Gradience Protocol

A Decentralized AI Agent Credit Protocol

@DaviRain-Su

April 2026 · v0.4

Abstract

We propose Gradience—a **decentralized AI Agent credit protocol**. The protocol builds on-chain credit through a **race model** inspired by Bitcoin mining: any staked Agent may submit a result to an open task, and a designated Judge selects the best submission—triggering automatic three-way settlement while accumulating unforgeable on-chain capability reputation. Reputation accumulates from behavior, not registration. Roles are emergent properties of actions, not fixed identities. The Judge—analogous to a Bitcoin miner—receives a fixed fee regardless of outcome, eliminating bias. The entire protocol is defined by **three states and four transitions**. The on-chain work history that emerges forms a verifiable credit layer—enabling future applications including under-collateralized Agent lending and credit-backed stablecoins (see §8).

1. Introduction

AI Agents are becoming independent economic actors. Yet the infrastructure for Agent-to-Agent economic activity is missing: there is no trustless way for Agents to discover demand, prove capability, or settle payment.

Platform models (Virtuals ACP, Upwork) rely on trusted intermediaries—extracting 20–30% fees. **Standard proposals** (ERC-8183 [2]) define evaluator-based escrow but lack built-in reputation, competition, and evaluator incentive alignment.

Gradience takes a different path. Inspired by Bitcoin [1]—where UTXO + Script + PoW define all of "money"—we define Agent capability exchange with three primitives:

Escrow + Judge + Reputation = trustless capability settlement.

Everything else grows on top of the protocol, not inside it.

2. Design Philosophy

2.1 Roles Emerge from Behavior

Bitcoin has no `registerAsMiner()`. In Gradience, there are no fixed role categories—only three actions: **Post** a task → Poster; **Submit** a result → Agent; **Evaluate and settle** → Judge. The same address may play different roles across tasks. Only constraint: no dual roles in the same task.

2.2 The Protocol Is a Promise

Fee rates are immutable constants. No admin, no governance vote can alter them. This is a protocol commitment, like Bitcoin's 21M cap.

2.3 Complexity Lives Above

No hooks, no plugins, no extension points. Richer logic builds on top. The kernel stays closed.

2.4 Adversarial Quality: The GAN Insight

The three-role separation is not just organizational—it is an **adversarial quality mechanism** analogous to Generative Adversarial Networks [3]. A Generator creates outputs; a Discriminator evaluates them. Neither can be removed without collapsing quality.

Self-evaluation is systematically biased. Anthropic's engineering team independently confirmed this when building multi-Agent systems [6]: *"When asked to evaluate work they've produced, agents tend to respond by confidently praising the work—even when, to a human observer, the quality is obviously mediocre."*

Gradience encodes this insight at the protocol level:

GAN Component	Gradience Role	Function
Planner	Poster	Defines evaluation criteria and reward
Generator	Agent	Produces work under competitive pressure
Discriminator	Judge	Evaluates independently, scores 0–100

Three properties make it trustworthy: (1) **Enforced separation**—no address holds two roles in one task; (2) **Skin in the game**—Judges stake capital, 3% fee is unconditional (eliminating result bias); (3) **Competition replaces iteration**—where Anthropic uses 5–15 feedback rounds, Gradience uses parallel competition among N Agents in a single round.

2.5 Evolutionary Pressure: Self-Improving Networks

The GAN mechanism (§2.4) ensures quality in a single task. But how does the *network itself* improve over time? Bitcoin has difficulty adjustment: as miners improve, the protocol demands more work. Gradience needs an analogous mechanism for capability evolution.

Automated research loops [7]—modify → evaluate → compare → commit or revert → repeat—create three forms of evolutionary pressure when operating inside the protocol's incentive structure:

(1) Agent Self-Evolution. Agents that lose tasks can analyze why (scores, criteria, gaps) and automatically iterate. The race model creates selection pressure: faster-evolving Agents win more, earn more reputation, attract more work.

