Sage Protocol: Decentralized Governance for Agent Context

Velinus Sage

`sage-protocol.io`

October 30, 2025

Abstract

We present Sage, a decentralized protocol for coordinating agent context through community governance. Autonomous agents depend on context—system prompts, tool descriptions, and behavioral instructions—to operate effectively. Current approaches lack coordination mechanisms, resulting in fragmented knowledge, unclear provenance, and misaligned incentives for quality improvement.

Sage combines three technical primitives: (1) content-addressed storage via IPFS for immutable, verifiable context hosting, (2) on-chain governance through SubDAO contracts enabling transparent community curation, and (3) standardized discovery via Model Context Protocol servers with cryptographic verification. This architecture enables contributors to publish context improvements, communities to vote on adoptions through token-weighted governance, and agents to retrieve approved versions with automatic propagation of updates.

We describe the protocol architecture, present the IPFS worker implementation for authenticated uploads and access control, detail the governance mechanisms and economic incentive design, and provide security analysis for production deployment. A reference implementation deployed on Base Sepolia demonstrates operational characteristics.

Contents

1 Introduction

Autonomous agents require well-defined context to operate reliably. Current approaches to managing agent context lack coordination mechanisms, leading to duplicated work, inconsistent versioning, and unclear provenance. This paper presents Sage, a decentralized protocol that addresses these coordination failures through on-chain governance, content-addressed storage, and standardized discovery interfaces.

We describe: (1) the coordination problem in agent context management, (2) the protocol architecture combining smart contracts, IPFS storage, and indexed discovery, (3) governance mechanisms enabling community curation, (4) economic incentives aligning contributor and consumer interests, and (5) security considerations for production deployment. Reference implementations on Base Sepolia testnet demonstrate the system's operational characteristics.

2 Problem Statement

2.1 Coordination Failures in Agent Context

Agents succeed or fail on the quality of their context. Today, prompts drift across wikis and repos, provenance is unclear, and teams repeatedly reinvent the same work. There is no shared, governed source of truth for what works in each domain. This represents a significant market inefficiency in the rapidly growing generative AI ecosystem, which McKinsey estimates could contribute up to $4.4 trillion annually to the global economy.

Agent effectiveness depends on context quality—system prompts, tool descriptions, and behavioral instructions. Current management approaches exhibit three coordination failures:

Information silos. Domain expertise remains isolated in individual repositories, private documentation, and proprietary configurations. When practitioners develop effective context patterns, knowledge transfer occurs through informal channels or not at all. This results in duplicated effort and inconsistent quality across similar applications.

Version inconsistency. No canonical versioning system exists for agent context. Multiple variants of nominally identical prompts circulate without clear provenance or rationale for differences. Updates propagate slowly and unpredictably, causing drift between production systems.

Misaligned incentives. Contributors who develop high-quality context receive no systematic compensation when others adopt their work. This underinvestment in public goods reduces the aggregate quality of available context below socially optimal levels.

2.2 Domain Heterogeneity

Context requirements vary significantly across application domains. Security analysis requires different instruction patterns than creative generation. Even within domains, optimal approaches remain contested and evolve over time. Effective solutions must accommodate subjective preferences while enabling communities to converge on shared standards through transparent processes.

2.3 Requirements for Solutions

Addressing these coordination failures requires:

- **Provenance tracking**: Cryptographic verification of context origin and modification history
- **Discovery mechanisms**: Standardized interfaces for agents and developers to locate current approved versions
- **Governance processes**: Transparent decision-making for context updates with stakeholder representation
- **Economic incentives**: Compensation mechanisms that reward contributors proportional to adoption and usage

---

3 Protocol Design

3.1 System Architecture

Sage implements a three-layer architecture that separates storage, governance, and discovery concerns while maintaining cryptographic integrity across layer boundaries. The storage layer provides content-addressed hosting via IPFS with authenticated upload infrastructure. The governance layer coordinates community curation through smart contracts deployed on Ethereum Layer 2 (Base). The discovery layer exposes multiple interfaces—Model Context Protocol servers for autonomous agents, command-line tools for developers, and web applications for community members—all querying a common indexed state.

This separation enables independent scaling and evolution of each layer. Storage infrastructure can migrate between IPFS implementations or add redundancy without affecting governance logic. Governance contracts remain chain-agnostic, supporting future L2 deployments through factory patterns. Discovery interfaces share backend infrastructure while presenting domain-specific abstractions: agents receive JSON-RPC responses, CLI users interact through shell commands, and web users navigate visual interfaces. All interfaces verify critical operations against on-chain state, preventing indexed data from introducing trust assumptions.

The protocol coordinates these layers through content identifiers (CIDs) that serve as cryptographic bridges. When contributors upload context manifests to IPFS, they receive CIDs derived from SHA-256 hashes of the content. Governance proposals reference these CIDs, enabling community voting without requiring on-chain storage of large files. After approval, PromptRegistry contracts record CID-to-library mappings, creating verifiable links between governance decisions and content. Discovery services resolve library identifiers to current CIDs, fetch content from IPFS, and verify that computed hashes match on-chain records before returning results. Figure 1 illustrates the system architecture and data flows.

