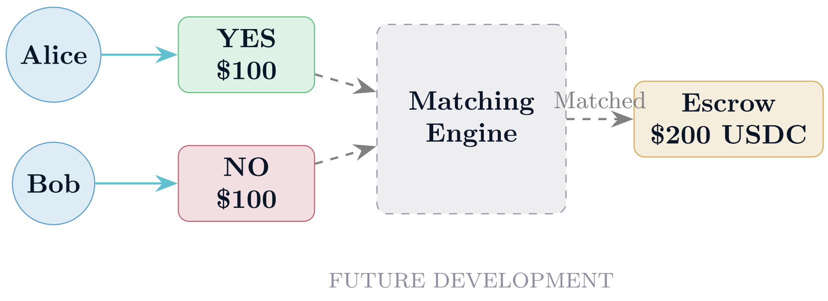

Figure 6: Prospective internal matching eliminates market exposure

8 Development Roadmap

Phase	Milestone	Deliverables
0	Hackathon	Frontend, five track API integrations, oracle price feeds, trade ledger API
1	Post-Hackathon	Smart contracts for escrow and matching, settlement logic
2	Beta	Real order matching, settlement automation, audit
3	Launch	Production deployment, liquidity bootstrapping

Table 4: Development roadmap from hackathon to production

8.1 Business Blockers

The following must be addressed before launching as a production service:

Blocker	Description
Smart Contracts	Escrow, matching, and settlement logic
Settlement Logic	Winner determination and payout distribution
Liquidity	Counterparties for unmatched positions
Legal Structure	Prediction markets face regulatory requirements
Backend Infrastructure	Data serving at scale

Table 5: Requirements for production launch

9 Technical Stack

Component	Technology
Runtime	Bun
Build System	Nix Flakes
DOM Updates	morphdom
Wallet Connection	viem + @wagmi/core
Mobile Wallets	WalletConnect v2
Cross-Chain Bridge	@lifi/sdk
Trading Execution	Pear REST API
MPC Orchestration	salt-sdk
Yield Generation	Valantis stHYPE
Price Feeds	HyperCore Oracle
Trade Ledger	Hyperliquid Info API

Table 6: Technology stack

9.1 Chain Configuration

Network	Chain ID	RPC
HyperEVM Mainnet	999	https://rpc.hyperliquid.xyz/evm
HyperEVM Testnet	998	https://rpc.hyperliquid-testnet.xyz/evm
Arbitrum (Salt Orchestration)	42161	https://arb1.arbitrum.io/rpc

Table 7: Chain configuration

10 Conclusion

Nemesis demonstrates how five hackathon track technologies can integrate into a single coherent product. The visual novel interface provides an approachable entry point for new users, while the battle trading mechanics create social and viral growth potential.

The hackathon submission delivers:

- Real LI.FI integration for cross-chain onboarding
- Real Valantis integration for protocol treasury yield
- Real Pear integration for trade execution
- Real Salt integration for Robo Manager automated execution
- Real Insilico integration for trade ledger API
- Real oracle integration for live price feeds
- Polished frontend with visual novel aesthetic
- Simulated prediction market demonstrating the concept

The prospective production system would add smart contracts for escrow and settlement, winner determination logic, and the infrastructure required for a fully functional prediction market.

Every trader needs a Nemesis.