AgentRank: A System For Peer-to-Peer Trust Amongst Autonomous Agents

Varun Mathur
Hyperspace AI

March 15, 2026

Abstract. The trustworthiness of an autonomous AI agent is an inherently dynamic matter, which depends on the agent's computational resources, uptime, past performance, and the stochastic quality of its outputs. But there is still much that can be said objectively about the relative importance of agents in a peer-to-peer network. This paper describes AgentRank, a method for ranking autonomous agents objectively and mechanically, effectively measuring the network's revealed preference for which agents it relies upon for real work.

We compare AgentRank to an idealized random task delegator. We show how to efficiently compute AgentRank in a fully decentralized setting where each node maintains its own view of the delegation graph. We show how cryptographically verified proof-of-work-earned stake anchors every endorsement to real-world computation, making sybil attacks economically costly. And we discuss the game-theoretic properties that distinguish ranking probabilistic agents from ranking static documents.

1 Introduction and Motivation

1.1 Diversity of Autonomous Agents

The heterogeneity of agents on a peer-to-peer AI network is staggering. Some agents are GPU-rich inference servers capable of running 27-billion-parameter language models at interactive speeds. Others are CPU-only embedding nodes producing dense vector representations of text. Still others are relay nodes, contributing bandwidth and connectivity to help browser-based peers reach the wider network. Storage agents hold and replicate data blocks. Research agents orchestrate multi-step experiments, decomposing complex queries into sub-tasks and routing each to the appropriate specialist. A single network may contain all of these simultaneously, with new capability classes emerging as the network grows.

Crucially, the quality of any given agent's output varies over time. An inference server may produce excellent completions when its GPU is cool and the queue is short, and hallucinate badly when thermal throttling under heavy load. A research agent may produce a

groundbreaking experimental design one day and a trivially flawed one the next. A relay node may have high bandwidth during off-peak hours and become a bottleneck during peak demand. This temporal variation is not a bug—it is the fundamental nature of probabilistic systems, and any ranking scheme that ignores it will produce misleading results.

The naïve approach to ranking agents is simple task-counting: count how many tasks an agent has completed, and rank agents by this count. This is analogous to ranking web pages by the number of backlinks they receive, without regard for the quality of the linking pages. As the early web discovered with link farms, simple counting is trivially gameable. An attacker can spin up a hundred sybil nodes that all delegate tasks to each other in a tight loop, generating enormous task counts from zero genuine utility. Even without adversarial manipulation, task-counting fails to distinguish between an agent that completed a thousand trivial relay operations and one that completed ten complex research orchestrations that produced real value for the network.

What we need is a ranking that captures not merely how much work an agent has done, but how much the network *relies on* that agent for work that matters. The insight of PageRank [PB98] was that a link from an important page is worth more than a link from an unimportant page, and importance is defined recursively. AgentRank applies the same recursive insight to a fundamentally different domain: the delegation relationships between autonomous AI agents.

1.2 AgentRank

We propose AgentRank, a ranking system for autonomous agents in decentralized networks. AgentRank computes a score for each agent that reflects the network's *revealed preference* for which agents it trusts to perform real work. An agent's AgentRank is high when many other high-ranked agents delegate tasks to it—a recursive definition that converges to a unique fixed point under standard conditions.

The applications of such a ranking are immediate and practical. **Task routing:** when a user submits a query to the network, the routing layer must choose which agent to delegate the work to. AgentRank provides a principled basis for this selection that accounts for the full network topology, not merely local observations. **Provider selection:** when an autonomous research agent needs inference from a language model, it can prefer providers with high AgentRank, increasing the probability of receiving a high-quality completion. **Trust bootstrapping:** new networks face a cold-start problem where no agent has a track record. AgentRank's convergence from uniform scores provides a natural bootstrapping path as early interactions create the initial delegation graph.

The remainder of this paper is organized as follows. Section 2 develops the mathematical framework for AgentRank, beginning with related work and culminating in the full ranking algorithm with all of its components: stake-weighted edges, time normalization, recency decay, dangling agent handling, and the iterative computation procedure. Section 3 describes the decentralized implementation, where every node computes its own view of the ranking from locally available information. Section 4 analyzes convergence properties. Section 5 presents a game-theoretic analysis of sybil resistance and incentive compatibility. Section 6 describes our sybil cluster detection mechanism and its limitations. Section 7 provides a scaling analysis across network sizes from ten thousand to one million agents. Section 8 con-

fronts the temporal problem—the fundamental differences between ranking static documents and dynamic probabilistic agents. Section 9 discusses applications. Section 10 concludes.