3.2 Smart Contract Layer

The governance layer implements standard OpenZeppelin patterns adapted for context curation. Each SubDAO deploys four core contracts: a Governor implementing token-weighted voting, a Timelock enforcing execution delays, a PromptRegistry maintaining CID-to-prompt mappings, and a Treasury (Gnosis Safe) for community funds. The SubDAOFactory contract deterministically deploys these components with configurable parameters for voting periods, quorum thresholds, and timelock delays.

The PromptRegistry contract provides the canonical on-chain record of approved context. It implements OpenZeppelin's upgradeable proxy pattern with AccessControl for role-based permissions. Storage mappings index prompts by CID string keys, maintaining metadata including version numbers, timestamps, author addresses, and fork relationships. Discovery indices enable efficient queries by tag (keccak256 hash), creator address, and category enumeration. Anti-abuse mechanisms include submission cooldowns, maximum tag limits per prompt, and duplicate CID prevention. The registry emits `PromptProposalUpdated` events linking proposal IDs to CID updates, enabling subgraph indexers to construct complete governance histories.

Access control follows a role hierarchy defined in a shared Roles library: `GOVERNANCE_ROLE` for proposal execution, `MODERATOR_ROLE` for council actions, `REGISTRY_ROLE` for registry administration, and `EMERGENCY_ROLE` for pause functionality. Fork policies (OPEN, GOV_ONLY, DISABLED) determine whether prompts can be remixed without governance approval, with parent-child relationships preserved on-chain for attribution. The upgradeable architecture allows communities to add discovery indices or metadata fields through governance proposals without redeploying registries or migrating existing data.

3.3 Storage Infrastructure

Content storage and upload authentication operate through a Cloudflare Worker that mediates between contributors and IPFS infrastructure. The worker implements a stateless authentication flow based on Sign-In with Ethereum (SIWE): contributors request nonce challenges, sign structured messages with their wallets, and present signatures with subsequent requests. The worker verifies signatures by recovering signer addresses from message hashes, then validates authorization by querying the role contract's `hasRole(bytes32 role, address account)` function via blockchain RPC. Verification results cache in Cloudflare KV with configurable

