

Hyperliquid and HyperEVM Complete Developer Guide

Technical architecture and consensus fundamentals

Hyperliquid operates on a dual-layer architecture unified under HyperBFT consensus, [\(Gitbook +2\)](#) achieving remarkable performance with 0.07-second block times and single-block finality. [\(Hyperpc\)](#) [\(Crypto.com\)](#) The system processes up to 200,000 orders per second on HyperCore, [\(Hyperliquid Docs +2\)](#) with theoretical capacity reaching 2 million TPS. [\(Chainstack +7\)](#)

The **HyperBFT consensus algorithm**, inspired by HotStuff, enables Byzantine fault tolerance through pipeline processing. [\(X +2\)](#) The network maintains 21 active validators selected by stake, [\(Hyperliquid Docs\)](#) requiring minimum self-delegation of 10,000 HYPE tokens. [\(LayerZero\)](#) [\(Crypto.com\)](#) Validators must maintain sub-200ms latency to one-third of the validator set by stake weight to ensure optimal performance. [\(GitHub\)](#)

The **unified state architecture** eliminates traditional bridging risks by sharing the same consensus mechanism and state tree between HyperCore and HyperEVM. [\(Gitbook +3\)](#) Processing follows a strict order: L1 block → EVM block → EVM→Core transfers → CoreWriter actions, [\(Gitbook\)](#) ensuring atomic operations across layers. [\(Gitbook +2\)](#)

Network specifications include median end-to-end latency of 0.2 seconds for co-located clients, with 99th percentile at 0.9 seconds. [\(MEXC Blog\)](#) [\(Hyperliquid Docs\)](#) The on-chain mempool accepts only the next 8 nonces per address, automatically pruning transactions older than 24 hours. [\(LayerZero +2\)](#)

HyperEVM technical specifications and capabilities

HyperEVM implements the **Cancun EVM specification** (without blob transactions), [\(Gitbook\)](#) [\(gitbook\)](#) supporting all Cancun hardfork opcodes including TLOAD/TSTORE for transient storage and MCOPY for memory operations. [\(QuickNode\)](#) [\(Gitbook\)](#) The implementation features full EIP-1559 support with a unique twist - both base fees and priority fees are burned, [\(Hyperliquid Docs\)](#) [\(Gitbook\)](#) creating a deflationary gas model. [\(Gitbook +2\)](#)

The **dual-block architecture** optimizes for different use cases: [\(QuillAudits +3\)](#)

```
javascript
```

```
// Small blocks (default)
{
  gasLimit: 2_000_000,
  duration: 1_second,
  useCase: "regular_transactions"
}

// Large blocks (for deployments)
{
  gasLimit: 30_000_000,
  duration: 60_seconds,
  useCase: "contract_deployments",
  activation: {"type": "evmUserModify", "usingBigBlocks": true}
}
```

Medium +2

Precompiled contracts starting at `0x00` provide direct HyperCore access: [Gitbook](#) [gitbook](#)

- User positions, balances, vault information
- Market data including mark and oracle prices
- Staking data with delegations and validator info
- System metrics and L1 block numbers [Gitbook +2](#)

The **CoreWriter contract** at `0x33` enables smart contracts to send actions to HyperCore, [LayerZero +2](#) supporting limit orders, vault transfers, token delegation, and staking operations with approximately 25,000 gas burn plus 47,000 total gas for basic calls. [Gitbook +2](#)

Development environment setup

Required tools and versions

```
bash
```

```
# Node.js 22+ required for Hardhat 3
node --version # Should be v22.x.x or later

# Install Foundry
curl -L https://foundry.paradigm.xyz | bash
foundryup

# Install Hardhat
npm install --save-dev hardhat

# HyperEVM-specific CLI
npm install -g @layerzerolabs/hyperliquid-composer
```

Network configuration

```
javascript

// hardhat.config.js
module.exports = {
  solidity: "0.8.19",
  networks: {
    hyperevm_mainnet: {
      url: "https://rpc.hyperliquid.xyz/evm",
      chainId: 999,
      accounts: [process.env.PRIVATE_KEY]
    },
    hyperevm_testnet: {
      url: "https://rpc.hyperliquid-testnet.xyz/evm",
      chainId: 998,
      accounts: [process.env.PRIVATE_KEY]
    }
  }
};
```

