ChatGPT

Hyperliquid System Architecture and Components

Hyperliquid is a layer-1 blockchain purpose-built for on-chain trading. It uses a custom **HyperBFT** Proof-of-Stake consensus (inspired by HotStuff) to achieve sub-second finality [1]. Crucially, the blockchain state is split into **HyperCore** and **HyperEVM** components [1]. HyperCore implements a fully on-chain decentralized DEX and clearing engine (spot and perpetual order books with margin, liquidations, etc.), while HyperEVM provides a general-purpose EVM execution environment on top of the same consensus [1] [2]. These layers are depicted in the Hyperliquid stack diagram below.

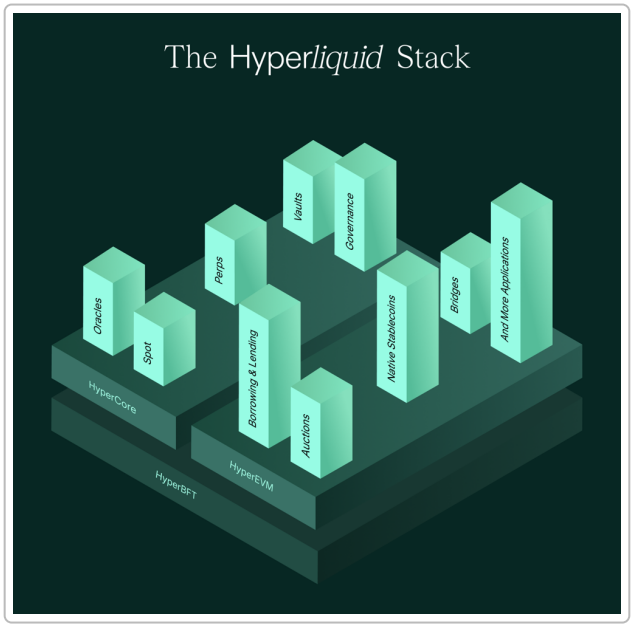

Figure: The Hyperliquid stack – HyperBFT consensus underpins two execution layers: HyperCore (trading engine and financial primitives) and HyperEVM (general smart contracts). Each layer exposes functions (oracles, orderbooks, vaults, lending, governance, bridges, etc.) on the unified chain.

HyperCore and HyperEVM share a single deterministic ledger and consensus, meaning they are *not* separate chains [3] [4]. In practice, blocks contain both HyperCore actions and EVM transactions. Blocks are finalized in one step (by HyperBFT), with no re-orgs, enabling ultra-low latency (≈0.2s median) and high throughput (≈200k+ orders/sec as of today [5]). Overall, HyperCore provides the **high-speed trading engine**, and HyperEVM provides the **rich smart-contract environment**, all on one trust-minimized platform.

HyperCore: Trading Engine and L1 Execution

Role: HyperCore is the native L1 layer that implements Hyperliquid's on-chain **spot markets, perpetual futures, margin accounts, and clearinghouse** logic. It maintains order books, user balances, positions, and vaults entirely on chain. Every order, trade, and liquidation is executed via consensus (no off-chain

matchers) [2] [1]. This design guarantees a single consistent ordering of all trades and one-block finality, maximizing decentralization and fairness.

Architecture: HyperCore is built from first principles for financial throughput. Key modules include:

- **Clearinghouse / Matching Engine:** Manages orderbooks for spot and perpetual markets. Incoming orders (limit, market, IOC, etc.) are matched on-chain, and trades are atomically settled against user margin accounts. No off-chain orderbook is needed [2].
- **Margin and Liquidation Logic:** Tracks collateral, margin tiers, and auto-liquidation. Users' positions are recorded on-chain. Liquidations (and auto-deleveraging) occur via the same consensus-signed transactions as normal trades.
- **Oracles:** On-chain price oracles feed asset prices to perps and lending protocols. Oracles run within HyperCore for minimal latency.
- **Vaults (Strategy Pools):** HyperCore natively supports "vaults", which are on-chain strategy contracts/pools that can execute market-making or other strategies. Depositors share in profits; vault owners earn fees [6].
- **Staking and Governance:** The native token (HYPE) is staked by validators and delegators to secure the network. Staking logic (validator set rotation, rewards) runs on HyperCore.
- **Multi-sig Accounts:** HyperCore natively supports multi-signature accounts as a built-in primitive [7]. An account can be converted to a multi-sig that requires *m-of-n* signatures (configured via special HyperCore actions). This enhances security (e.g. for treasury keys) without relying on EVM contracts.