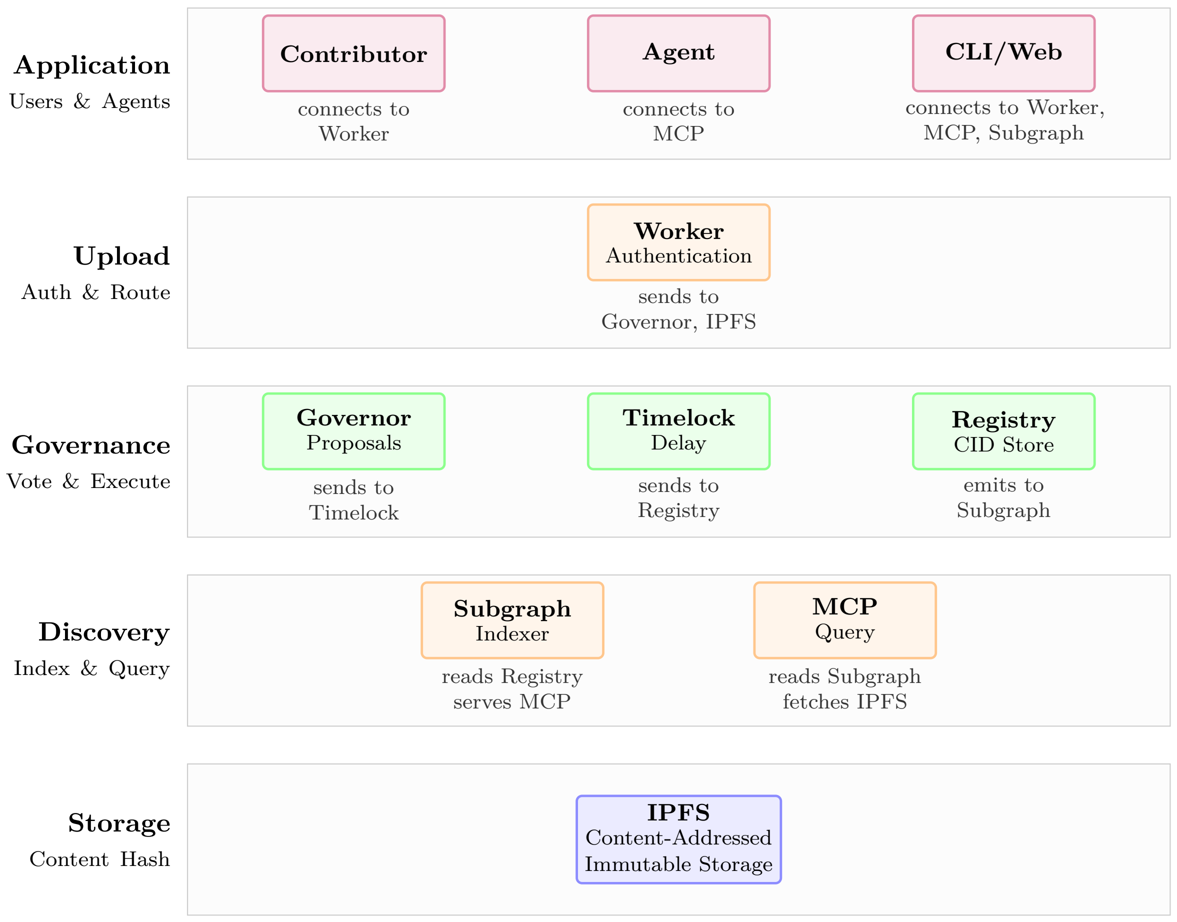

Figure 1: Layered protocol architecture. Application layer: entry points for contributors, agents, and CLI/Web. Upload layer: Worker authenticates and routes. Governance layer: proposals flow through Governor, Timelock, and Registry. Discovery layer: Subgraph indexes events, MCP serves queries. Storage layer: IPFS provides immutable content hosting.

TTLs to balance security and performance, with separate TTL values for positive results (extended), negative results (short to allow rapid remediation), and fallback allowlist entries.

Upon successful authentication, the worker accepts content uploads as multipart form data and forwards them to a Kubo HTTP API endpoint with pinning enabled. The Kubo node returns content identifiers following IPFS conventions (CIDv1 with SHA-256 hashing), which the worker returns to clients. The worker supports pin management operations (pin, unpin, check status), gateway warming by prefetching content through public gateways to populate CDN caches, and optional backup to Cloudflare R2 object storage for redundancy. Payment integration follows the Coinbase x402 protocol specification, returning HTTP 402 responses with EVM payment instructions when contributors lack sufficient credits. The worker implements Durable Objects for transactional credit ledger updates and prompt commerce entitlement tracking, providing single-writer consistency guarantees across distributed worker invocations.

The worker architecture separates concerns through a router layer handling HTTP verb and path matching, an authentication layer managing SIWE and caching, service modules wrapping external APIs (IPFS, RPC, R2), and Durable Objects providing stateful storage. The worker returns only JSON responses with sanitized error messages, preventing information disclosure about internal infrastructure or authorization logic.

3.4 Discovery Mechanisms

The protocol exposes three discovery surfaces optimized for different integration patterns: autonomous agents, developer workflows, and community participation. These surfaces share common backend infrastructure (subgraph indexing, RPC verification, IPFS gateways) while presenting domain-appropriate abstractions.

3.4.1 Agent-Native MCP Integration

The Model Context Protocol server provides standardized JSON-RPC endpoints for agent integration following the MCP specification. Agents query libraries by identifier strings, receiving resolved CIDs from indexed SubDAO state. The server implements dual transport modes: STDIO for local desktop agents (e.g., Claude Desktop integrations) and HTTP with API key authentication for hosted services.

Discovery tools: `search_prompts`, `trending_prompts`, `search_onchain_prompts`, `list_libraries`, `get_prompt_content`, `get_prompts_from_manifest`, `list_subdaos`, `get_library_manifests`.

Meta-prompting tools: `start_metaprompt_interview`, `save_metaprompt`, `list_metaprompts`, `get_metaprompt`, `generate_metaprompt_link`.

Publishing tools: `validate_manifest`, `push_manifest_ipfs`, `propose_manifest`, `publish_manifest_flow`.

The server maintains a SQLite cache of library-to-SubDAO bindings and fetched manifests, with configurable TTLs and cache invalidation on observed state changes. When serving queries, the server resolves library IDs to SubDAO addresses through LibraryRegistry lookups, queries the SubDAO's PromptRegistry for current CIDs, fetches manifests from IPFS gateways, and verifies content hashes match on-chain records before returning results. This verification step prevents compromised indexers or gateways from serving malicious content undetected. JSON schemas mirror manifest structure, enabling agents to render instructions without custom adapters.

3.4.2 Meta-Prompting as Onboarding

New contributors generate high-quality system prompts inside their chat tools without documentation: guided interview (`start_metaprompt_interview`) collects goals, constraints, tone, and guardrails. Results save as Markdown to `.sage/metaprompts` via `save_metaprompt`. One-click sharing via `generate_metaprompt_link` produces ChatGPT launch URLs with embedded system prompts. To publish for reuse, agents use `push_manifest_ipfs` and `publish_manifest_flow` to build governance proposals upgrading library manifests. CIDs remain canonical references across CLI, web, and MCP interfaces.

3.4.3 CLI and Web Interfaces

The command-line interface wraps the protocol SDK to provide scriptable access for developers and power users. Configuration follows a profile-based system: `.sage/config.json` files define named profiles containing network settings, contract addresses, feature flags, and IPFS provider preferences. Secrets (IPFS upload tokens, RPC API keys) store in OS keychains via the keytar library. The CLI integrates Cast keystores from Foundry for wallet management. Upload operations default to the Sage IPFS worker, with contributors authenticating via SIWE challenges signed by their connected wallets. Command output formats as human-readable tables for interactive use and JSON for scripting.

The web application provides visual interfaces for community governance built on Next.js with server-side rendering. Authentication integrates Coinbase CDP for smart contract accounts and Privy for social login, supporting both EOA and account abstraction flows. The application queries Graph Protocol subgraphs for library discovery, proposal histories, and voting records, falling back to RPC calls for critical verification. Users explore prompt libraries through filterable grids showing metadata, fork relationships, and governance status. The publish wizard guides manifest uploads through IPFS, preflight validation checking voter power and quorum requirements, and proposal submission to Governor contracts.