Account activation requirement

Before deployment, activate your address on HyperCore by sending at least \$1 worth of USDC or HYPE. Without activation, block switching and deployment operations will fail with "User or API Wallet does not exist" errors. [LayerZero](#)

Block size management

```
bash
```

Switch to large blocks for deployment

```
npx @layerzerolabs/hyperliquid-composer set-block \  
--size big \  
--network mainnet \  
--private-key $PRIVATE_KEY
```

Deploy contract

```
forge create --rpc-url https://rpc.hyperliquid.xyz/evm \  
--private-key $PRIVATE_KEY \  
src/MyContract.sol:MyContract
```

Switch back to small blocks

```
npx @layerzerolabs/hyperliquid-composer set-block \  
--size small \  
--network mainnet \  
--private-key $PRIVATE_KEY
```

Smart contract development on HyperEVM

Contract deployment workflow

solidity

```
pragma solidity ^0.8.19;
```

}

```
bash
```

bash

```
forge verify-contract <CONTRACT_ADDRESS> \  
src/ContractName.sol:ContractName \  
--chain-id 999 \  
--verifier sourcify \  
--verifier-url https://sourcify.parsec.finance/verify
```

Medium

APIs, SDKs, and libraries

Python SDK usage

```
python  
  
from hyperliquid.info import Info  
from hyperliquid.exchange import Exchange  
from hyperliquid.utils import constants  
  
# Initialize clients  
info = Info(constants.MAINNET_API_URL, skip_ws=False)  
exchange = Exchange(wallet_address, base_url=constants.MAINNET_API_URL)  
  
# Get user state  
user_state = info.user_state("0xYourAddress")  
  
# Place order  
order_result = exchange.order(  
    coin="BTC-PERP",  
    is_buy=True,  
    sz=0.1,  
    limit_px=50000,  
    order_type={"limit": {"tif": "Gtc"}},  
    reduce_only=False  
)
```

TypeScript/JavaScript integration

```
javascript
```

```
const { Hyperliquid } = require('hyperliquid');

const sdk = new Hyperliquid({
  privateKey: process.env.PRIVATE_KEY,
  testnet: false,
  enableWs: true
});

// Place order
const orderResult = await sdk.exchange.placeOrder({
  coin: 'ETH-PERP',
  is_buy: true,
  sz: '1.0',
  limit_px: '3000',
  order_type: { limit: { tif: 'Gtc' } }
});

// Subscribe to real-time updates
sdk.subscriptions.subscribeToAllMids((data) => {
  console.log('Price updates:', data);
});
```

REST API endpoints

```
bash

# Get open orders
curl -X POST https://api.hyperliquid.xyz/info \
  -H "Content-Type: application/json" \
  -d '{"type":"openOrders","user":"0xYourAddress"}'

# Get L2 order book
curl -X POST https://api.hyperliquid.xyz/info \
  -H "Content-Type: application/json" \
  -d '{"type":"l2Book","coin":"BTC-PERP"}'
```

WebSocket subscriptions

```
javascript
```

```
const ws = new WebSocket('wss://api.hyperliquid.xyz/ws');

ws.send(JSON.stringify({
  method: "subscribe",
  subscription: {
    type: "l2Book",
    coin: "ETH-PERP"
  }
}));

ws.send(JSON.stringify({
  method: "subscribe",
  subscription: {
    type: "userEvents",
    user: "0xYourAddress"
  }
}));
```

solidity


```

interface IL1Read {
    function getUserBalance(address user, uint32 asset) external view returns (uint256);
    function getOraclePrice(uint256 tokenIndex) external view returns (uint256);
    function getMarkPrice(uint256 tokenIndex) external view returns (uint256);
}

contract DataReader {
    address constant BALANCE_PRECOMPILE = 0x00000000000000000000000000000000000000000000000000000000000000802;
    address constant ORACLE_PRECOMPILE = 0x00000000000000000000000000000000000000000000000000000000000000807;

    function checkUserPosition(address user) external view returns (uint256, uint256) {
        uint256 balance = IL1Read(BALANCE_PRECOMPILE).getUserBalance(user, 0);
        uint256 price = IL1Read(ORACLE_PRECOMPILE).getOraclePrice(0);
        return (balance, price);
    }
}