HyperCore does *not* natively run arbitrary smart-contract code. It supports only predefined actions (place order, cancel, transfer, stake, etc.). This specialization allows extreme optimization. All HyperCore state (balances, orders, oracles, etc.) is committed via HyperBFT, giving strong on-chain guarantees and one-step finality [2] [1]. In summary, **HyperCore serves as the decentralized exchange engine and L1 state machine for financial primitives**, offering deterministic execution and scalability tailored for high-frequency trading [1] [2].

HyperEVM: Smart-Contract Execution Environment

Role: HyperEVM is Hyperliquid's integrated Ethereum-compatible environment. It lets developers deploy standard EVM contracts (ERC-20 tokens, DeFi protocols, NFTs, etc.) directly on Hyperliquid. Unlike a separate chain, HyperEVM runs under the *same* HyperBFT consensus as HyperCore [3] [8]. This means smart contracts can freely interact with HyperCore state and vice versa.

Architecture: HyperEVM has a unique **dual-block architecture** to balance speed and capacity [9]. Two block types alternate in the chain:
- **Small (Fast) Blocks:** Produced very frequently (initially ~every 2 seconds) with a modest gas limit (e.g. 2M gas) [10]. These allow rapid confirmation of transactions, ideal for user-interactive actions. - **Big (Slow) Blocks:** Produced less often (e.g. every 60 seconds) with a much higher gas limit (e.g. 30M gas) [10]. These enable inclusion of large or complex contract calls (large contract deployments, batch operations) that don't need millisecond latency.

Each EVM transaction is flagged (via a per-account user setting) to target either fast or big blocks [10]. Users switch block mode by sending a special action (`evmUserModify`) on HyperCore. By separating block speed

from size, HyperEVM can be upgraded independently in each dimension (e.g. make fast blocks even faster, increase gas in slow blocks) without a single tradeoff [11].

HyperEVM provides full Ethereum APIs (JSON-RPC, Web3 compatibility, accounts, and gas-paid transactions with native HYPE gas). It includes: - **State & Contracts:** An on-chain state root stored in HyperCore blocks. Developers can deploy ERC-20s, oracles, AMMs, NFTs, etc. - **Precompiles/System Contracts:** HyperEVM includes special system interfaces (see below) for reading/writing HyperCore state. - **Wrapped Native Token:** Because HYPE is the native chain token, HyperEVM uses HYPE as "ETH" (native gas). A canonical "Wrapped HYPE" (WHYPE) ERC-20 contract (similar to WETH) is deployed on EVM at a fixed system address. Users can deposit EVM HYPE into this contract and get WHYPE, or withdraw HYPE to/from it [12] [13].

Because HyperEVM is part of the same chain, it can leverage the highly liquid orderbooks and assets of HyperCore directly. For example, on HyperEVM you could call a *read precompile* to get a perp price or spot balance, or invoke a *write system contract* to place a trade on HyperCore [14] [15]. In effect, HyperEVM makes the deep financial primitives of HyperCore available as EVM "building blocks" [8] [16]. (At launch, HyperEVM is in an alpha stage; not all write interfaces are live yet, but core read capabilities are functional [17] [18].)

HyperCore ⇄ HyperEVM Interoperation

Since HyperCore and HyperEVM share consensus, interoperability is native. There is **no inter-chain bridge** needed; instead, assets and messages flow across layers via built-in system contracts and standardized actions [4] [19].

