Hypurrliquid White Paper

Reverse-Engineered Hyperliquid Reimplementation

April 4, 2026

Abstract

This white paper describes the system-design view of the Hyperliquid reimplementation effort. The goal is not merely to build an exchange that feels similar. The goal is to reconstruct a deterministic exchange chain whose consensus, execution engine, liquidation logic, delayed EVM re-entry, and state-hash surfaces remain faithful enough to support replay and protocol research.

1 System Purpose

Hyperliquid is a purpose-built exchange chain that combines perpetual futures, spot markets, account abstraction and portfolio margin, lending via BOLE, prediction markets via HIP-4 outcomes, and a tightly coupled EVM surface. The core engineering challenge is not just matching orders. It is preserving one deterministic state machine across consensus, matching, clearing, validator control, EVM-originated actions, and state hashing.

2 System Model

At a high level the system is composed of five interacting planes:

1. **Consensus**. HyperBFT orders blocks, rotates proposers, and finalizes execution.
2. **Exchange state machine**. The `Exchange` object is the central L1 state surface.
3. **Matching and clearing**. Orders mutate books, fills mutate balances and positions, and clearing logic enforces margin and liquidation rules.
4. **EVM bridge surface**. HyperEVM runs alongside the L1 state machine and re-enters L1 through CoreWriter-delayed actions and transfer queues.
5. **Control and governance**. Validators, staking, bridge signatures, SetGlobal/VoteGlobal actions, and safety toggles sit above ordinary trading flow.

3 Architectural Layers

3.1 Consensus and Networking

The consensus layer decides block order, proposer rotation, and commitment. The networking layer carries validator-to-validator consensus traffic, gossip and peer sync, state/bootstrap handoff for catching-up nodes, and runtime signals such as concise hashes and validator vote surfaces.

Current repo truth is that ingress is broadcaster-mediated, validator and sentry admission is a separate transport gate, validator eligibility is epoch-scoped, and the network surface stays split between block-stream or bootstrap traffic and peer or RPC verification rather than one flat socket.

3.2 Exchange Execution

A block generally follows this shape:

1. recover and identify actors,
2. run `begin_block` hook surfaces,
3. deliver signed actions,
4. run block-finalization logic,
5. compute response-hash and state-hash material,
6. feed commit and vote surfaces.

3.3 Product Families

Product family	Core behavior	Main risk surface
Perps	matched books + clearinghouse	margin, liquidation, ADL
Spot	spot books and balances	transfer routing, spot bookkeeping
Portfolio margin	unified risk over eligible assets	portfolio maintenance ratio and liquidation ordering
BOLE	borrow/lend pool	utilization, health factor, market/partial/backstop liquidation
Outcomes	1x outcome markets	settlement and collateral conservation
EVM-originated actions	delayed CoreWriter path	delayed execution ordering and safety gating

3.4 Universe Model

The implementation is converging on an explicit universe model:

- core perp universe: `""`
- singleton spot universe: `"spot"`
- named HIP-3 perp universes: `dexName`

4 Account and Risk Modes

Hyperliquid separates where assets live from how risk is computed. The important account and risk modes are classic, dex abstraction, unified account, and portfolio margin. These are not separate universes. They are overlays on top of the asset-and-universe model.

5 Solvency and Safety Goals

5.1 Deterministic State Evolution

All validators must reach the same post-block state from the same ordered input stream. That requires the same action decode, same execution order, same response serialization, and same LtHash updates.

The repo now also treats the final app hash as two explicit halves rather than one monolithic digest: an L1 half and an EVM half. Chain-scoped RespHash backends and the split app-hash surface should stay explicit in docs and replay work.

5.2 Margin and Liquidation Safety

Risk-bearing products must not leave losses unaccounted for. The system therefore needs correct maintenance checks, deterministic liquidation triggers, a consistent fallback path when liquidation is insufficient, and ADL where necessary.

5.3 Collateral Conservation

Collateral should not be created by bad transfer routing, incorrect BOLE repayment accounting, incorrect outcome settlement or merge logic, or inconsistent bridge finalization.

5.4 Controlled EVM Re-entry

HyperEVM is not allowed to bypass L1 fairness or ordering constraints. The CoreWriter plus ActionDelayer path exists to slow and serialize EVM-originated actions before they re-enter the main L1 state machine.

6 Liquidation and ADL

Perp and portfolio-margin liquidation are part of the main risk engine. BOLE liquidation is a separate family with its own health-factor and backstop semantics. Outcomes are not ordinary perp-style liquidations; they are primarily a settlement and collateral-conservation problem instead.

7 EVM, Bridge, and Delayed Actions

The EVM is tightly integrated but not sovereign over L1 state. The important bridge points are L1 transfers to and from HyperEVM, CoreWriter-delayed actions, and bridge validator signatures and finalized withdrawal state.

Bridge control is staged through separate withdrawal signatures, finalized-withdrawal votes, validator-set signatures, and finalized validator-set votes. Current repo truth also keeps bridge signing on validator signer keys rather than a detached bridge-only signer family.

The open work on this lane is the control semantics around `enabled`, `delayer_mode`, `status_guard`, and `vty_trackers`, not whether a named delayed-action drain exists.

8 Outcomes and Special Risk Surfaces

The outcomes system introduces question-level metadata, named outcomes, fallback outcomes, settlement transitions, and merge/split/negation-like token mechanics. The leading safety hypothesis in this repository is that the hardest outcome risk sits in question-level reconciliation, especially around fallback and settled named outcomes.

9 Implementation Strategy

This project is trying to reach protocol truth in a controlled way:

1. ingest evidence,
2. promote evidence into claims,
3. implement only confirmed or chain-scoped truths,
4. add regressions,
5. run crate tests and replay/parity checks,
6. sync docs and claims when repository truth moves.

10 Current Boundaries

The hardest parity surfaces still open are exact response-hashing parity across mainnet and testnet paths, deeper per-effect begin-block semantics, some bridge and staking transition details, deeper outcome reconciliation logic, and ADL and liquidation edge semantics.

11 Reading Guide

Recommended reading order:

1. the large Hypurrliquid paper for the broad reverse-engineering reference,
2. this white paper for system design,
3. the yellow paper for protocol truth and execution details,
4. liquidation and block-lifecycle references for risk execution and block flow,
5. findings and status pages for active truth maintenance.