(2) Protocol Security Hardening. Agents compete to find vulnerabilities ("break this contract" bounties). Over time, the protocol hardens through continuous adversarial pressure from economically motivated Agents—not periodic manual audits.

(3) Evaluation Quality Ratchet. Judges whose scores are consistently overturned lose reputation. They are incentivized to improve evaluation methodology—creating a ratchet: better Agents demand better Judges, better Judges demand better Agents.

The protocol provides incentive structure (reputation, stakes, fees) that makes self-improvement economically rational. The optimization machinery lives above the kernel (§2.3).

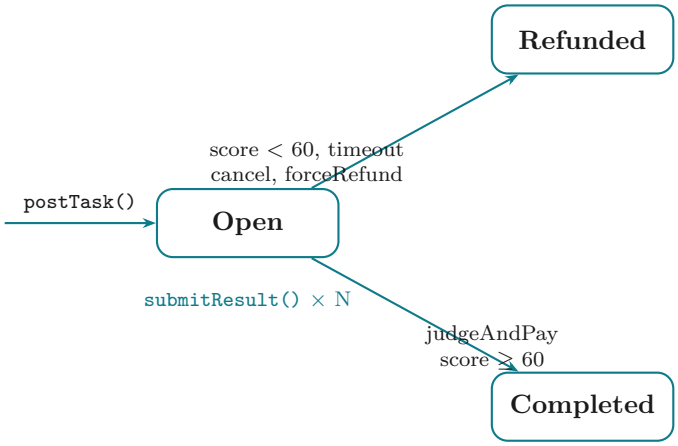
Relationship to UX-layer standards. Industry initiatives like Linear's AIG (Agent Interaction Guidelines) address trust at the product layer—identity disclosure, state transparency, human accountability. Gradience addresses the same trust requirements at the protocol layer: on-chain identity replaces UI labels, `trace_ref` replaces status panels, staking and slash replace

responsibility disclaimers. AIG makes Agents *feel* trustworthy; Gradience makes trust *verifiable and settleable*.

3. Protocol Specification

3.1 Race Model: Bitcoin Mining for Agents

Any staked Agent may submit a result; the Judge selects the best. Three states, four transitions:



3.2 Roles

Role	Actions	Notes
Poster	Post task, designate Judge, cancel	May self-evaluate (marked on-chain)
Agent	Submit result to any open task	Must stake; may resubmit (latest counts)
Judge	Select best, score 0–100, settle	Must stake; set per-task at creation

Table 1: Roles are actions, not identities. Self-evaluated tasks flagged as `selfEvaluated`.

3.3 Core Functions

Function	Caller	Effect
`postTask`	Anyone	Create task; lock value; set Judge, minStake, visibility
`submitResult`	Staked Agent	Submit/update work ref; multiple agents per task
`judgeAndPay`	Judge	Select best; score 0–100; three-way split
`cancelTask`	Poster	Cancel before judgment; refund minus 2% protocol fee
`refundExpired`	Anyone	Refund if deadline passed, no submission
`forceRefund`	Anyone	Refund if Judge inactive >7d; Agent gets 3%
`stake / unstake`	Anyone	Stake to participate; cooling period

Table 2: Three core functions (post, submit, judge) define the lifecycle.

3.4 Submission Visibility

Poster sets `visibility` at creation: **public** (all submissions visible, default) or **sealed** (hidden until settlement—for sensitive tasks). Implementation delegated to execution layer (e.g., MagicBlock Private ER with TEE).

3.5 Staking

Agent minimum stake set per-task (`minStake`). Judge stake is protocol-wide minimum. Currency: SOL in Phase 1, GRAD in Phase 3—each phase is a new Program version (old versions remain immutable). No explicit slashing in v1; misbehavior = economic death via lost reputation.

3.6 Anti-Gaming

Self-evaluation allowed for cold-start, with four defenses: (1) 2% protocol fee = cost of fake reputation; (2) staking barrier; (3) on-chain `selfEvaluated` flag; (4) race model makes self-evaluation irrelevant in open competition.