Asset Linking and Transfer

Every HyperCore token ("spot asset") can be linked to an ERC-20 on HyperEVM, enabling frictionless transfers between layers [20] [19]. In practice: - Each HyperCore token (indexed by an asset ID) has a *system address* on the EVM (addresses beginning 0x20... or special HYPE 0x22... address) [21]. - To enable transfers, the token deployer links the Core asset with an EVM ERC-20. The Core side must hold the full supply at the system address and an on-chain "requestEvmContract" action is sent [22]. The EVM contract then proves its intent (via nonce or storage) and a finalizing action completes the link [23]. - **Core → EVM:** A user calls `spotSend` on HyperCore with the destination = the token's system address. Once consensus confirms, a system transaction emits an ERC-20 `transfer(recipient,amount)` on HyperEVM, crediting the user's ERC-20 balance [19]. - **EVM → Core:** A user transfers tokens to the system address on HyperEVM (or calls the *wrapped HYPE* receive). The linked contract emits a standard `Transfer` event. Once 2/3 of validators sign it, a Core transaction credits the user's HyperCore balance [19].

HYPE (the native token) is special. On HyperEVM, HYPE is just the gas currency (no ERC-20 wrapper). To move HYPE from Core → EVM, one simply transfers Core HYPE to the EVM system address, which results in raw HYPE balance on EVM (gas balance). To move HYPE back to Core, one sends an EVM transaction with value to the HYPE system contract (address `0x222…22`); this emits a log event that validators recognize to credit the Core ledger [24]. A canonical **Wrapped HYPE (WHYPE)** ERC-20 exists on EVM (like WETH) for any contract logic needing an ERC-20 token [12] [13].

Messaging and State Access

HyperEVM contracts can call into HyperCore in two ways:

- **Read Precompiles (EVM → Core read-only):** The EVM provides precompiled contracts at addresses `0x...0800+` that let EVM code query HyperCore state. These include methods to read spot balances, perp positions, vault equity, staking delegations, oracle prices, and the latest L1 block number [14]. Calls to these precompiles are included in EVM transactions and incur gas; results are guaranteed to reflect the latest HyperCore state at block time [14]. This allows, for example, a lending contract to fetch XYZ/ABC prices directly from the HyperCore orderbooks via a simple Solidity call [25] [14].
- **System Write Contract (EVM → Core actions):** On a testnet (and soon mainnet), a special system contract at address `0x3333...3333` lets EVM contracts actually *send* HyperCore actions. Through a small Solidity library (L1Write), contracts can invoke HyperCore functions like placing orders (IOC, limit), transferring assets between spot/perp, staking, or vault operations [15]. When such an EVM call executes, it emits a message that the HyperCore runtime processes as a user action, enabling a contract to participate in the DEX. (E.g. an EVM contract could programmatically liquidate an undercollateralized loan by sending orders to HyperCore.)

Because of this tight integration, **HyperCore and HyperEVM form one unified state** [4] [19]. There is effectively **zero bridge risk** between them: asset movements and messages are finalized by the same consensus as any on-chain trade [4]. This is a major advantage over cross-chain bridges: deploying contracts and moving tokens between layers is permissionless and trust-minimized.

Layer Responsibilities and Use Cases

HyperCore and HyperEVM are designed for complementary purposes:

- **HyperCore (Trading Engine / Financial Primitives):** This layer is the backbone of Hyperliquid's exchange. It is optimized for *order execution, matching, and clearing*. Use cases include:
- High-frequency trading bots that place/cancel orders via HyperCore's API or (soon) via the EVM write interface. The sub-second finality makes strategies reliable [2] [1].
- Perpetual futures trading with built-in margin and funding logic. The state machine enforces margin rules and handles liquidations on-chain.
- Custom vault strategies – e.g. a market-maker can run an automated strategy using HyperCore's high-throughput orderbooks and allow others to deposit into the vault [6].
- The native **spot and perp market data** (orderbook, price, etc.) lives here. Any application needing the raw DEX liquidity (e.g. a custom UI or an arbitrage service) interacts with HyperCore.

