ChatGPT

Monad: Key Technical Innovations for App Builders

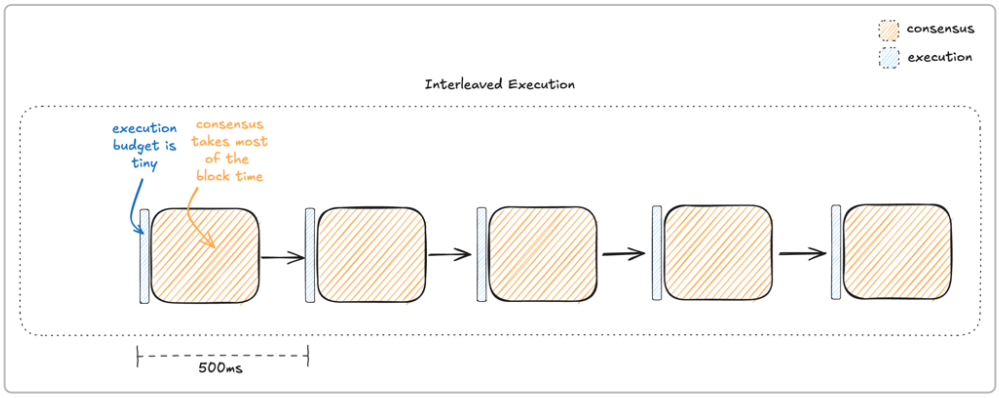
Transaction Costs and Gas Efficiency

- **Ethereum-compatible gas model:** Monad supports EIP-1559 (base fee + priority fee) with the same per-opcode gas costs as Ethereum [1] [2]. In other words, gas units and opcode prices mirror Ethereum, ensuring familiar behavior.
- **Low-fee target:** With a very high block capacity, Monad aims to drive fees near zero. The current testnet base fee is 50 gwei per gas (and will become dynamic), but the high throughput means base fees can remain extremely low under load [3].
- **Gas accounting:** To prevent denial-of-service in its asynchronous model, Monad currently charges *gas limit* rather than actual gas used (i.e. sender pays `value + gas_price * gas_limit`) [4] [5]. (This is a testnet DOS-prevention measure and may be revised.)
- **EIP-1559 ordering:** Transactions are ordered by descending total gas price (base + tip) via a priority auction, just as on Ethereum [2] [6].
- **Larger contracts:** Monad raises the max contract size to 128 KB (vs 24 KB on Ethereum), allowing more complex smart contracts without fragmentation [7].

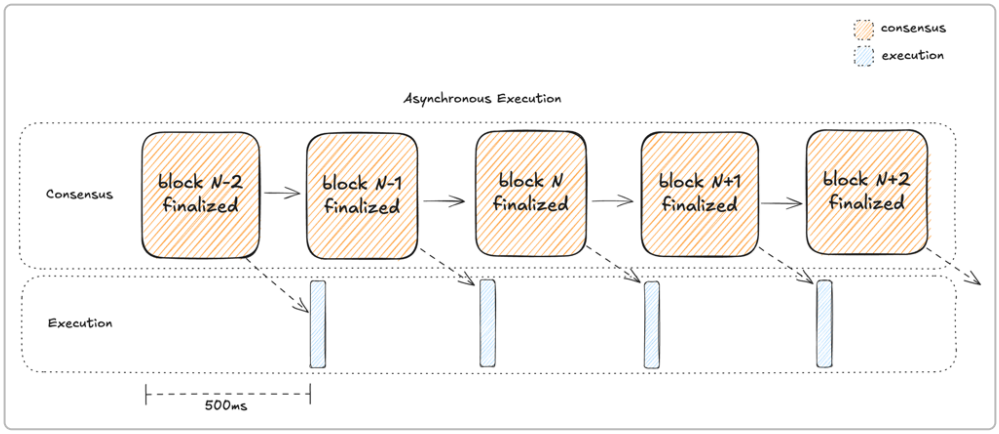
Network Performance (Throughput & Latency)

- **High throughput:** Monad targets throughput on the order of 10,000+ transactions per second [8]. Thanks to pipelining and parallelism, each 500 ms block can include many transactions, far more than Ethereum's 30M-gas (≈100 ms) limit.
- **Low latency:** The block interval (round time) is 500 ms, with *speculative* transaction finality in ~500 ms and *full* finality in about 1 second [9]. In practice this means transactions confirm very quickly.
- **Efficient block propagation:** Monad's **RaptorCast** protocol uses erasure-coded multicast to flood blocks. Leaders encode each new block into chunks and distribute them via a two-level broadcast tree, using the full bandwidth of the network. This dramatically speeds up block delivery while remaining Byzantine-fault-tolerant [10].
- **Optimized mempool:** There is no single global mempool. Instead, transactions sent to a node are forwarded to the upcoming few block leaders (local mempools) [11]. This reduces gossip overhead and latency, helping high throughput.