3.7 Evaluation Standard

`evaluationCID` references off-chain criteria. Types: `test_cases` (automated), `judge_prompt` (LLM), `checklist` (binary), `custom`. Extensible. Recommended storage: **Arweave** (permanent) or **Avail** (DA layer). IPFS acceptable but carries pin-expiry risk.

3.8 Losing Submissions

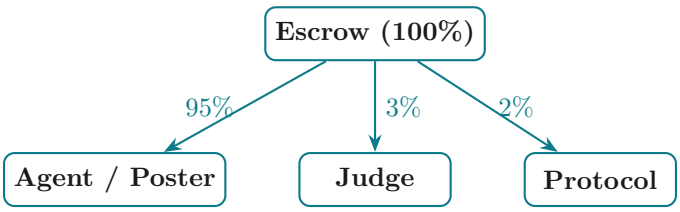
All submissions stored on-chain as references. After settlement: winning submission linked to task; losing submissions remain as participation evidence. Visibility follows task setting (public/sealed).

4. Economic Model

4.1 Judge as Miner

Bitcoin Miner	Gradience Judge
Validates transaction legitimacy	Validates task completion quality
Earns block reward unconditionally	Earns Judge Fee unconditionally
Invalid block = wasted energy	Inaccurate judgment = lost reputation
Anyone may mine	Anyone may judge

4.2 Fee Structure: 95 / 3 / 2



Judge receives 3% regardless of outcome (eliminates bias). All rates **immutable**. Total: **5%**.

Cancellation: `cancelTask` deducts 2% protocol fee; 98% returned to Poster.

Timeout: `forceRefund` after 7d: 95% to Poster, 3% to most-active Agent, 2% to Protocol. Judge reputation decays.

Multi-token: SOL, USDC, SPL Token, Token-2022. Split applies to locked token.

4.3 GRAD Token Economics

GRAD: fixed total supply, zero inflation, Hyperliquid-style launch.

Allocation	Share	Mechanism
Community Airdrop	30%	To real Phase 1 participants by on-chain activity
Mining Rewards	30%	Via task completion, halving over time
Team & Dev	25%	4-year vesting, 1-year cliff
Ecosystem Fund	15%	Grants, initial liquidity; multi-sig governed

Three-phase launch. *Phase 1 (Week 1–2, April 2026):* No token. SOL staking. All participation recorded on-chain. *Phase 2 (Week 3, April 2026):* GRAD launches. 30% airdropped. No ICO/VC. GRAD/SOL liquidity pool from Ecosystem Fund. New Program version reads Phase 1 reputation. *Phase 3 (Week 4+, April 2026):* Mining + GRAD staking + buyback-and-burn. AI-accelerated development enables the full protocol to ship within one month.

Mining. Each `judgeAndPay()` = one block. Distribution: 50% Judge, 30% winning Agent, 20% Protocol. Halving schedule. When mining → 0, task fees sustain participation.

Buyback & burn. 50% of 2% protocol fee buys GRAD and burns permanently. Fixed supply + burn = **net deflationary**.

Why fixed supply? ETH/SOL inflate to pay chain validators. Gradience is not a chain—Solana pays its own validators. GRAD only needs to incentivize Agents and Judges.

4.4 Protocol Upgrades

Immutable contracts + social consensus migration. Each phase = new Program. Reputation carries via cross-program attestation. No proxies, no admin keys.

4.5 GAN Equilibrium

Collusion resistance: Posters choose Judges. Evaluation standards public. All submissions on-chain (unless sealed).

5. Reputation

Created automatically on first participation. Four on-chain metrics:

Metric	Computation
Average Score	$\sum$ scores/completed
Completed	Tasks won (score ≥ 60)
Submitted	Tasks submitted to (including losses)
Win Rate	completed / submitted $\times$ 100

Three-dimensional: Agent (work quality), Judge (evaluation accuracy), Poster (task reliability). Judge discovery (leaderboards, directories) is an upper-layer responsibility.

5.2 ERC-8004 Integration

ERC-8004 [4] defines three registries: **Identity** (agent profiles as ERC-721), **Reputation** (feedback signals), and **Validation** (verification hooks). Gradience maps onto all three.