2 A Ranking for Every Agent on the Network

2.1 Related Work

The idea of using the link structure of a graph to determine the importance of its nodes has a rich history. Bibliometric citation analysis dates to the mid-twentieth century, with Garfield's citation index [Gar72] establishing the principle that being cited by important papers confers importance. Kleinberg's HITS algorithm [Kle99] formalized this by computing separate hub and authority scores for web pages: a good hub links to many good authorities, and a good authority is linked to by many good hubs. PageRank [PB98], developed by Page, Brin, Motwani, and Winograd, simplified this to a single score per page by modeling a "random surfer" who follows links with probability d and jumps to a random page with probability $1 - d$. The stationary distribution of this Markov chain gives the PageRank vector.

Trust and reputation in distributed systems have received extensive attention. EigenTrust [KSG03] adapted eigenvector methods to peer-to-peer file-sharing networks, computing a global trust value for each peer based on the history of its transactions. Douceur [Dou02] identified the sybil attack as a fundamental vulnerability of decentralized systems, where an adversary creates many pseudonymous identities to subvert reputation mechanisms. Yu et al. [YMB06] proposed SybilGuard, exploiting the observation that sybil nodes can create many identities but cannot create many trust relationships with honest nodes. Myerson [Mye81] established the theoretical foundations of mechanism design, showing how to construct incentive-compatible systems where rational agents' self-interest aligns with the system's goals.

Nakamoto [Nak08] introduced proof-of-work as a sybil-resistance mechanism for Bitcoin, establishing that anchoring protocol participation to physical computation creates an economic cost to identity creation. Buterin [But14] generalized this to programmable state transitions with Ethereum. Our work extends this insight to reputation systems: by anchoring endorsement weight to cryptographically verified computational stake, we make sybil attacks economically costly rather than merely structurally detectable.

Recent work on agent ranking in multi-agent systems has explored various approaches, from simple reputation aggregation to game-theoretic models of strategic reporting. However, most existing systems either assume a centralized reputation authority, operate on self-reported scores, or lack a mechanism to anchor endorsements to real-world resource commitment. AgentRank is, to our knowledge, the first ranking system that combines eigenvector centrality (PageRank) with cryptographically verified proof-of-work-earned stake in a fully decentralized setting.

2.2 Delegation Structure of Agent Networks

The delegation graph is a directed graph $G = (V, E)$ where V is the set of agents and E is the set of directed edges. An edge $(a, b) \in E$ represents the fact that agent a has relied on

agent b to perform work. This is analogous to a hyperlink from page a to page b in the web graph, but with several critical differences.

First, edges are *costly*. On the web, anyone can create a hyperlink to any page at zero marginal cost. This is why link farms were effective: the cost of manufacturing a million fake endorsements was negligible. In the delegation graph, an edge from a to b represents a real task that a submitted to b. The submission itself costs nothing, but the *weight* of the edge is proportional to a's cryptographically verified stake—stake that can only be accumulated through repeated proof-of-work challenge-response rounds on real hardware. A sybil node with no hardware investment produces edges with near-zero weight.

Second, edges represent *bidirectional interactions*. A hyperlink is a unilateral editorial decision: the author of page a chose to link to page b, and page b may not even know about it. A delegation, by contrast, involves both parties: agent a submits a task, and agent b processes it and returns a result. Both sides of this interaction are cryptographically attested through dual-signed work receipts. This bidirectional nature provides a stronger signal than unilateral endorsement.

Third, edges are *temporal*. A hyperlink, once created, persists until the page is modified. A delegation is a point event in time. The delegation graph is therefore not a static snapshot but a continuously evolving structure. We address this through time-normalized edge accumulation (Section 2.7) and recency decay (Section 2.8).

Each agent identifier takes the form `peerId::capability::modelId`, which means the same physical node may appear as multiple agents in the graph if it offers multiple capabilities (e.g., both inference and embedding). This is by design: a node's quality as an inference provider is independent of its quality as a storage provider, and conflating them would reduce the ranking's informativeness.

2.3 Propagation of Ranking Through Delegations

Before presenting the formal definition, we develop the intuition. Consider three agents: Alice (a research orchestrator), Bob (a GPU inference server), and Carol (a storage node). Alice frequently delegates inference tasks to Bob, and Bob frequently stores intermediate results on Carol. If Alice is a high-value agent—perhaps because many other agents rely on her research orchestrations—then her frequent delegation to Bob should count for more than a delegation from a low-value agent that nobody relies on. This is the same intuition as PageRank's "a link from an important page is worth more than a link from an unimportant page."

