SATSUMA

A permissioned, Ethereum-compatible Layer 1 for autonomous agents

Open reads. Allowlisted writes, enforced by the protocol itself. Two-second blocks with deterministic finality, stable fees, and a single accountable operator — dedicated infrastructure for software that transacts on its own.

Litepaper
v1.0

Published
July 2026

Author
Satsuma Labs

Network
Chain ID 555555555 · SUMA

ABSTRACT

Satsuma is a sovereign Layer-1 blockchain built to host on-chain protocols for AI agents. It is fully Ethereum-compatible — Solidity contracts deploy unmodified, the network speaks standard Ethereum JSON-RPC, and tools like MetaMask, Hardhat, and Foundry work out of the box. Blocks are produced every 2 seconds and become irreversible within seconds through an explicit finality protocol: no probabilistic confirmations, no reorganizations.

Satsuma is deliberately not an open network. Writes are permissioned inside the protocol itself — only accounts on an on-chain allowlist can transact or deploy contracts — while reads are open to anyone, from anywhere, without registration. The network is operated end to end by Satsuma Labs, a single accountable operator, rather than an anonymous validator set.

This paper describes the network as it operates today: its architecture, access model, trust assumptions, fee economics, and measured performance.

1 Motivation: agents need a different chain

Autonomous agents are becoming economic actors. They hold budgets, pay for services, coordinate with other agents, and execute long-running plans without a human approving each step.

The natural infrastructure for this activity is a blockchain: a shared, neutral ledger with programmable rules, atomic settlement, and a complete audit trail. But public blockchains were designed for a different threat model — anonymous human participants kept honest by economic incentives — and several of their defining properties are actively hostile to autonomous software:

Property	Public L1 / L2	What an agent needs
Participants	Pseudonymous, permissionless	Known, vetted counterparties
Inclusion	Open mempool; front-running and MEV extraction	No anonymous adversaries observing intent
Fees	Auction-driven; volatile by orders of magnitude	Stable and predictable for budget-bound planning
Finality	Probabilistic, or minutes to settle	Deterministic, in seconds
Operations	Best-effort; no accountable party	An operator with a name and an obligation

An agent cannot renegotiate its budget when gas spikes 40×. It cannot reason about a transaction that is "probably final." It should not have to defend its every intention against front-running bots in a public mempool. And when the network misbehaves, "no one is responsible" is not an acceptable answer for production systems.

Satsuma inverts each of these properties. Every account that can write to the chain is known and admitted deliberately. Blocks arrive on a fixed 2-second cadence and finalize deterministically within seconds. Fees follow a smooth, utilization-targeted curve instead of an auction. And a single accountable operator — Satsuma Labs — runs the network and answers for it.

> The result is intentionally not a public commons. It is **dedicated, controlled infrastructure**: a chain whose entire design budget is spent on being a reliable settlement and coordination layer for autonomous software.

---

2 Network architecture

Satsuma is a sovereign Layer 1: it produces its own blocks, finalizes them with its own validator set, and depends on no external chain for security or data availability.