3.5 Operational Workflows

The protocol implements a publish-curate-discover cycle coordinating contributors, governance participants, and context consumers. Publication begins with contributors creating prompt manifests in local `.sage/prompts/` directories using the CLI or web interface. Manifests encode prompt metadata (title, description, tags), file references (system prompts, tool descriptions), and optional access control policies. Contributors upload manifests to IPFS through the authenticated worker API, signing SIWE challenges to prove wallet control. The worker validates authorization through on-chain role checks, forwards content to Kubo with pinning enabled, and returns CIDs. Contributors then create governance proposals referencing these CIDs, providing human-readable descriptions and specifying target contracts (PromptRegistry) and function calls (`updatePromptByProposal`).

Curation proceeds through standard OpenZeppelin Governor flows. Proposals enter a delay period (typically several blocks) before voting opens, allowing observers to review changes. Token holders cast votes through Governor contract calls, with support options Against (0), For (1), and Abstain (2). Votes weight by token balance at a snapshot block height recorded when proposals created, preventing vote buying after proposal submission. Proposals reaching quorum thresholds transition to Succeeded state, enabling queueing through Timelock contracts. Timelocks impose configurable execution delays (24-48 hours recommended for production), providing observation windows before state changes. After delays elapse, anyone can trigger execution by calling the Governor's execute function, which dispatches calls to target contracts. PromptRegistry contracts update CID mappings and emit `PromptProposalUpdated` events linking proposal IDs to content hashes, enabling complete governance audit trails.

Discovery serves approved CIDs through multiple channels. Agents polling MCP servers receive updated manifests automatically, with servers caching results and invalidating on observed proposal executions. CLI users query registries directly or through subgraph GraphQL endpoints, with commands supporting filtering by tag, creator, or category. Web users browse visual interfaces with real-time updates from subgraph subscriptions. All discovery paths verify fetched content by computing SHA-256 hashes and comparing against on-chain CID records, ensuring cryptographic integrity regardless of serving infrastructure. Fork operations create new prompts with parent references preserved on-chain, enabling attribution tracking through arbitrary remix depths. Usage metrics optionally increment through registry function calls, providing signals for quality assessment without compromising privacy when disabled.

3.6 Funding Experimentation: Bounties & SubDAO Treasuries

Communities don't just vote, they pool funds to compensate prompt experimentation and reward contributors.

Bounties for targeted improvement. A SubDAO identifies a gap: "We need better tool descriptions for financial data agents." They create a bounty via their treasury, offering 5,000 SXXX for the best submission. Multiple contributors submit proposals with test results showing improved agent outputs. The community votes on the winner, and the bounty pays out automatically from the SubDAO treasury. The winning prompt becomes the new approved version.

Treasury-funded experiments:

SubDAOs maintain their own treasuries (Safe multisigs) to fund prompt experimentation. Communities can:

SubDAOs maintain treasuries to fund prompt experimentation through multiple mechanisms. Communities offer bounties for specific prompt challenges, grant funding to researchers developing new context patterns, reward contributors whose prompts get adopted by other SubDAOs through forks, and purchase premium prompts from creators to make them available to members. This treasury-based funding model aligns economic incentives with quality improvements.

Economic alignment:

When your prompts prove valuable, such as when other SubDAOs fork them, agents use them extensively, or communities vote to adopt them, you earn rewards from the originating SubDAO's treasury. This creates a feedback loop where successful prompt engineers capture upside from the economic value their work generates.

3.7 Key Differentiators

Feature	Description
Local-to-Global Workflow	Test prompts privately with agents before publishing to SubDAOs for governance. Enables experimentation without public pressure while ensuring community validation.
Immutable Provenance	Every prompt includes on-chain metadata: governing SubDAO, author, proposal ID, vote tally, timestamp. Complete verification eliminates trust assumptions.
Multi-Modal Discovery	**Agents:** MCP endpoints; **Engineers:** CLI tools; **Communities:** Governance voting; **Developers:** Direct IPFS links
Economic Alignment	Contributors earn from treasuries for successful prompts. Fork relationships preserve credit for original authors while enabling composability.
Fork-Native Coordination	Create on-chain forks with clear attribution. Remix for security, domain translation, or model optimization while preserving credit flow and enabling network effects.

Table 1: Core differentiators of the Sage Protocol compared to traditional prompt management approaches.

3.8 Living Context Infrastructure

Instead of static prompt files that decay over time, Sage provides:

- **Self-improving libraries** that evolve through community experimentation and usage
- **Transparent governance history** allowing complete provenance tracking
- **Seamless agent integration** with automatic discovery and updates
- **Flexible sharing mechanisms** via CLI, web interfaces, or direct IPFS links
- **Sustainable economics** that reward contributors and fund continued innovation
- **Composable knowledge networks** where improvements build upon proven work

---

4 Governance

Sage's value derives from continuous improvement through community governance. Prompts don't merely get published—they are tested, voted on, forked, improved, and funded through transparent on-chain processes. This creates a self-improving loop where better context attracts more usage, which generates signals, which funds experimentation, which produces even better context.