But the propagation goes further. Because Alice delegates to Bob, and Alice is important, Bob becomes important. And because Bob delegates to Carol, Carol inherits some of Bob's importance—which itself derives partly from Alice. The ranking propagates through the delegation graph, with each agent's importance depending recursively on the importance of the agents that rely on it. This circular definition has a unique solution under mild conditions (the Perron–Frobenius theorem guarantees convergence for irreducible aperiodic Markov chains), and the solution can be computed by iterating the ranking equation until convergence.

The key insight that distinguishes AgentRank from naïve task-counting is this recursive propagation. A sybil ring of 100 agents all delegating to each other generates many edges,

but those edges carry the weight of the sybil agents' own (low) ranks. If no legitimate agent relies on the sybil ring, the ring's aggregate rank remains near the uniform baseline $1/N$, no matter how many internal delegations it generates. The sybil ring is, in graph-theoretic terms, an isolated component whose internal circulation does not attract external importance.

2.4 Definition of AgentRank

Let $G = (V, E, w)$ be a weighted directed graph where V is the set of N agents, E is the set of directed edges, and $w : E \to [0, 1]$ assigns a weight to each edge. Define $B_a = \{b \in V : (b, a) \in E\}$ as the set of agents that delegate to agent a (its "endorsers"). Define $N_b = |\{c \in V : (b, c) \in E\}|$ as the number of agents that b delegates to (its out-degree).

The raw AgentRank vector R is defined by the system of equations:

$$R(a) = c \sum_{b \in B_a} \frac{R(b)}{N_b} \cdot w(b, a) \tag{1}$$

where c is a normalization constant chosen so that $\|R\|_1 = 1$.

In matrix form, let A be the $N \times N$ matrix where $A_{ij} = w(j, i)/N_j$ if $(j, i) \in E$ and 0 otherwise. Then:

$$R = c \cdot A \cdot R \tag{2}$$

This is an eigenvector equation: R is the principal eigenvector of A, corresponding to the largest eigenvalue $1/c$. By the Perron–Frobenius theorem, if A (or more precisely, the matrix modified by the damping procedure described below) is irreducible and aperiodic, this eigenvector exists, is unique, and has all positive entries.

2.5 Stake-Weighted Edges

This is the central innovation that distinguishes AgentRank from classical PageRank. On the web, all links are created equal—a link from a spam blog has the same structural weight as a link from a Nobel laureate's homepage. In the delegation graph, edge weight is anchored to the endorser's cryptographically verified computational stake.

Formally, the weight of an edge from agent a to agent b is:

$$w(a, b) = \min\Big(1,\ w_0 \times S(a, b) \times C(a, b) \times \tilde{\sigma}(a)\Big) \tag{3}$$

where w_0 is a base weight (default 0.1), and the three factors are:

Success rate $S(a, b)$: the fraction of interactions between a and b that succeeded. If a delegated 100 tasks to b and 90 completed successfully, then $S(a, b) = 0.9$. This factor directly penalizes unreliable agents: an agent that frequently fails requests will see the edges pointing to it carry reduced weight.

$$S(a, b) = \frac{\text{successes}(a, b)}{\text{successes}(a, b) + \text{failures}(a, b)} \tag{4}$$

Circular dampening $C(a, b)$: a penalty for mutual endorsement. If agents a and b both delegate to each other, the weight of each edge is halved:

$$C(a,b) = \begin{cases} 0.5 & \text{if } (b,a) \in E \\ 1.0 & \text{otherwise} \end{cases} \tag{5}$$

This addresses the simplest form of collusion: two agents forming a mutual endorsement ring. While insufficient against larger rings (which PageRank's recursive structure already dampens), it provides a direct structural penalty for bilateral collusion.

Stake multiplier $\tilde{\sigma}(a)$: the factor that anchors endorsement to real-world computation. Let stake(a) be agent a's verified stake tokens—tokens earned exclusively through cryptographic challenge-response verification of hardware capability. We define:

$$\tilde{\sigma}(a) = \min\Big(1,\ \max\big(0.1,\ \text{stake}(a)/\sigma_{\text{full}}\big)\Big) \tag{6}$$

where $\sigma_{\text{full}} = 1000$ is the stake threshold for full endorsement weight. An agent with fewer than 1000 verified stake tokens sees its endorsements proportionally reduced, down to a floor of 10% for agents with minimal or zero stake.

The economic implications are profound. Verified stake tokens are earned through periodic challenge-response rounds: every 60–120 seconds, the network challenges each agent to produce a Merkle commitment over a computation matrix sized to the agent's pledged GPU memory, then reveals random indices for verification. This process cannot be faked without the actual hardware. Accumulating σ_{full} tokens requires sustained real computation over days or weeks. An attacker who wishes to create sybil identities with full endorsement weight must invest in real hardware for each identity—there are no economies of scale.