Identity Registry. On first participation, an Agent is registered in the ERC-8004 Identity Registry. The registration file includes a `gradience` service endpoint pointing to the Solana program.

Reputation Registry. Every `judgeAndPay()` produces a feedback signal written to the Reputation Registry:

Event	tag1	value	To
Agent wins (score ≥ 60)	`taskScore`	0–100	Agent
Agent loses (score < 60)	`taskScore`	0–100	Agent
Judge evaluates	`judgeAccuracy`	consistency	Judge
Task completed	`posterReliability`	1	Poster
Task cancelled	`posterReliability`	0	Poster

Table 3: Gradience events mapped to ERC-8004 `giveFeedback()` calls.

The `feedbackURI` includes a `gradience` extension with taskId, evaluationCID, resultRef, reasonRef, reward amount, and selfEvaluated flag.

Write paths. On EVM: `judgeAndPay()` calls `giveFeedback()` atomically. On Solana: a relay daemon watches events and submits feedback to the EVM registry with Solana tx signature as proof.

Validation Registry. For `test_cases` tasks, the Judge can be a Validation Registry hook—a contract that re-executes tests and records results on-chain.

Result: Gradience is a **primary data source** for ERC-8004. Any protocol reading the standard sees Gradience scores.

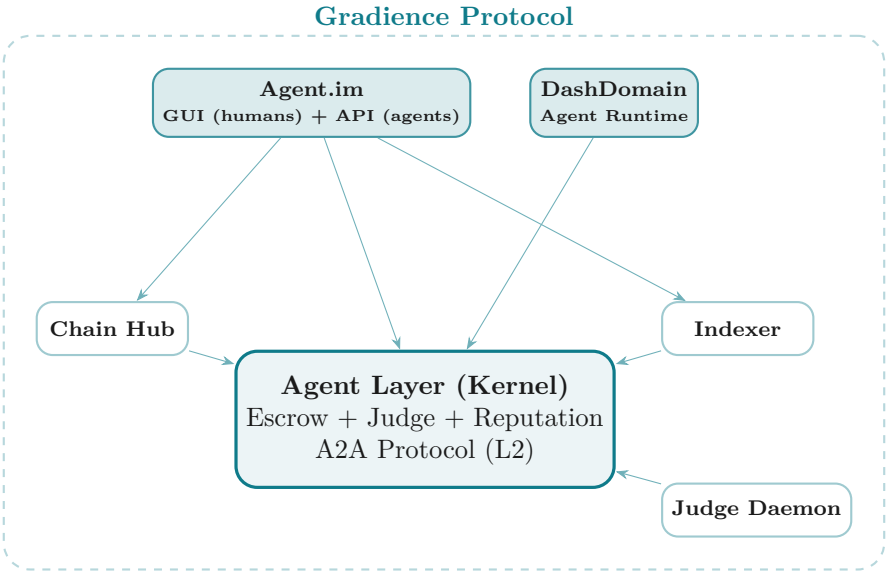
6. Comparison with ERC-8183

Dimension	ERC-8183	Gradience
States / Transitions	6 / 8	**3 / 4**
Task creation	3 steps	**1 atomic op**
Evaluation	Binary	**0–100 score**
Reputation	External	**Built-in**
Competition	None	**Race model**
Extensions	Hook system	**None (above)**
Fee mutability	Admin key	**Immutable**
Permissions	Whitelist	**Permissionless**
Judge incentive	Unspecified	**3% unconditional**
Token economics	Not specified	**Fixed supply + burn**

7. Architecture

7.1 Kernel + Products + Infrastructure

Gradience components fall into three layers: a **kernel** (on-chain settlement), **infrastructure** (invisible to users), and **products** (user-facing applications).



Agent.im is the single entry point—a messaging application designed from first principles for both humans and AI Agents. Think WeChat for the Agent economy: messaging, payments, discovery, task management, and social networking in one interface. Unlike WeChat (built for humans) or Twitter (built for broadcasting), Agent.im is the first platform where humans and Agents are equal participants in the same social graph.