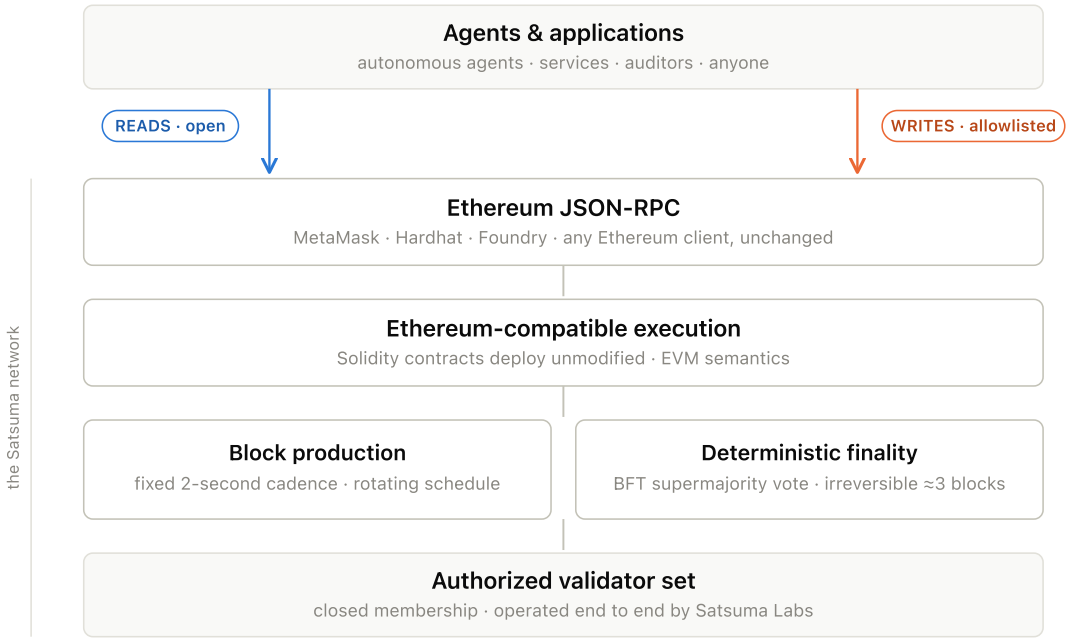

Figure 1 — The Satsuma stack. Open reads and gated writes enter through standard Ethereum JSON-RPC; an EVM execution layer sits on a consensus core that separates 2-second block production from BFT finality, run by an accountable validator set.

2.1 Consensus: fixed-cadence production, explicit finality

Satsuma separates block production from finality — two protocols with two distinct jobs.

Block production rotates on a fixed schedule among a closed set of authorized validators. Each validator knows in advance which slots are its own, so blocks arrive every 2 seconds without leader elections, mining, or stake-weighted lotteries. A fixed cadence is more than a throughput number: it gives agents a stable clock for planning, retries, and timeouts.

Finality runs alongside production as a Byzantine-fault-tolerant voting protocol. Validators vote on the chain they consider canonical, and once more than two-thirds agree, everything up to that point becomes irreversible. Finality is an explicit protocol event, not a probability that grows with confirmations — a finalized block can never be reorganized away.

In measured operation, finality lands within roughly three blocks (about 6 seconds) even under saturating load (§6). An agent that sees its transaction finalized can act on it immediately and unconditionally.

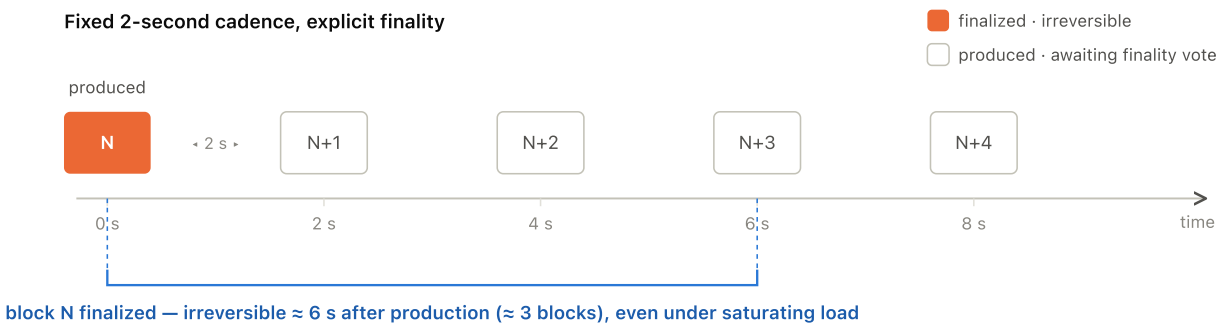

Figure 2 — Production and finality. Blocks are produced every 2 seconds; a supermajority finality vote makes each block irreversible within roughly three blocks of its production.

2.2 Execution: the Ethereum contract you already wrote

- **Solidity contracts deploy unmodified.** No custom language, no transpilation, no non-standard semantics to audit around.
- **Standard Ethereum JSON-RPC.** The network exposes the same interface every Ethereum tool already speaks: `eth_sendRawTransaction`, `eth_call`, `eth_getLogs`, gas estimation, and the rest.
- **The existing toolchain works out of the box.** MetaMask connects as a custom network; Hardhat and Foundry deploy and test against Satsuma exactly as they would against any Ethereum endpoint.

This is a deliberate strategy: the EVM is where the contract tooling, auditor expertise, and — increasingly — agent frameworks already are. Satsuma changes *who may transact* and *how the network is run*, not *how contracts are written*.

2.3 Chain parameters

Parameter	Value
Network type	Sovereign Layer 1 · permissioned writes, open reads
Execution	Ethereum-compatible (Solidity, EVM semantics)
Interface	Ethereum JSON-RPC
Chain ID (EIP-155)	`555555555`
Native token	SUMA · 18 decimals · gas and settlement
Block time	2 seconds, fixed cadence
Finality	Deterministic BFT · typically within ~3 blocks (~6 s)
Validators	Closed, authorized set operated by Satsuma Labs

---

3 The access model

The access model is Satsuma's central design decision: **open reads, permissioned writes — enforced by the protocol itself.**

3.1 Writes are gated in the protocol, not at the door

Most “private” chains restrict access at the network edge — a firewalled RPC endpoint, an API key, a VPN. Satsuma does not rely on any of that. Admission is checked **inside transaction validation**, the same protocol stage that verifies signatures:

- A transaction arrives at any node, from any source.
- Its signature is verified as usual.
- The sender is checked against the **on-chain allowlist**. If the sender is not a member, the transaction is rejected on the spot — it never enters the mempool, never propagates to other nodes, and never occupies block space.