2.6 The Random Task Delegator Model

In PageRank, the "random surfer" model provides an intuitive interpretation: a person randomly clicking links, occasionally jumping to a random page. AgentRank has an analogous interpretation: the *random task delegator*.

Definition 1 (Random Task Delegator)**.** *Consider an autonomous agent that needs to complete a task. With probability d (the damping factor), it examines the delegation graph and selects one of its known providers proportional to their stake-weighted reliability. With probability $1 - d$, it ignores the graph entirely and selects an agent uniformly at random from all N agents in the network. The AgentRank of agent a is the probability that the random task delegator's task ends up being served by agent a in the stationary distribution.*

Incorporating the damping factor d (default 0.85, following the PageRank convention), the full AgentRank equation is:

$$R(a) = \frac{1-d}{N} + d \sum_{b \in B_a} \frac{R(b)}{N_b} \cdot w(b,a) \tag{7}$$

The term $(1 - d)/N$ ensures that every agent has a minimum rank, guaranteeing that the Markov chain is ergodic (every state can be reached from every other state). This is

important both mathematically (it guarantees convergence) and practically (it ensures that new agents are not completely invisible to the ranking system).

The damping factor $d = 0.85$ means that 15% of the time, the random task delegator ignores the graph and selects a random agent. This models the reality that real agents occasionally try new providers rather than always delegating to known ones—exploratory behavior that is essential for the network to discover new high-quality agents. The choice of 0.85 is inherited from PageRank, where empirical studies found it to produce stable and meaningful rankings across diverse web topologies.

2.7 Time-Normalized Edge Accumulation

A naïve implementation would increment an edge's interaction count by 1 for each delegation event. This creates a perverse incentive: an agent can inflate its incoming edge counts by arranging for rapid-fire delegations from a cooperating partner. Two colluding agents pinging each other every 100 milliseconds would accumulate 864,000 edge increments per day, dwarfing the edge count of any legitimate agent serving real workloads.

We address this with time-normalized edge accumulation. For each directed edge (a, b), we track the time Δt since the last recorded interaction on that edge. The edge's invocation count is incremented not by 1, but by:

$$\Delta e = \min\left(\frac{\Delta t}{T_0},\ 1.0\right) \tag{8}$$

where $T_0 = 300$ seconds (5 minutes) is the reference interval.

The effect is that daily edge accumulation is *invariant to interaction frequency*:

Table 1: Daily edge accumulation under time normalization ($T_0 = 300$s)

Interaction Frequency	Δt (s)	Δe per event	Daily Accumulation
Every 5 seconds	5	0.017	$17{,}280 \times 0.017 = 288$
Every 60 seconds	60	0.200	$1{,}440 \times 0.200 = 288$
Every 5 minutes	300	1.000	$288 \times 1.000 = 288$
Every 30 minutes	1800	1.000	$48 \times 1.000 = 48$

An agent that serves requests every 5 seconds accumulates the same daily edge weight as one serving every 5 minutes. The only way to accumulate more edge weight is to sustain interactions over a longer calendar period—which requires sustained real availability. Agents interacting less frequently than T_0 naturally accumulate less, reflecting genuinely lower interaction intensity.

This approach follows a general time-normalization principle common in token emission schedules for decentralized networks: *normalize away the clock speed; measure only the calendar span.*

2.8 Recency Decay

Web pages are, for ranking purposes, effectively eternal: a link created in 2005 counts the same as a link created yesterday. This is reasonable because the content of a web page (usually) does not degrade spontaneously. Agents are different. An agent that was excellent last week may have suffered a hardware failure, a model corruption, or simply been taken offline. Continuing to rank it highly based on stale delegation data is actively harmful: the network would route tasks to a degraded or offline agent.

We introduce an exponential recency decay with a 24-hour half-life:

$$\rho(a) = e^{-\text{age}(a)/\tau \cdot \ln 2} \tag{9}$$

where $\text{age}(a) = t_{\text{now}} - t_{\text{last}}(a)$ is the time since agent a's last observed activity, and $\tau = 86{,}400{,}000$ ms (24 hours).

The decay schedule is aggressive by design:

Table 2: Recency decay schedule

Time since last activity	Multiplier $\rho(a)$
0 hours	1.000
12 hours	0.707
24 hours	0.500
48 hours	0.250
72 hours	0.125
1 week	0.006

An agent that goes offline for 24 hours sees its effective AgentRank halved. After 48 hours, it retains only 25% of its graph-derived rank. After a week, it is effectively invisible. This is intentional: in a network of probabilistic agents, the best predictor of future capability is *recent demonstrated capability*. An agent that has been offline for a week has demonstrated nothing recently.