```

Testing and debugging approaches

Testing framework setup

```

javascript

// test/HyperEVM.test.js
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("HyperEVM Contract", function () {
    let contract;

    beforeEach(async function () {
        const Contract = await ethers.getContractFactory("HyperEVMContract");
        contract = await Contract.deploy();
        await contract.deployed();
    });

    it("Should read oracle prices", async function () {
        const price = await contract.getOraclePrice(0);
        expect(price).to.be.gt(0);
    });
});

```

Debugging with transaction traces

```

bash

```

Get detailed transaction trace

```
curl https://rpc.hyperliquid.xyz/nanoreth \  
-X POST \  
-H "Content-Type: application/json" \  
--data '{  
  "method":"debug_traceTransaction",  
  "params":["0xTX_HASH"],  
  "id":1,  
  "jsonrpc":"2.0"  
'
```

Use Foundry's debugger

```
forge debug --debug 0xTX_HASH --rpc-url https://rpc.hyperliquid.xyz/evm
```

QuickNode

Gas profiling

bash

Run tests with gas reporting

```
REPORT_GAS=true npx hardhat test
```

Foundry gas report

```
forge test --gas-report
```

Monitor real-time gas prices

```
curl https://api.hyperliquid.xyz/info \  
-d '{"type":"meta"}'
```

Integration methods and patterns

Web3 provider setup

javascript

// Browser with MetaMask

```
const provider = new ethers.BrowserProvider(window.ethereum);  
await provider.send("eth_requestAccounts", []);
```

// Node.js application

```
const provider = new ethers.JsonRpcProvider("https://rpc.hyperliquid.xyz/evm");
```

// Custom configuration

```
const network = {  
  chainId: 999,  
  name: "HyperEVM",  
  ensAddress: null  
};  
const customProvider = new ethers.JsonRpcProvider(  
  "https://rpc.hyperliquid.xyz/evm",  
  network  
);
```

Event monitoring

javascript

// Contract events

```
contract.on("OrderPlaced", (user, asset, amount, event) => {  
  console.log(`Order placed: ${user} ${asset} ${amount}`);  
});
```

// Filter historical events

```
const filter = contract.filters.OrderPlaced(null, "BTC-PERP");  
const events = await contract.queryFilter(filter, -1000);
```

Performance optimization

Gas optimization strategies

solidity

```

contract GasOptimized {
    // Pack struct variables
    struct Order {
        uint128 price; // Packed together
        uint64 quantity; // saves storage slot
        uint32 timestamp; //
        bool isActive; //
    }

    // Use mappings instead of arrays when possible
    mapping(address => Order) public orders;

    // Cache storage values in memory
    function processOrders(address user) external {
        Order memory order = orders[user]; // Single SLOAD
        // Process order using memory variable
    }
}

```

Block type optimization

```

javascript
// Determine appropriate block type
function selectBlockType(gasEstimate) {
    if (gasEstimate < 2_000_000) {
        return "small"; // Use fast 1-second blocks
    } else {
        return "large"; // Use 60-second blocks for complex operations
    }
}

```

Security best practices

Access control implementation

```

solidity

```

```

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/security/Pausable.sol";

contract SecureContract is AccessControl, ReentrancyGuard, Pausable {
    bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");

    constructor() {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    }

    function criticalFunction()
        external
        onlyRole(OPERATOR_ROLE)
        nonReentrant
        whenNotPaused
    {
        // Implementation
    }
}

```

Precompile validation

```

solidity

function validatePrecompileData(address precompile, bytes memory input)
    internal
    view
    returns (bytes memory)
{
    (bool success, bytes memory result) = precompile.staticcall(input);
    require(success, "Precompile call failed");
    require(result.length > 0, "Empty response");
    return result;
}