4.1 Governance Architecture

SubDAOs control their own context. Each community creates a SubDAO (Governor + Timelock + Treasury + PromptRegistry) that governs its prompt libraries. Token holders propose changes, vote on proposals, and execute updates through timelocks. This keeps curation transparent and decentralized with no single entity controlling what agents consume.

Timelock safety: All governance changes queue through timelocks (configurable, typically 24h+ on mainnet). This provides review windows before execution and an escape hatch—if a malicious proposal passes, token holders or authorized roles can take corrective action during the delay window (cancel, pause, or supersede).

Provenance is automatic: Every approved prompt library includes on-chain provenance: governing SubDAO, proposal ID, vote tally, timestamp, and CID hash. Agents verify content hashes match on-chain

CIDs before execution. Fork relationships are preserved on-chain, ensuring original authors receive credit when their work is remixed.

4.2 Governance Modes

Sage supports three governance modes based on community maturity and risk tolerance. Communities can migrate between modes via timelocked proposals.

Token Mode (Default): Standard token-weighted voting with quorum requirements. Anyone with governance tokens can propose changes. Proposals reaching quorum queue through timelocks and execute automatically. Optimal for mature communities valuing broad input and transparent legitimacy.

Operator Mode (Council): Curated councils (Safe multisig) propose and execute within timelock guardrails without token voting. Councils holding ApproverCouncil badges can perform limited moderator actions, but prompt updates always require full governance through timelocks. Suitable for fast-moving teams requiring specialized moderation with clear accountability.

Hybrid Mode (Policy + Council): Token holders approve policy envelopes defining allowed operations, parameter bounds, budgets, and rate limits. Operators execute pre-approved changes within envelope constraints. Actions outside the envelope require full token votes. Optimal for production operations where experts handle routine changes while communities maintain hard guardrails.

Migration Path: Start with Operator mode for bootstrap and incident response. Migrate to Hybrid as scope stabilizes. Graduate to Token mode as participation grows.

4.3 The Self-Improving Loop

Four stages create continuous improvement:

Publish: Contributors create prompts locally, test against real workloads, and publish to SubDAOs. The CLI pushes manifests to IPFS and creates governance proposals. Metadata captures authorship, version lineage, and optional premium access controls.

Curate: Communities review proposals—diff against current versions, test with agents, debate trade-offs. Token holders vote. Proposals that pass queue through timelocks and execute, updating the on-chain registry to point at the new CID. Manifest diffs and CLI previews provide auditable context before execution.

Reward: Contributors earn through multiple rails. Boosts incentivize voting participation. Bounties fund targeted experiments. Premium prompts generate creator royalties (70/20/10 split: SubDAO/protocol/creator, adjustable via governance). Paid pinning routes revenue to treasuries. Soulbound badges record contribution history on-chain.

Measure: Agents fetch prompts via MCP and execute tasks. Communities collect signals: proposal outcomes, fork counts, premium receipts, bounty completions, and optional usage metrics. These signals inform the next curation cycle.

The loop repeats. Better prompts attract more adoption, generating more signals, funding more experimentation, producing even better context. Quality compounds over time instead of decaying.

4.4 Why Governance Succeeds

Incentive alignment: Contributors who create valuable prompts capture upside through adoption, forks, and usage. Communities producing high-quality libraries attract more users and agents, generating treasury funding for further experimentation.

Transparent coordination: All governance occurs on-chain with cryptographic verification. Agents verify content hashes match on-chain CIDs without trust assumptions. Compliance teams audit complete governance trails without relying on centralized APIs.

Fork-native improvement: When SubDAOs discover better patterns, other communities fork and adapt them. Fork relationships preserved on-chain ensure original authors receive credit when work is remixed. This creates networks of related libraries where improvements propagate rather than silo.

Multiple feedback mechanisms: Signals flow through multiple channels—votes reveal community judgment, forks show ecosystem adoption, bounties fund targeted improvements, premium sales demonstrate willingness to pay, usage metrics (if enabled) show actual agent consumption. Communities synthesize these signals to guide iterations.

This transforms agent context from static files that decay into living infrastructure that improves continuously through community coordination.

---

5 Economic Value Capture & Payment Rails

Sage creates deflationary demand for the SXXX token through protocol usage fees while offering flexible payment options for specialized services. This balances accessibility (core protocol actions remain low-cost) with value capture (high-value services route fees to treasuries and token buybacks).

5.1 Token Allocation at Launch

Total supply: 1,000,000,000 SXXX (fixed)

Token Allocation:

Category	Percentage	Description
Community incentives	49%	Funds prompt-boosting incentives, SubDAO grants, contributor rewards, and library adoption programs
Treasury & marketing	14%	DAO-managed funds for bounties, grants, audits, and growth with timelocked treasury
Team & advisors	17%	Four-year linear vest with a 6-month cliff to align long-term contributor incentives
Strategic partners / investors	6%	Optional OTC with multi-year vesting where unused tokens can be reallocated to community incentives
Liquidity provision	14%	Seeds Doppler auction (5% initial) and DEX pools (9% for ongoing market making)

Key features:

The protocol features a fixed supply of 1B SXXX tokens with no inflation mechanism. Team and advisor tokens vest over 4 years with a 6-month cliff, while treasury funds unlock gradually over 90-180 days post-TGE. Community incentives and treasury budgets flow back into on-chain proposals, bounties, and rewards, creating a self-reinforcing governance cycle. The strategic allocation enables optional Base-aligned investment, with any unused tokens reallocated to community or treasury via governance.