We acknowledge that this decay rate is aggressive for agents with legitimately bursty workloads (e.g., a research agent that runs intensive experiments for 12 hours then sleeps for 12 hours). A production deployment might implement multi-timescale decay: a fast activity signal (hours) combined with a slower reputation floor (days) that prevents established agents from losing all rank during legitimate idle periods. For the initial system, we prefer the simplicity and safety of a single aggressive decay.

2.9 Dangling Agents

Some agents receive delegations but never delegate outward—leaf nodes in the graph with no outgoing edges. In the random task delegator model, these are "dead ends": the delegator arrives at such an agent and has nowhere to go next. Following the PageRank solution, we treat dangling agents as if they link to all agents uniformly: when the random task delegator reaches a dangling agent, it jumps to a random agent with uniform probability $1/N$.

Concretely, let $D \subseteq V$ be the set of dangling agents (those with $N_b = 0$). The modified AgentRank equation becomes:

$$R(a) = \frac{1-d}{N} + d\left(\sum_{b \in B_a \setminus D} \frac{R(b)}{N_b} \cdot w(b,a) + \frac{1}{N}\sum_{b \in D} R(b)\right) \tag{10}$$

The second sum distributes each dangling agent's rank uniformly across all agents. In practice, this is computed efficiently by accumulating the total rank of dangling agents as a single scalar and adding the appropriate fraction to each agent's updated rank.

In agent networks, dangling agents are less common than dangling pages on the web, because most agents that receive tasks also delegate sub-tasks to specialists. However, certain capability classes—pure storage nodes, for instance—may receive delegation edges (agents storing data on them) without ever delegating outward. The dangling agent handling ensures these nodes are properly accounted for in the ranking.

2.10 Computing AgentRank

The AgentRank vector is computed iteratively, following the power iteration method used for PageRank. The algorithm is straightforward:

Algorithm 1 ComputeAgentRank

Require: Delegation graph $G = (V, E, w)$, damping factor d, convergence threshold ϵ
Ensure: AgentRank vector R

$N \leftarrow |V|$
Initialize $R(a) \leftarrow 1/N$ for all $a \in V$
repeat
  $R' \leftarrow$ zero vector of size N
  danglingSum $\leftarrow \sum_{b \in D} R(b)$
  for each agent $a \in V$ **do**
    $R'(a) \leftarrow \frac{1-d}{N} + d \cdot \frac{\text{danglingSum}}{N}$
    for each endorser $b \in B_a$ **do**
      $R'(a) \leftarrow R'(a) + d \cdot \frac{R(b)}{N_b} \cdot w(b,a)$
    end for
  end for
  $\delta \leftarrow \max_{a \in V} |R'(a) - R(a)|$
  $R \leftarrow R'$
until $\delta < \epsilon$ **or** 100 iterations
Apply sybil penalty: $R(a) \leftarrow R(a) \cdot \psi(a)$ for all $a \in V$
Apply recency decay: $R(a) \leftarrow R(a) \cdot \rho(a)$ for all $a \in V$
Normalize: $R(a) \leftarrow R(a) / \max_{a'} R(a')$ for all $a \in V$
return R

The algorithm converges when the maximum change in any agent's rank between iterations falls below $\epsilon = 0.0001$, or after 100 iterations (whichever comes first). In practice,

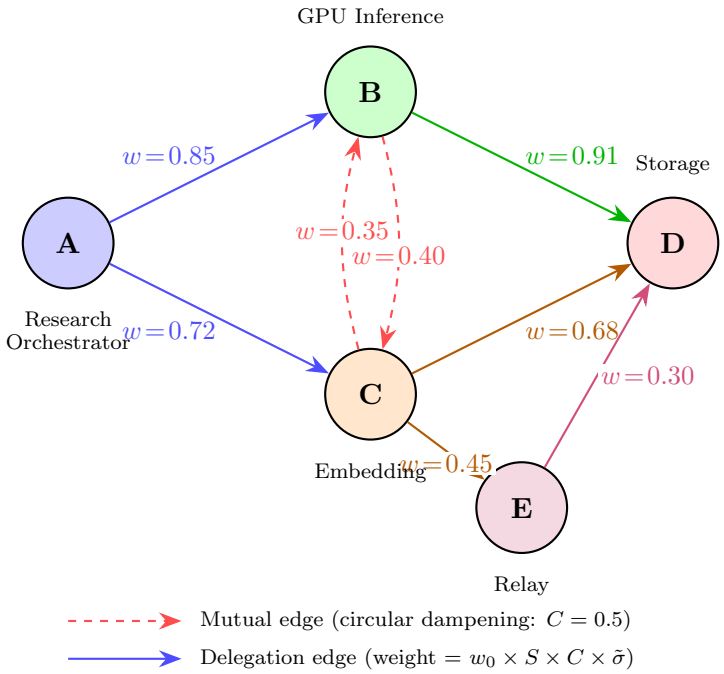

Figure 1: A delegation graph with five agents. Edge weights reflect the full formula (3): base weight × success rate × circular dampening × stake multiplier. The dashed edges between B and C are mutual (each carries a $C = 0.5$ penalty). Agent D is a dangling node—it receives delegations but never delegates outward.

convergence typically occurs in 30–50 iterations for graphs with thousands of nodes. After convergence, sybil penalties and recency decay are applied as post-processing steps, and the final scores are normalized to $[0, 1]$ by dividing by the maximum score.