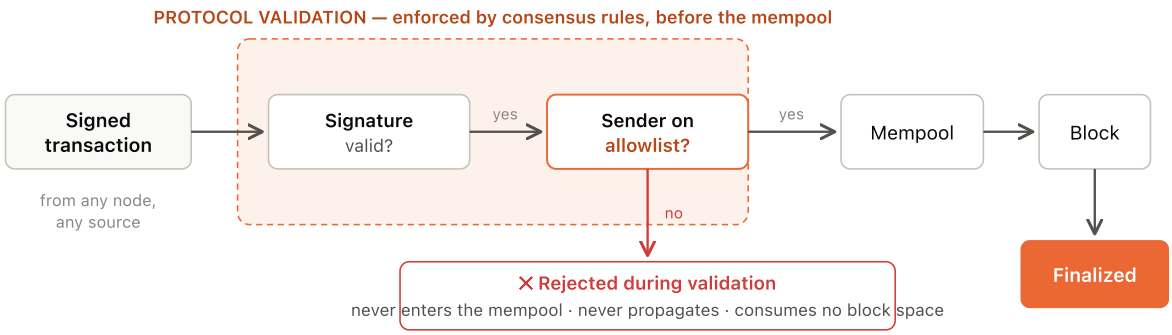

Figure 3 — Transaction admission. Signature verification and the protocol-level allowlist check both happen before the mempool. A non-allowlisted transaction is rejected at the door of the protocol, not at a network firewall.

> Because enforcement lives in the protocol, exposing a public RPC endpoint does not open the chain to writes. There is no perimeter to misconfigure: a transaction from a non-admitted account is invalid **by consensus rules**, no matter which node it reaches or how it got there.

3.2 Reads are open to everyone

Anything that does not modify state is unrestricted: state queries, contract view calls, `eth_call` simulation, gas estimation, log and event queries. Any developer, auditor, or agent can inspect the full state of the chain without an account or an API key. Two consequences are worth stating plainly:

- **Transparency is a feature.** Counterparties and observers can independently verify every balance, every contract, and every transaction ever executed. A permissioned chain does not have to be an opaque one.
- **The chain keeps no secrets.** All on-chain state is publicly readable. Applications that need confidentiality must encrypt at the application layer before writing.

3.3 Contract deployment is doubly gated

Deploying code is gated twice: the deployment transaction must pass the standard admission check like any other write, **and** the deployment operation itself independently requires allowlist membership. Even if transaction-level gating were ever relaxed, the ability to introduce new code on the chain would remain restricted.

3.4 Administration without lockout

The allowlist lives on-chain and is managed by the network's root authority (§4). Membership changes are ordinary, auditable on-chain operations — additions and removals are visible in the public record like any other transaction.

The root authority itself is always permitted to transact. This is a deliberate safety property: no allowlist change, however erroneous, can lock the administrator out of the network's own controls.

---

4 Trust and security model

Satsuma's trust model is different from a public chain's — and the difference is the point. We state it explicitly, because a permissioned network that is vague about its trust assumptions does not deserve anyone's workload.

What users trust. Satsuma Labs operates the validator set, manages the allowlist, and holds the network's root authority, which can change protocol rules, adjust membership, and administer the treasury. Using Satsuma means trusting Satsuma Labs to operate the network honestly and competently — in the same sense that using a cloud provider means trusting its operator, with one critical difference: every action taken on Satsuma is recorded on a publicly readable ledger.

What the operator cannot do. The protocol enforces real limits even on the root authority as it stands today:

- **It cannot forge transactions.** Every transaction requires a valid signature from the sending account's key. Neither validators nor the root authority can fabricate a transaction *from* an account they do not control.
- **It cannot act invisibly.** All state and all history are publicly readable (§3.2). Administrative operations — allowlist changes, protocol upgrades, treasury movements — land in the same auditable record as everything else.
- **It cannot rewrite finalized history undetected.** Finalized blocks are cryptographically chained and continuously observed by anyone reading the chain. Tampering with history would be immediately evident to every reader and every tooling integration.