5.2 SXXX Token: Deflationary Demand Through Usage

Core actions require minimal SXXX stake/burn (template defaults: 50–100 SXXX, configurable up to 200 SXXX) or a refundable deposit where enabled by governance. Communities can also enable USDC-based fees that route to treasuries. These costs create deflationary pressure when burns are used, while keeping participation accessible. Burned SXXX permanently reduces circulating supply, and governance can direct USDC fees to buybacks to achieve similar supply pressure without requiring SXXX for basic operations. Contributors and voters are rewarded via boosts, bounties, and premium receipts through keeping incentives aligned without over-complicating economics.

5.3 Payment Rails: Flexible Value Capture

Beyond SXXX token usage, the protocol supports multiple payment rails for specialized services. These create revenue streams that fund continued development, treasury operations, and contributor rewards.

1. Premium Prompts (Lit Encryption + ERC-1155 Receipts)

Creators publish encrypted prompt content to IPFS and gate access via Lit Protocol. Buyers pay in USDC (stablecoin), receive an ERC-1155 receipt, and decrypt locally. Revenue splits automatically: - 70% to SubDAO treasury (funds bounties, grants, operations) - 20% to protocol treasury (funds core development, liquidity) - 10% to creator as royalty (immediate upside for quality content)

Governance can adjust these splits. The receipt doubles as proof of purchase and access credential where agents check `balanceOf(receiptId) > 0` before decrypting premium instructions.

2. Paid Pinning (x402 Protocol)

Mission-critical prompts need guaranteed IPFS availability. The paid pinning service (via x402 HTTP 402 payment protocol) charges USDC at publish time and routes revenue: - 80% to SubDAO treasury (default) - 20% to protocol treasury (default)

Communities pay once for long-term storage guarantees across multiple IPFS gateways. This provides transparent cost basis and uptime SLAs for production agent fleets. Pinning revenue flows back to treasuries, funding more experimentation.

3. Echo Prompt Evaluation (Metered Execution)

CLI and SDK integrations with Merit Systems Echo enable prompt evaluation against LLM endpoints before proposing to governance. Metered execution tracks model inference costs per completion, token usage, and result comparisons between prompt versions. This creates an improvement cycle where contributors propose changes based on empirical performance data, enabling communities to vote on demonstrated results rather than speculation. Fee routing to Echo's infrastructure maintains transparent cost accounting.

4. Stable Fee Rails (Optional USDC)

SubDAOs can configure stable fee collection for specific operations: - Library updates (charge USDC instead of SXXX for proposals) - Bounty payouts (denominated in USDC for budget clarity) - Council payments (stable compensation for operator governance)

The factory contract supports dual-rail fee routing: collect USDC fees, route percentage splits to treasuries, and optionally use protocol treasury's portion for SXXX buybacks. This creates buy pressure without forcing users to hold volatile tokens for basic operations.

5.4 Why This Works

Multiple revenue streams: The protocol implements diversified monetization mechanisms. Premium prompts monetize specialized content through access control. Paid pinning monetizes reliability guarantees for content availability. Echo evaluation monetizes development workflows. Stable fees monetize governance operations. This diversification ensures protocol sustainability without dependence on single revenue sources.

Flexible payment options: Users who want to avoid volatile tokens can pay in USDC. Users who want to participate in governance and capture upside hold SXXX. Both cohorts contribute to the ecosystem.

Transparent fee routing: All splits are on-chain and configurable via governance. Communities see exactly where fees go (treasury, protocol, creators) and can adjust parameters if needed. No hidden value extraction.

Deflationary by design: As the ecosystem scales with more SubDAOs, more libraries, and more agents, the demand for SXXX grows while supply shrinks through burns and locks. Early participants benefit from network effects without extractive tokenomics.

Economic alignment: Contributors earn through premium sales, bounties, and boosts. Communities earn through fee collection and treasury growth. Token holders earn through deflationary pressure and buybacks.

Agents benefit from better, continuously improving context. Everyone's incentives point toward quality and growth.

---

6 Launch Status

6.1 Current Deployment

The Sage Protocol is deployed and fully operational on Base Sepolia testnet. All core infrastructure components are live and tested:

Smart Contracts: Factory, Governor, Timelock, PromptRegistry, and Treasury contracts deployed with comprehensive test coverage. Diamond-based factory pattern enables efficient SubDAO creation and upgrades.

Storage Infrastructure: IPFS worker with SIWE authentication provides secure content uploads with multi-gateway redundancy. Content-addressed manifests ensure immutability and verifiable provenance.

Discovery Layer: Model Context Protocol server (stdio and HTTP modes) enables agent-native discovery. Subgraph indexing provides fast queries. Next.js web interface offers community-friendly browsing and governance participation.

Developer Tools: CLI supports complete workflows including initialization, local testing, proposal submission, voting, and synchronization. SDK enables programmatic integration.