Architecture and Execution Model



Traditional blockchains (like Ethereum) interleave consensus and execution. In that model, a block proposer must execute all transactions before finalizing a block, and voters must execute them to validate the block. This means only a tiny fraction of each block's time can be spent executing transactions (Ethereum's 12 s block gives only ≈100 ms for execution) [12].



Monad **decouples consensus and execution** in an asynchronous pipeline. Blocks are proposed and agreed upon *without* immediate execution. Instead, transaction execution runs in parallel (slightly lagged) with consensus. This pipelining allows both consensus and execution to occupy the full block interval [13] [14], greatly expanding the compute budget.

- **Asynchronous execution:** Consensus determines transaction ordering first; nodes then execute the agreed-upon blocks. Because execution is "off the hot path," Monad can use the entire block time for computation [14] [13]. This overcomes Ethereum's 1%-of-block-time limit and enables much larger gas-per-block.
- **Parallel (optimistic) transaction execution:** Within each block, Monad runs transactions in parallel on multiple cores. It uses *optimistic concurrency*: transactions start executing before earlier ones finish, and conflicts are detected and retried if needed. Crucially, blocks are still ordered linearly and the final state is **identical** to Ethereum's sequential execution [15] [16]. In short, parallelism is only an internal optimization – semantics and outcomes match Ethereum exactly.
- **Custom State Database (MonadDb):** Monad stores Ethereum's Merkle Patricia Trie data in a purpose-built key-value store optimized for SSDs. MonadDb implements the trie natively (no generic

LSM/B-tree layer) and uses asynchronous I/O (e.g. Linux io_uring) to avoid blocking during state reads/writes [17] [18]. It even optionally bypasses the filesystem, writing directly to block devices to maximize SSD throughput. This design ensures fast state access at scale, further boosting execution performance.

Consensus Mechanism

- **MonadBFT (PoS-BFT):** Monad uses a leader-based, pipelined Byzantine-Fault-Tolerant consensus (a HotStuff variant with research improvements) [19] [20]. Validators stake MON tokens (Proof-of-Stake) for Sybil resistance, and delegation is supported [20]. Under partial synchrony assumptions, the protocol tolerates up to 33% Byzantine stake [19].
- **Pipelined finality:** Consensus proceeds in rounds; each round a new block is proposed and voted on. The protocol allows one block per round by pipelining quorum certificates (QC) on the next proposal. In normal operation, communication is linear in the number of nodes ("fan-out/fan-in") [9]. Timeouts incur quadratic messaging, but in the common case the design is very efficient.
- **Fast blocks and finality:** By default Monad runs 500 ms rounds (blocks), and reaches *full finality* in ~1 s [9]. The pipeline design means that while one block is being proposed, the last block's votes are still being gathered, etc. This achieves extremely low latency for a BFT chain.

Developer Experience and Tooling

- **Ethereum compatibility:** Monad offers full EVM bytecode and RPC compatibility, so developers can use familiar tools. Standard wallets like MetaMask or Phantom work out of-the-box (just point to Monad's RPC URL and chain ID) [21]. Developers can deploy and interact with contracts using Remix, Hardhat, Foundry, Truffle, web3.js/ethers.js, etc., exactly as on Ethereum.
- **Toolchain support:** Popular Solidity frameworks are supported. For example, Foundry and Hardhat (the two leading EVM toolkits) can compile, test, deploy and debug Monad contracts with no custom compiler needed [22]. Quickstarts and guides exist for Remix, Foundry, Hardhat, and others.
- **Infrastructure providers:** Leading RPC & staking services already support Monad Testnet. Providers such as Alchemy, Blockdaemon, QuickNode, Thirdweb, and others offer nodes and APIs for Monad [23]. (Some even have special promotions for Monad projects.) Indexers, explorers and analytics tools are likewise being built or extended for the Monad ecosystem.
- **Developer resources:** Monad publishes full documentation, tutorials, and a testnet explorer. There is an active developer Discord and foundation support for integrations. In short, building on Monad feels very similar to building on Ethereum, with the added benefit of much higher performance.