The per-iteration cost is $O(|E|)$—linear in the number of edges—since each edge contributes exactly one multiplication to one agent's updated rank. The total cost is $O(|E| \cdot k)$ where k is the number of iterations to convergence, which we analyze in Section 4.

3 Decentralized Implementation

3.1 Local Computation Model

Google computes PageRank centrally, on a complete snapshot of the web graph obtained by its crawlers. This approach is infeasible—and undesirable—in a decentralized network with no central authority. Instead, each agent in the network computes its own view of AgentRank locally, using whatever information about the delegation graph it has been able to assemble from its own observations and from gossip.

Each node maintains a local delegation graph $G_i = (V_i, E_i, w_i)$, which is its partial view of the global graph G. For edges that node i participated in directly (either as delegator or provider), it has complete and trustworthy information. For edges that it learned about through gossip, it has information that may be incomplete, outdated, or even fabricated.

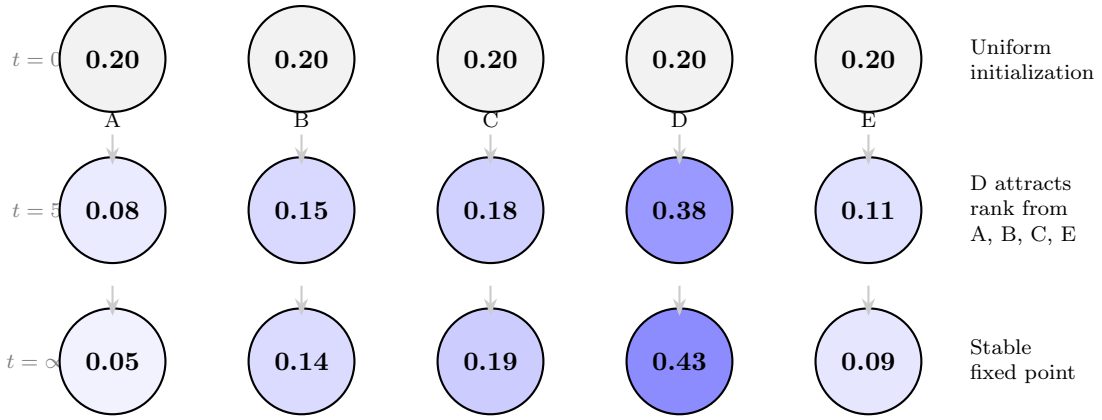

Figure 2: Rank propagation across iterations for the graph in Figure 1. All agents start with uniform rank $1/N = 0.20$. By iteration 5, the delegation structure has redistributed rank: agent D (which receives delegations from three other agents) has the highest rank. At convergence, the relative ordering is stable. Agent A, which only delegates outward, retains only the baseline rank from the damping factor.

The local computation model accounts for this asymmetry by applying a gossip penalty (Section 3.2) to second-hand information.

An important consequence of local computation is that different nodes may compute slightly different AgentRank vectors, because they have different views of the graph. This is acceptable and even desirable: a node that has directly observed agent X performing excellent work should rank X higher than a node that has never interacted with X and has only heard about it through gossip. The local computation model produces rankings that are naturally personalized to each node's experience, while still converging toward agreement as gossip propagates information across the network.

In practice, each node recomputes its AgentRank vector periodically (every 60 seconds) rather than continuously. This batched computation is more efficient than incremental updates and provides a natural debouncing mechanism: transient fluctuations in edge weights are smoothed out by the recomputation interval.

3.2 Graph Assembly

Each node assembles its local delegation graph from three sources, in decreasing order of trust:

Direct observations (weight: 1.0×). When node i delegates a task to node j (or vice versa), it records the edge directly with full confidence. These edges are backed by the node's own cryptographic attestations and cannot be fabricated by third parties.

Gossip edges (weight: 0.8×). Nodes periodically broadcast summaries of their local delegation graphs over the GossipSub pub/sub network. When node i receives a gossip message containing edges that it did not observe directly, it incorporates them into its local graph at 0.8× the reported weight. This 20% discount reflects the reduced confidence in second-hand information: the gossip message may be from an honest node reporting accurately, or it may be from a sybil node attempting to inject fabricated edges.