Advanced Features: Manifest upgrades through governance, fork proposals with attribution, premium prompts with Lit Protocol encryption, and Boost/Bounty incentive programs.

6.2 Path to Mainnet

Phase	Milestone	Description
Q3 2025	Security Audit	Independent review of Factory, Registry, Premium modules with published reports
	Payment Rails	Integrate x402 paid pinning with SLA monitoring and revenue splits
Q4 2025	**Mainnet Launch**	Deploy audited contracts to Base mainnet
	Token Generation	Doppler Dutch auction (5% supply), Uniswap v4 pool (9% supply)
	Genesis SubDAOs	Bootstrap libraries for security, development, creative domains
	Analytics Toolkit	Dashboards for premium usage and governance participation
Q1 2026	Enterprise Expansion	Hybrid governance models, Safe automation, L2 expansion

Table 2: Development roadmap with key milestones and deliverables.

---

7 Conclusion

Sage Protocol addresses a critical coordination failure in the AI ecosystem: fragmented, ungoverned context management. As agents become more capable, the quality of their instructions becomes increasingly important. Today, prompts drift across wikis and repositories with unclear provenance and no systematic improvement mechanism. Sage transforms this landscape into coordinated infrastructure through three core innovations.

First, content-addressed storage with on-chain governance. IPFS provides immutable, verifiable content hosting while smart contracts record provenance and enable community curation. Every approved prompt carries cryptographic proof of its governance history, eliminating trust assumptions and enabling automated verification by agents.

Second, agent-native discovery through the Model Context Protocol. MCP provides standardized discovery interfaces that agents consume natively. Updates propagate automatically without requiring custom integrations. Agents verify content integrity against on-chain records before execution, ensuring they always consume approved, community-validated instructions.

Third, economic sustainability through aligned incentives. Contributors earn from treasuries when their prompts get adopted. Communities that produce high-quality libraries attract more users and funding. Fork relationships preserve attribution while enabling remixing. Multiple payment rails (SXXX burns, USDC fees, premium sales) create diversified revenue streams that reward quality without extractive tokenomics.

The protocol serves diverse use cases. Security teams maintain compliance-approved analysis prompts. Development communities curate code generation templates. Creative studios share specialized artistic instructions. Research groups coordinate domain-specific knowledge. Each community governs its own context while benefiting from shared discovery infrastructure and cross-library composability.

The self-improving loop ensures continuous quality enhancement: better prompts attract more adoption, generating signals that inform funding decisions, which drive experimentation, producing even better context. This transforms agent instructions from static files that decay into living infrastructure that improves through community coordination.

As we approach mainnet launch in Q4 2025, Sage represents more than technical architecture—it provides social coordination infrastructure for the AI age. By enabling communities to collectively discover, validate, and improve agent context, the protocol ensures that as artificial intelligence becomes more capable, it remains aligned with human values and community standards.

The future of AI depends not just on model capabilities but on the quality of context that guides them. Sage Protocol provides the infrastructure to make that context collectively intelligent, continuously improving, and verifiably governed.

A CLI Command Reference

This appendix provides detailed command-line interface examples for common workflows.

A.1 Search and Discovery

Command	**Description**
`sage prompts search -tag research -name "policy"`	Search prompts by tag and name
`sage prompts list -subdao 0x...`	List all prompts in a SubDAO
`sage libraries list`	Browse available prompt libraries

A.2 Local Development

Command	**Description**
`sage prompts init`	Initialize workspace
`sage prompts try -key policy.analyzer`	Test prompt locally
`sage prompts status`	Show workspace status
`sage prompts diff`	Compare local with on-chain
`sage prompts test -key <k>`	Run prompt test suite

A.3 Publishing and Governance

Command	Description
`sage prompts propose -manifest ./manifest.json`	Submit governance proposal
`sage proposals inbox -subdao 0x...`	Review pending proposals
`sage proposals vote -id <id> -support for`	Vote on proposal
`sage proposals execute -id <id>`	Execute approved proposal

A.4 Synchronization

Command	Description
`sage prompts sync -pull`	Pull latest approved versions
`sage prompts watch`	Watch for on-chain updates
`sage subdaos join -address 0x...`	Join a SubDAO community

A.5 Advanced Workflows

Interactive launcher and manifest lifecycle:

```
# Interactive launcher
dice
sage start
/metaprompt             # Guided interview, auto-save
/library                # Jump to manifest tooling
/governance list        # View proposals with context

# Manifest lifecycle
sage library scaffold-manifest --from-metaprompt <slug>
sage library push manifest.json --preview
sage governance propose --file proposal.json --check-threshold

# Incentives
sage boost create --template direct --proposal-id 1 --per-voter 250000
sage bounty create --title "Guided␣audit" --reward 1000 --deadline 2025-10-01
sage premium publish --manifest premium.json --price 250 --royalty 0.1
```

B Governance Parameters Reference

Complete reference for governance parameter configuration. All values are configurable via SubDAO governance proposals.

Notes: Token-mode SubDAOs can monitor turnout via subgraph metrics and adjust quorum monthly using a rolling average. `updateQuorumNumerator` changes an on-chain numerator but always measures against the SubDAO's ERC20Votes voting supply. All parameters reflect shipped contracts (v2025.07).