Importantly, HyperCore is *not* Turing-complete: you cannot deploy arbitrary code here. You can only use its predefined financial operations. This is intentional for performance and safety.

- **HyperEVM (General Smart Contracts):** This layer is the programmable platform. Use cases include:
- **Token issuance and DeFi apps:** Deploy ERC-20 tokens, Automated Market Makers (AMMs), lending protocols, collateralized stablecoins, NFTs, governance DAOs, etc. These contracts operate like on Ethereum but with instant access to HyperCore liquidity.

- **Leveraging DEX primitives:** For example, a new token project can deploy its ERC-20 on HyperEVM and simultaneously launch a spot market on HyperCore in a permissionless auction [26]. Once linked, the token trades immediately on the core orderbook and is also usable in EVM DeFi.
- **Hybrid protocols:** A lending or margin contract could collateralize assets, then use HyperCore's price feeds and orderbooks for risk management [25]. As described in the docs, an EVM lending contract can read real-time prices via precompiles and even liquidate positions by sending orders to HyperCore – all in a few lines of Solidity [25] [14].
- **General Web3 tooling:** Standard wallets, explorers, and developer tooling (Solidity, Ethers.js, etc.) work out-of-the-box on HyperEVM.

In short, **build on HyperEVM when you need complex logic or standard EVM features**, and you want to exploit HyperCore's liquidity as an on-chain primitive [8] [16]. **Use HyperCore (via API or system calls) when your focus is raw trading performance or using the DEX itself**, especially for simple order placement or market-making strategies [2] [1]. Since assets are unified, many projects will use both layers together (e.g. token contract on EVM + trading on Core).

Decentralization and Security

Hyperliquid is a **permissionless PoS network** secured by a BFT consensus and economic stake. Its security guarantees include:

- **HyperBFT Consensus:** A partially synchronous HotStuff-based protocol. Block proposers and committers are validators weighted by staked HYPE [27] [1]. A block is finalized with one round of voting (one-block finality), provided >2/3 of stake signs. The assumption is that less than 1/3 of stake is Byzantine at any time. This gives the typical BFT guarantees: safety unless >1/3 of validators collude [27] [1].
- **Validator Decentralization:** Anyone meeting technical and legal requirements can apply to be a validator in Hyperliquid's delegation program [28]. Validators must self-stake (min 10k HYPE) and run reliable nodes. The Hyper Foundation also delegates to high-quality validators. A distributed set of validators (no single entity) ensures censorship-resistance and security.
- **On-chain Order Execution:** Unlike many "DEXs" with off-chain or centralized matching, Hyperliquid's order books are on-chain. There is *no trusted matcher*; every trade goes through consensus. This removes operational centralization and front-running risk related to private orderflow [2].
- **Asset Custody:** All user balances and positions are recorded on-chain in HyperCore state. There are no external custodians. To withdraw assets out of the system (e.g. to Arbitrum or other chains), there is a multi-sig/bridge process where 2/3 of validators must sign withdrawals [29]. The Hyperliquid bridge contracts (for external networks) have been professionally audited [30]. Within Hyperliquid, "transfers" between Core and EVM are just ledger moves, not external bridges.
- **Native Multi-sig:** As a security feature, HyperCore accounts (including the Protocol's) can be converted to multi-sig, requiring multiple keys to authorize actions [7]. This protects high-value accounts (like treasury wallets) against single-key compromise without leaving the on-chain domain.
- **Audits and Bug Bounty:** Critical components (e.g. bridge contracts) are audited (e.g. by Zellic [30]), and Hyperliquid runs a public bug bounty program. The openness of the code and documentation fosters community review.