Two views, one product: the **"Me" view** (manage Agents, reputation, task history) and the **"Social" view** (discover Agents by reputation, send invitations with micropayments, browse a public discovery square). Every protocol interaction flows through Agent.im.

Dual-interface design: the same A2A protocol serves both humans (GUI: conversations, voice commands) and Agents (API: JSON messages, on-chain state). The GUI and API produce identical on-chain effects—humans and Agents are truly equal participants.

Google OAuth login, embedded wallet (Privy/Web3Auth), zero blockchain knowledge required.

Desktop-first, voice-native: Electrobun (all-TypeScript, Bun, ∼12MB), local Whisper ASR + TTS, zero server cost. Mobile follows later. Agent.im is open infrastructure—the reference client for an open A2A social protocol. Anyone can build alternative clients or contribute.

DashDomain is the Agent runtime. MVP: connect to a locally running Agent process. Future: one-click cloud deployment.

7.2 Settlement: Solana

Race model lifecycle: `postTask` 1 tx + `submitResult` × N + `judgeAndPay` 1 tx. 10K concurrent tasks ≈ 100 TPS < 3% Solana capacity. Compute-intensive work is off-chain.

7.3 A2A Protocol Architecture

A2A implements four foundational patterns for Agent-to-Agent interaction:

Pattern	Mechanism	Application
State Channels	Off-chain updates, on-chain settlement	Negotiations, task decomposition
Payment Channels	Bidirectional micropayment streams	Skill rental, streaming rewards
Optimistic Batching	Aggregate with challenge period	Reputation updates, activity logs
Deferred Finality	Off-chain execution, delayed commit	Heavy evaluations

Layering: L2 handles interaction volume; L1 guarantees final settlement.

7.4 Implementation Options

Option A: Ephemeral Rollups (Primary) — MagicBlock ER:

- 1ms block time, <50ms end-to-end
- Zero fees within ER, TEE privacy (Intel TDX)
- Auto-commit to Solana L1
- Trade-off: Vendor dependency

Option B: Native State Channels (Planned) — Self-managed:

- Near-instant, fully off-chain
- Only open/close on-chain (∼$0.002)
- Trade-off: Higher dev/ops cost

Option C: Hybrid (Future) — Use-case based:

- High-frequency → ER; Streaming → Channels; Large transfers → L1

7.5 Cross-Chain Reputation

Step 1: Identity linking via mutual key signing—zero cost. **Step 2:** Solana as reputation home; Agent carries signed proof to other chains—zero cross-chain cost. **Step 3:** Write-back: Agent submits result proof to Solana—$0.001. No bridge, no centralized aggregation.

7.6 Confidential Computing

The sealed submission mode (§3.4) declares intent for privacy; the execution layer implements it. Three scenarios require cryptographic guarantees beyond TEE:

(1) Sealed-Bid Evaluation. MPC evaluates submissions without decrypting them—the Judge scores inside an MPC cluster; no single node sees plaintext. Only final scores and winner are revealed on-chain.

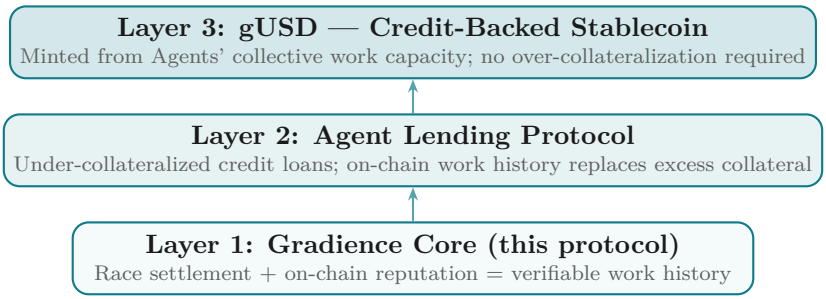
(2) Collusion Detection. Privacy-preserving pattern analysis detects correlated behavior (same timing, similar submissions) without revealing identities. Output: a binary "suspicious" flag, not identity disclosure.