The honest asymmetry. The root authority can change the protocol itself, and a protocol change could in principle alter any of the above. This is disclosed, not hidden: the guarantee Satsuma offers is not “trustless,” it is **accountable** — a named operator, acting on a public ledger, with every administrative action visible to the people it affects. For agent workloads that today run against entirely opaque APIs, this is a strict upgrade in verifiability; for workloads that require trustlessness against their own operator, a public chain remains the right tool, and we say so.

Network security. The write path is protected by the admission model itself: with no anonymous participants, the network's attack surface shifts from “anyone on the internet” to a finite, known, revocable set of actors. The read path is stateless with respect to consensus and can be scaled and hardened independently of the validators.

5 Fee economics

Fees on Satsuma exist to meter resources and keep the network healthy — not to run an auction for scarce block space.

How a fee is computed. Each transaction pays in proportion to the resources it actually consumes: a compute-and-state component that scales with execution cost, plus a size component per byte of transaction data. Fees are paid in SUMA, the native token.

Stability by design. Instead of a bidding market, Satsuma uses a smooth dynamic multiplier that targets **25% average block utilization**. When sustained load pushes blocks above the target, fees rise gradually; when load falls, they decay back. There are no gas auctions and no sudden spikes driven by someone else's activity — the fee an agent estimates is, in ordinary operation, the fee it pays. Priority tips are supported but not required.

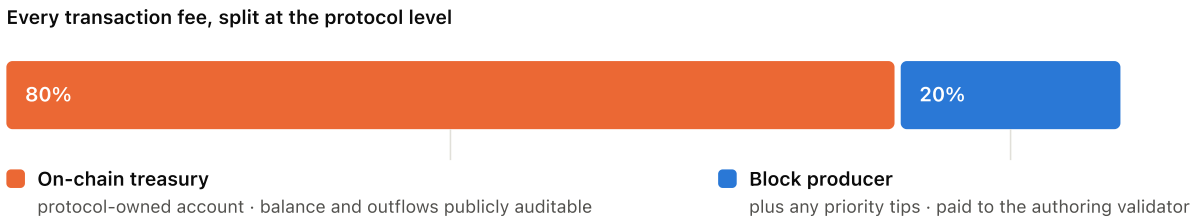

Figure 4 — Protocol-level fee split. 80% of every transaction fee accrues to the on-chain treasury; 20%, plus any priority tip, goes to the block producer.

SUMA is a network resource, not a speculative instrument: it meters usage, settles fees, and funds operations. There is no public sale and no yield program.

6 Measured performance

The numbers below are measurements of the network as it exists, not projections.

They were produced by a reproducible benchmark harness on commodity cloud hardware (8-vCPU servers, one validator per server), with pinned software versions, repeated runs, and results reported as medians with confidence intervals. We publish them with their conditions because performance claims without conditions are marketing.

Block time	Finality	Read throughput	Read latency, p99
2 s	**~6** s	**14,300** queries/s	**4** ms
fixed cadence, held under saturating load	deterministic, ≈3 blocks, under load	eth_call storm · single RPC server · median	p50 2 ms · p95 3 ms

Three observations for context:

- **Reads are effectively free and fast.** Agents are read-heavy — polling state, simulating calls, watching events — and the read path sustains five-digit query rates at single-digit-millisecond latency on one commodity server, scaling horizontally by adding endpoints.
- **The current ceilings are policy and software, not hardware.** Under saturating write load, validator hardware remains far from exhausted; throughput is bounded by deliberately conservative per-block resource budgets and by a contract-execution path that today runs single-threaded. Controlled experiments confirm block capacity scales linearly when those budgets are raised, and profiling shows individual transactions are metered several times more conservatively than their real cost.
- **Capacity grows with demand, not with headlines.** Because the workload is known and admitted (§3), capacity planning is an engineering exercise against real requirements rather than provisioning for adversarial spam. Raising throughput is a calibration program on the identified bounds — not a redesign.

---

7 Roadmap

Satsuma is live and operating as described. Near-term direction, stated conservatively:

- **First tenants.** Agent-oriented protocols deployed as Solidity contracts, beginning with autonomous-development infrastructure — DAO-governed coordination between human maintainers and AI agents, operated by Satsuma Labs.
- **Performance calibration.** Recalibrating the conservative resource budgets identified in §6 against measured costs, raising write throughput within the existing 2-second cadence.
- **Operational hardening.** Expanding the validator topology and read infrastructure as tenant workloads grow.

> The access model, the fee design, and the accountability model described in this paper are not transitional — **they are the product.**

Satsuma is designed, built, and operated by **Satsuma Labs**.
Network explorer: explorer.satsuma.one · Engineering blog: engineering.satsuma.one · Contact: satsuma.one

This document describes the Satsuma network as of July 2026. Parameters and measurements are current as of publication; the canonical, up-to-date values are those observable on-chain.