Ethereum (EVM) Compatibility

- **Full EVM equivalence:** Monad's EVM implementation is bytecode-identical to Ethereum (current Cancun fork). It supports all Ethereum opcodes (including new ones like TLOAD/TSTORE) and produces exactly the same execution results as Ethereum [7] [24].
- **Same account/tx model:** Accounts are identical (20-byte addresses with ECDSA keys) [25]. Transactions use the same typed envelope (EIP-2718) and RLP encoding as Ethereum [26]. Access lists (EIP-2930) and EIP-1559 transactions are supported; blob transactions (EIP-4844) are forthcoming.
- **Linear semantics:** Despite the parallel execution optimizations, blocks and transactions are still processed in a single linear order [16]. This means all on-chain logic (e.g. reverts, event ordering, etc.)

behaves the same as on Ethereum. It is safe to port Solidity contracts without changes – Monad is designed to be a seamless drop-in for EVM apps.
- **Gas and pricing:** Since the execution model is identical, gas accounting works the same way: each opcode has the same cost and EIP-1559 burning/bidding mechanism is in place [1] [2]. Developers can rely on existing understanding of gas usage.

Security and Decentralization Principles

- **Balanced trilemma:** Monad explicitly aims for "speed without sacrifice." It improves scalability and throughput while striving to maintain strong decentralization [27] [28]. The team emphasizes that performance gains **must not** come at the expense of credibility or neutrality [28].
- **BFT safety:** MonadBFT provides production-grade security: once a block is finalized, it is final unless more than one-third of stake is malicious [19]. The pipelined PoS-BFT design protects against equivocation and censorship. In practice this means transactions are extremely unlikely to be reverted once finalized.
- **Decentralization:** By using a PoS model with reasonable stakes and encouraging many validators, Monad seeks a wide validator set. The architecture (e.g. optimized state storage on SSD) is designed to lower hardware barriers so that more participants can run full nodes. This supports censorship-resistance and censorship-neutrality.
- **Resilience:** Features like RaptorCast (for reliable block delivery) and delayed state roots (checks on prior state) add robustness against network faults and misbehaving leaders. In short, Monad's protocol layers are built with Byzantine-robust algorithms at every step.

Summary: For application developers seeking low-cost, high-throughput execution on a well-designed Layer-1, Monad delivers a compelling package. It preserves Ethereum compatibility (same contracts, same tools, same gas model) while innovating under the hood with parallelism, pipelining, and efficient state management. The result is a chain with ~500 ms blocks, ~1 s finality, and on the order of 10,000 TPS [8] [9] – all on an open, PoS-BFT protocol designed around decentralization [28] [19].

Sources: Official Monad documentation [29] [30] [9] [31]. (All features above are drawn from Monad's dev docs.)

---

[1] [16] [25] [26] [31] Other Details | Monad Developer Documentation
https://docs.monad.xyz/monad-arch/other-details

[2] [3] [7] [8] [20] [21] [24] [29] [30] Monad for Developers | Monad Developer Documentation
https://docs.monad.xyz/introduction/monad-for-developers

[4] [6] Gas on Monad | Monad Developer Documentation
https://docs.monad.xyz/developer-essentials/gas-on-monad

[5] [11] Differences between Monad and Ethereum | Monad Developer Documentation
https://docs.monad.xyz/developer-essentials/differences

[9] [19] MonadBFT | Monad Developer Documentation
https://docs.monad.xyz/monad-arch/consensus/monad-bft

[10] RaptorCast | Monad Developer Documentation
https://docs.monad.xyz/monad-arch/consensus/raptorcast

12 13 14 Asynchronous Execution | Monad Developer Documentation
https://docs.monad.xyz/monad-arch/consensus/asynchronous-execution

15 Parallel Execution | Monad Developer Documentation
https://docs.monad.xyz/monad-arch/execution/parallel-execution

17 18 MonadDb | Monad Developer Documentation
https://docs.monad.xyz/monad-arch/execution/monaddb

22 Toolkits | Monad Developer Documentation
https://docs.monad.xyz/tooling-and-infra/toolkits

23 RPC Providers | Monad Developer Documentation
https://docs.monad.xyz/tooling-and-infra/rpc-providers

27 Introduction | Monad Developer Documentation
https://docs.monad.xyz/

28 Why Monad: Decentralization + Performance | Monad Developer Documentation
https://docs.monad.xyz/introduction/why-monad