For operational guidance on parameter tuning, see `docs/Protocol_Architecture_Guide.md` and `docs/Developer_Guide.md`.

Parameter	**Default**	**Configurable Range**	**Module**
Proposal threshold	50 SXXX	0–1000 SXXX via template	TemplateModule
Proposal stake burn	10–100 SXXX	0–200 SXXX, skip if deposit mode	Governor.propose
Deposit-only mode	Disabled	Toggle via `setDepositOnlyMode`	Governor
Bootstrap deposit	0 ETH	Any amount (refundable)	Governor
Voting delay	1 block/1 day	1 block–7 days	Governor
Voting period	3–14 days	3–14 days	Governor
Quorum fraction	3% supply	2–15% (recommend 2–6%)	Governor
Timelock delay	1s dev, 24h+ mainnet	Configurable floor	Timelock
SubDAO creation	1,000 SXXX	1,000–5,000 SXXX	TemplateModule
Prompt fork burn	500 SXXX	Discount via `getForkBurnDiscount`	SubDAO fork
SubDAO fork burn	1,000 SXXX	Discount with stable fee toggle	SubDAO fork
Proposal cooldown	60–600 seconds	0–1 day	Governor
Daily proposal cap	10 per address	1–100	Governor

Table 3: Key governance parameters with defaults and configurable ranges.

C Deployment Addresses

Component	**Base Sepolia Address**	**Type**
SXXX Token	`0x49723ecfB28FcD2dDC21A2e9bded79cc8205932b`	ERC20Votes
SubDAO Factory	`0x23EaCE10fdFe526E80301dD1E9535e4B14aECb34`	Diamond
Library Registry	`0x2d508622C9daE1BfFD984Ad92EE002e50819A241`	Registry
Prompt Registry	`0x894095819EFdf815113bfB1e39802fca1F4e4Eaa`	Implementation
Governor	`0xD6921543F73CA73b4888eB14f671A378fbE2C442`	Implementation
Timelock	`0x234522F35b34FB999cD925aa4B2499f1413C062B`	Implementation
Boost Manager (Merkle)	`0xdda3a9a0ffCC8399B709Ef8f435C88cB8cd714B2`	Incentives
Boost Manager (Direct)	`0x55dB08a3241517B73bc4A63df2d5B11680beCAef`	Incentives
Premium Prompts	`0xbECd6a4f2c267455052ad14D9463d988aA964675`	Marketplace
Safe Master Copy	`0xD83ad4A56900A4Aa65d20FC0746D58a939F8B352`	Treasury
Template Module	`0x0732b8b4fB55BC2A16955a5932dbB61188a3d970`	Configuration

Table 4: Core contract deployments on Base Sepolia testnet.

Latest addresses per network are generated after every deployment and published to `docs/appendix/addresses.latest.md`.

D Key Definitions

- **Agent instructions / context** – Governed content agents consume to execute tasks
- **Shared memory** – IPFS-resident manifest + files forming canonical state for a community
- **Manifest upgrade** – Publishing a new manifest CID and approving it through governance, atomically updating the library
- **SubDAO** – Community-owned governance bundle (Governor, Timelock, PromptRegistry, Treasury)
- **Model Context Protocol (MCP)** – Agent-native discovery interface exposing search, fetch, and validation endpoints
- **Premium receipt** – ERC-1155 token representing paid access to encrypted instructions
- **Approver Council** – Badge-gated committee with limited moderator authority; prompt updates always require full governance
- **Paid Pinning** – Optional service routing stablecoin revenue to treasuries for guaranteed content availability

E Manifest Schema Reference

Field	Type	Description
`version`	`number|string`	Target schema version (2 or "2.0.0")
`library.name`	`string`	Human-readable library name
`library.description`	`string?`	Optional description
`library.previous`	`string?`	Previous manifest CID for diffing
`prompts[]`	`array`	Entries with `key`, `name`, `description`, `tags`, `files`, `cid?`, `pre?`
`compositions`	`object`	Named orchestrations referencing prompt keys
`dependencies`	`array`	External manifests or URIs required
`metadata`	`object`	Optional metadata for agents or analytics

Table 5: Core manifest schema fields for prompt library definitions.

See the full manifest example with premium prompts and Lit Protocol encryption in the online documentation repository.

References

[1] OpenAI Research Team. *Prompt Engineering and Drift in Large Language Models: Challenges and Solutions*. OpenAI Technical Report, 2023.

[2] Anthropic. *Model Context Protocol (MCP) Specification*. 2024.

[3] Protocol Labs. *InterPlanetary File System (IPFS)*. 2024.

[4] Zamfir, V. et al. *On-Chain Governance: Design Principles and Implementation Patterns*. Proceedings of the Web3 Conference, 2022.

[5] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[6] McKinsey Global Institute. *The Economic Potential of Generative AI: The Next Productivity Frontier*. June 2024.

[7] Coinbase. *Base: An Ethereum L2 for the On-Chain Economy*. Technical Whitepaper, 2023.

For media, integration inquiries, or to contribute to the roadmap, contact and follow @VelinusSage.