```

Security audit tools

```

bash

```

Static analysis

slither . --exclude-dependencies

Fuzzing

echidna contract.sol --contract ContractName

Formal verification

certoraRun verification.conf

Common pitfalls and solutions

Block size management issues

javascript

// Problem: Deployment fails with insufficient gas

// Solution: Ensure large blocks are enabled

```
async function safeDeployment(contractFactory) {
```

// Enable large blocks

```
  await sdk.setBlockSize("big");
```

```
  try {
```

```
    const contract = await contractFactory.deploy();
```

```
    await contract.deployed();
```

```
    return contract;
```

```
  } finally {
```

// Always restore to small blocks

```
    await sdk.setBlockSize("small");
```

```
  }
```

```
}
```

Cross-layer state synchronization

solidity

```
// Problem: Assuming immediate state updates
// Solution: Implement proper state checking
contract StateSync {
    uint256 constant SYNC_DELAY = 2; // blocks

    mapping(uint256 => bytes32) public pendingActions;

    function initiateAction() external {
        uint256 actionBlock = block.number + SYNC_DELAY;
        pendingActions[actionBlock] = keccak256(abi.encode(msg.sender, block.timestamp));
    }

    function executeAction(uint256 actionBlock) external {
        require(block.number > actionBlock, "Too early");
        require(pendingActions[actionBlock] != bytes32(0), "No pending action");
        // Execute after delay
    }
}
```

Documentation and developer resources

Essential documentation

- **Official Docs:** <https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers/hyperevm> Hyperliquid Docs
- **Architecture Wiki:** <https://hyperliquid-co.gitbook.io/wiki/architecture/hyperevm>
- **API Reference:** <https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers> Hyperliquid Docs

GitHub repositories

- **Python SDK:** <https://github.com/hyperliquid-dex/hyperliquid-python-sdk> Hyperliquid Docs GitHub
- **Rust SDK:** <https://github.com/hyperliquid-dex/hyperliquid-rust-sdk> Hyperliquid Docs
- **Node Software:** <https://github.com/hyperliquid-dex/node> Hyperliquid Docs
- **Community Resources:** <https://github.com/HyperDevCommunity/AwesomeHyperEVM>

RPC providers

```
javascript

const providers = {
    public: "https://rpc.hyperliquid.xyz/evm",
    quicknode: "https://your-endpoint.hype-mainnet.quicknode.pro/",
    testnet: "https://rpc.hyperliquid-testnet.xyz/evm"
};
```

Block explorers

- **Mainnet:** <https://hypurrscan.io/>
- **Alternative:** <https://hyperevmscan.io/>
- **Testnet:** <https://testnet.purrsec.com/> gitbook

Community channels

- **Discord:** <https://discord.com/invite/hyperliquid> (#builders, #hyperevm) Hyperliquid Docs
- **Ecosystem Projects:** <https://www.hypurr.co/ecosystem-projects> Hyperliquid Docs
- **Analytics:** <https://data.asxn.xyz/dashboard/hyperliquid-ecosystem> Hyperliquid Docs

Development workflow summary

```
bash

# 1. Setup environment
npm init -y
npm install --save-dev hardhat @layerzerolabs/hyperliquid-composer

# 2. Configure networks
echo "PRIVATE_KEY=your_key" > .env

# 3. Activate account on HyperCore
# Send $1+ USDC to your address

# 4. Deploy contract
npx @layerzerolabs/hyperliquid-composer set-block --size big --network mainnet --private-key $PRIVATE_KEY
npx hardhat run scripts/deploy.js --network hyperevm_mainnet
npx @layerzerolabs/hyperliquid-composer set-block --size small --network mainnet --private-key $PRIVATE_KEY

# 5. Verify contract
forge verify-contract ADDRESS src/Contract.sol:Contract --chain-id 999

# 6. Test and monitor
npx hardhat test
```

This comprehensive guide provides all necessary technical information for developing on Hyperliquid and HyperEVM, with practical examples and specific commands that can be directly used in development workflows.