Trust Assumptions: Security rests on cryptographic keys and stake-weighted consensus. Users must trust that the validator quorum behaves honestly. Given Hyperliquid's open validator set and delegation, the design's trust assumption is similar to Cosmos or Tendermint networks: two-thirds of voting power must

not collude. Because all trading state is on-chain, users do *not* need to trust any off-chain service for order execution or custody.

In summary, Hyperliquid's decentralization stems from **on-chain enforcement of all rules** under a mature PoS consensus [2] [1]. There is no hidden order book or centralized operator. Validators merely certify transactions and execute state transitions, preserving integrity and fairness.

Developer Guidance: When to Use HyperCore vs HyperEVM

- **Building on HyperEVM:** If your application needs custom smart contracts, tokens, or any Ethereum-like functionality, use HyperEVM. It gives you full Solidity/JSON-RPC tooling and access to HyperCore's liquidity primitives. For example, deploy ERC-20s, AMMs, lending pools, DAOs, oracles, and simply use the built-in precompiles to fetch HyperCore price data or the write contract to execute trades [25] [14]. EVM is ideal for complex logic (e.g. collateral vaults, synthetic assets) that can benefit from Hyperliquid's fast markets. Also, HyperEVM is permissionless and open to all builders, with no integration needed beyond linking assets via HyperCore actions [26].

- **Building via HyperCore:** If you are developing a trading application (exchange UI, algorithmic trader, market-maker) that primarily interacts with order books and on-chain margin, use HyperCore's native interfaces. HyperCore offers REST/WebSocket APIs for placing orders and reading market state (see "API servers" docs). Using HyperCore directly gives the lowest possible latency and highest throughput for trading logic. However, note that you will be limited to HyperCore's existing actions (no arbitrary code). This is suitable for high-frequency bots, liquidity provision strategies, or anything needing the absolute fastest execution.

- **Combining Both:** Many use-cases span both layers. For instance, to list a new token, you might *simultaneously* deploy an ERC-20 on HyperEVM and register the token in HyperCore's spot market. Once linked, traders can immediately trade it on HyperCore, and any EVM app can handle it as a token. Another scenario: a DeFi contract on EVM might trigger trades on HyperCore via the write system contract (e.g. auto-liquidation in lending).

- **Practical Considerations:** HyperEVM is currently in **alpha** on mainnet [17], so some features (like high-throughput writes) are gradually rolling out. Developers should test in Hyperliquid's testnet (with faucet) and stay updated on HIPs and tooling. Importantly, because HyperCore is natively integrated, there are no bridge fees or delays between layers: moving assets is a synchronous on-chain process [4].

Recommendation Summary: Use HyperEVM whenever you need Ethereum-style programmability or to tap HyperCore liquidity from a smart contract [8] [25]. Use HyperCore (via its API or native actions) when you need to execute trades or financial primitives at the highest speed and security [2] [1]. In either case, you benefit from Hyperliquid's one unified protocol – no separate bridging or custodial trust is required.

Sources: All information above is drawn from the official Hyperliquid documentation [1] [2] [9] [3] [4] [19] [14] [15] [12] [13] [7] [30] and reflects the system's design as of 2025.

---

1 8 About Hyperliquid | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs

2 5 27 Overview | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/hypercore/overview

3 4 16 17 18 25 26 HyperEVM | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/hyperevm

6 Vaults | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/hypercore/vaults

7 Multi-sig | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/hypercore/multi-sig

9 10 11 Dual-block architecture | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers/hyperevm/dual-block-architecture

12 13 Wrapped HYPE | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers/hyperevm/wrapped-hype

14 15 Interacting with HyperCore | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers/hyperevm/interacting-with-hypercore

19 20 21 22 23 24 HyperCore <> HyperEVM transfers | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers/hyperevm/hypercore-less-than-greater-than-hyperevm-transfers

28 Delegation program | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/validators/delegation-program

29 Bridge | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/hypercore/bridge

30 Audits | Hyperliquid Docs
https://hyperliquid.gitbook.io/hyperliquid-docs/audits