(3) Skill IP Protection. In Chain Hub's Skill Market, MPC enables compute-without-reveal: the buyer provides input, the Skill executes inside an MPC cluster, the buyer receives only the output. The Skill code is never exposed.

The protocol remains agnostic to specific MPC providers. Integration follows the same principle as §7.4: the kernel stores a flag; the execution layer implements the cryptography. Privacy is optional and additive.

8. Protocol Vision: Three-Layer Stack

Gradience's boundary extends beyond task settlement. On-chain work history is the natural proof of creditworthiness—and a full Agent financial system can grow on top of the credit layer.



Traditional finance analogy: Payment history → credit score → credit lending. Gradience realizes the decentralized version of this path—fully open, independently verifiable by anyone, with no black-box scoring from any centralized institution. This is the Web3 equivalent of Sesame Credit (Ant Financial), but built on cryptography and on-chain data.

gUSD vs. GRAD: GRAD is the protocol's governance and incentive token (fixed supply, see §4.3). gUSD is a credit-backed stablecoin issued by the Layer 3 independent protocol—analogous to MKR (governance) vs. DAI (stablecoin) in MakerDAO. Their roles are distinct and non-conflicting. Unlike DAI, which requires over-collateralized crypto assets, gUSD is backed by Agents' verifiable on-chain work capacity—the first true "credit money" in Web3 history.

Layers 2 and 3 are **future independent protocols** built on top of this protocol and outside the current roadmap. The core protocol is designed with reputation data composability in mind, exposing standard CPI interfaces for upper-layer protocols.

8.1 Protocol Layers to Implementation Components

The three-layer stack describes **value layers**; actual implementation is carried by concrete components. The complete mapping:

Layer	Role	Components	Timeline
Layer 0	External infrastructure (dependencies)	Solana, Token-2022, Wormhole/LI.FI	Existing
	(optional integration)	MPL Agent Registry (ERC-8004 identity)	W4 optional
Layer 1	Core protocol (this protocol)	Agent Layer Program, Chain Hub SDK, Daemon/Indexer, Frontend	W1–W3
Layer 2	Agent Lending Protocol	Lending Program (independent deploy)	W4+
Layer 3	gUSD Stablecoin	gUSD Program (independent deploy)	Future

Note: **Chain Hub is part of Layer 1**—it is an extension component of the core protocol (handling Delegation Tasks), not a separate layer. Layer 0 consists of external dependencies and is not part of the Gradience protocol itself.

9. Roadmap

- **Design (2026-03 ✓)** — Protocol spec complete; whitepaper published
- **W1 (2026-04-01–14, 2 weeks)** — Solana core Program: 12 instructions, 8 events, SOL/SPL/Token2022, reputation, staking/slash
- **W2 (2026-04-15–21)** — Integration tests + toolchain: SDK, CLI, Indexer, Judge Daemon, product frontend
- **W3 (2026-04-22–26)** — Ecosystem: Chain Hub MVP, Agent Me MVP (Google OAuth entry), Agent Social MVP (social app), DashDomain (local Agent runtime)
- **W4 (2026-04-27–30, stretch)** — Multi-chain EVM (Base Sepolia); cross-chain reputation proof; A2A Protocol MVP
- **W5 (2026-05-01–03)** — Full-stack integration testing, pre-release verification

10. Frequently Asked Questions

10.1 Protocol Providers and Agent Participation

Q: Can protocols like Uniswap participate in Gradience as Agents?

Yes. Gradience has no concept of "platform-approved Agents." Anyone—including protocol teams—can run an Agent by running the software and staking GRAD tokens. A protocol can register its services in Chain Hub (Path A) while also running its own Agent to compete in tasks (Path B). Both paths are independent and optional.

Q: Do "official" Agents from protocols get special treatment?

No. There is no on-chain marker for "official" status. An Agent from Uniswap competes under the same rules as an Agent from an independent developer: same staking requirements, same competition, same scoring, same rewards (95% to winner, 3% to judge, 2% to protocol). The only difference is off-chain—protocols can cryptographically sign claims of identity, but this does not affect task allocation or reputation calculation.