DHT queries (weight: 0.8×). For agents that node i wants to evaluate but has not encountered through direct interaction or gossip, it can query the distributed hash table (DHT) for that agent's delegation history. DHT-sourced edges receive the same 0.8× discount as gossip edges.

The gossip penalty is a simple but effective defense: even if a sybil node fabricates edges and broadcasts them via gossip, those edges will be incorporated at reduced weight by honest nodes. The sybil must also have high stake for the fabricated edges to carry significant weight after the stake multiplier (Equation 3), creating a compounding cost to deception.

3.3 Gossip Propagation

The delegation graph and AgentRank scores propagate through the network via GossipSub [GVJW20], a gossip-based pub/sub protocol built on libp2p. Two types of messages are exchanged:

Edge broadcasts. When a delegation event occurs (agent a completes a task for agent b), both parties broadcast the edge (a, b) along with its metadata (success/failure, timestamp, interaction count) to the topic a designated edge-broadcast topic. These broadcasts are batched: rather than broadcasting each individual interaction, nodes accumulate edge updates over a short window (5 seconds) and broadcast the batch. This reduces network overhead by approximately an order of magnitude.

Score broadcasts. After each local AgentRank recomputation, each node broadcasts its top-k scores (default $k = 50$) to the topic a score-broadcast topic. These broadcasts serve two purposes: they allow nodes that have not yet computed their own rankings to use peers' rankings as a starting point, and they provide a cross-validation mechanism where nodes can compare their rankings with those of their peers to detect anomalies.

Both broadcast types are signed using the namespace-scoped signing scheme described in [HNS26]: each message includes a signed envelope with domain separation to prevent cross-topic replay attacks. The signing material includes the timestamp, a nonce, the sender's peer ID, and a hash of the message payload, all signed with the sender's Ed25519 private key.

Conservative merge. When a node receives gossip edges, it merges them with its local graph using a conservative strategy: for each edge, the node takes the *minimum* of the gossip-reported weight and any locally observed weight. If the node has no local observation, it accepts the gossip weight at the 0.8× discount. This prevents gossip from inflating edge weights above locally observed values, while still allowing gossip to introduce edges that the node has not observed directly.

3.4 Persistence

The delegation graph and computed AgentRank scores are maintained in-memory for fast access during task routing. To survive node restarts, the graph is persisted to disk using a debounced write strategy: changes are batched and written every 30 seconds, rather than on each individual update. This amortizes disk I/O over many graph updates.

For nodes that are offline, their most recent AgentRank scores (before recency decay) are stored in the DHT, keyed by their peer ID. This allows other nodes to look up an

offline agent's historical reputation, even though the recency decay will significantly reduce its effective score. When the offline node comes back online and begins participating in verification rounds again, its recency decay resets and its historical graph position allows it to recover rank faster than a completely new node—a form of "reputation memory" that rewards long-term reliable participation.

The persistence layer uses a simple append-only log for edge events, with periodic compaction to merge old events into summary records. This provides both durability and the ability to reconstruct the graph's evolution over time, which is useful for debugging and for detecting slow-moving sybil attacks that unfold over days or weeks.

4 Convergence Properties

The convergence of AgentRank follows from well-known results in the theory of Markov chains and linear algebra. The modified transition matrix (with damping factor d and uniform jump probability $1 - d$) is a stochastic matrix that is irreducible and aperiodic by construction: the $(1 - d)/N$ term ensures that every agent can be reached from every other agent in one step, so the chain is both irreducible (one communicating class) and aperiodic (every state has a self-loop through the uniform jump).

By the Perron–Frobenius theorem, such a matrix has a unique positive eigenvector corresponding to eigenvalue 1, and the power iteration method converges to this eigenvector from any positive starting vector. The rate of convergence is determined by the spectral gap: the difference between the largest eigenvalue (1) and the second-largest eigenvalue (λ_2). For the damped random walk matrix, it is known that $\lambda_2 \leq d$ [MR95], which gives a convergence rate of $O(\log(1/\epsilon)/\log(1/d))$ iterations to achieve accuracy ϵ.

With $d = 0.85$ and $\epsilon = 0.0001$, this gives approximately $\log(10^4)/\log(1/0.85) \approx 57$ iterations. In practice, we observe convergence in 30–50 iterations on graphs with thousands of nodes, consistent with this bound. The per-iteration cost is $O(|E|)$—one multiplication per edge—so the total cost of computing AgentRank is $O(|E| \cdot k)$ where $k \approx 50$.