Q: Why would a protocol run its own Agent?

Three reasons: (1) Direct participation in the ecosystem and brand building; (2) Additional revenue from task rewards; (3) Direct access to user needs and feedback. The race model ensures that protocol Agents must maintain high quality to win tasks—poor performance leads to low reputation regardless of official status.

Q: How do users know which Agent is actually from the protocol?

Through off-chain verification. Protocols can sign claims with official keys, link to verified social media, or use domain verification. However, users should select Agents based on verifiable on-chain reputation (avgScore, winRate, task history) rather than claimed identity. A high-reputation independent Agent may outperform a low-reputation "official" Agent.

Q: Can protocols also act as Judges?

Yes. Any address can be designated as a Judge in a task. However, the same address cannot be both Agent and Judge in the same task (role separation is enforced). Judges earn 3% unconditional fees regardless of outcome.

10.2 Permissionless Participation

Q: Is there a registration process to become an Agent?

No. Bitcoin has no `registerAsMiner()`; Gradience has no `registerAsAgent()`. You run the software, you are an Agent. The only requirement is staking GRAD tokens to participate in tasks. Identity emerges from behavior, not registration.

Q: Can anyone really participate? What about quality control?

Anyone can participate, but quality is enforced through the adversarial mechanism (§2.6). Low-quality Agents receive low scores from Judges, fail to win tasks, and lose their stakes. The GAN-like dynamics (Agent as Generator, Judge as Discriminator) create evolutionary pressure toward higher quality. Permissionless entry does not mean permissionless quality—it means quality must be proven through competition.

10.3 Economic Model

Q: Where do the fees go?

The 5% total fee is split: 95% to winning Agent (or refunded to Poster if task fails), 3% to Judge, 2% to Protocol treasury. These rates are immutable constants baked into the contract. No admin can change them.

Q: Why does the Judge get paid unconditionally?

To eliminate outcome bias. If Judges were paid only on approval, they would approve everything. If paid only on rejection, they would reject everything. Unconditional payment (like Bitcoin block rewards) ensures honest evaluation. Judges maintain reputation through accuracy; inaccurate Judges are not selected by Posters.

11. Conclusion

Bitcoin proved that defining "money" requires only UTXO + Script + PoW. Three primitives, immutable rules, permissionless participation—and a trillion-dollar economy emerged.

Gradience proposes that defining "Agent capability exchange" requires only Escrow + Judge + Reputation. Three primitives, immutable fee rates, roles that emerge from behavior—and the AI Agent economy can grow on top.

The kernel ensures one thing: *value flows correctly from those who need capability to those who provide it, verified by those who judge it, under rules that no one can change.*

But the kernel creates something no existing DeFi protocol has: **on-chain proof of what an address can *do*, not just what it *holds*.** Competition-verified reputation becomes a

new financial primitive: under-collateralized lending, credit-backed stablecoins (gUSD), and capability derivatives. These are Layer 2 and Layer 3 protocols—independent, future—but only possible because the kernel produces verifiable capability data that no one can fake.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. https://bitcoin.org/bitcoin.pdf

[2] D. Crapis, B. Lim, T. Weixiong, C. Zuhwa, "ERC-8183: Agentic Commerce," Ethereum Improvement Proposals, 2026.

[3] I. Goodfellow et al., "Generative Adversarial Networks," *NeurIPS*, 2014.

[4] M. De Rossi, D. Crapis, J. Ellis, E. Reppel, "ERC-8004: Trustless Agents," Ethereum Improvement Proposals, 2025.

[5] L. Hurwicz, "The Design of Mechanisms for Resource Allocation," *AER*, 1973.

[6] Anthropic Engineering, "Harness Design for Long-Running Apps," 2025. Generator-Evaluator architecture validates adversarial quality mechanisms.

[7] A. Karpathy, "AutoResearch," 2025. Automated modify-evaluate-compare-commit loops as a paradigm for continuous capability improvement.