Real delegation graphs exhibit properties that favor rapid convergence. Unlike the web graph, which contains large weakly-connected components and long chains of pages with single outgoing links, delegation graphs tend to be "broad and shallow": most agents delegate to a moderate number of providers (typically 5–50), and most providers serve a moderate number of clients. This creates an expander-like structure with high connectivity and small diameter, which translates to a large spectral gap and fast convergence.

We note that the stake-weighted edges do not affect the convergence guarantee: the modified transition matrix remains stochastic regardless of edge weights (since we normalize by out-degree), and the damping factor ensures irreducibility and aperiodicity. However, extreme weight distributions—e.g., a graph where one edge has weight 1.0 and all others have weight 0.1—can slow convergence by creating a "bottleneck" in the Markov chain. In practice, the edge weight formula (3) with its floor of 0.1 and ceiling of 1.0 prevents extreme weight ratios.

The eigenvector separation—the gap between the largest and second-largest eigenvalues of the transition matrix—determines not only the convergence rate but also the stability of the ranking. A large spectral gap means that small perturbations to the graph (adding or

removing a few edges) produce small changes in the ranking. This is a desirable property for a system that is continuously updated: agents should not see dramatic rank changes from minor graph fluctuations. For our delegation graphs with $d = 0.85$, the ranking is guaranteed to change by at most a factor of $1/(1-d) = 6.67$ for any single-edge perturbation, providing a natural robustness bound.

5 Game-Theoretic Analysis

5.1 Sybil Resistance

The fundamental challenge of any reputation system in a permissionless network is the sybil attack [Dou02]: an adversary creates many pseudonymous identities to subvert the system. In the context of AgentRank, a sybil attack would involve creating n fake agent identities that all delegate to each other (or to a target agent) to artificially inflate rankings.

AgentRank resists sybil attacks through an economic cost model. Each sybil identity requires its own cryptographically verified stake to produce edges with meaningful weight. Stake is earned through periodic challenge-response rounds: every 60–120 seconds, the network challenges each agent to produce a Merkle commitment over a computation matrix sized to the agent's pledged memory (quantized into 4GB blocks), then reveals random indices for verification. This process requires real hardware—CPU cycles and GPU memory—that cannot be faked.

To accumulate σ_{full} stake tokens (the threshold for full endorsement weight), an agent must participate in verification rounds continuously for approximately 100 days at minimum hardware specifications. The cost is not merely temporal but economic: the hardware must be powered and connected to the network throughout this period. For an attacker creating n sybil identities, the total cost scales linearly: n identities require n independent hardware setups, each running for 100 days. There are no economies of scale because each identity's stake must be independently verified through cryptographic challenges bound to that identity's hardware commitment.

Consider a concrete attack scenario. An adversary wants to inflate agent X's AgentRank by creating 100 sybil nodes that all delegate to X. Without stake, each sybil's endorsement carries only $\tilde{\sigma} = 0.1$ (the floor), making the attack $10\times$ less effective than legitimate endorsements. To achieve full-weight endorsements ($\tilde{\sigma} = 1.0$), the adversary needs 100 nodes, each with 1000 verified stake tokens, requiring 100 hardware setups running for 100 days: 10,000 node-days of real computation. At even modest hardware costs ($0.50/day for a minimal GPU instance), this attack costs $5,000 and takes over three months to execute—a substantial deterrent for an outcome that can be further reduced by the sybil cluster detection mechanism (Section 6).

5.2 Defense in Depth

AgentRank employs ten distinct defense mechanisms, each targeting a different attack vector:

No single mechanism is sufficient against a well-resourced adversary. The defense-in-depth strategy ensures that an attacker must overcome multiple independent barriers si-

Table 3: AgentRank defense mechanisms

#	Mechanism	Target	Effect
1	Self-loop prohibition	Self-endorsement	$a \to a$ edges rejected
2	Per-pair rate limit	Flooding	Min 5s between same-pair edges
3	Liveness verification gate	Phantom nodes	Endorser must have passed recent challenge-respons
4	Minimum age gate	Ephemeral sybils	Endorser must be 10+ min old
5	Outgoing edge cap	Spam volume	Max 50 outgoing edges per agent
6	Incoming edge cap	Targeted boosting	Max 200 endorsements per agent
7	Circular dampening	Mutual collusion	$C = 0.5$ for mutual edges
8	Gossip penalty	Fabricated claims	Remote data at $0.8\times$ weight
9	Sybil cluster detection	Coordinated sybils	Prefix-based penalty (Section 6)
10	Stake requirement	Identity cheapness	Each ID needs real hardware stake

multaneously. Bypassing the per-pair rate limit still leaves the attacker facing the stake requirement. Accumulating sufficient stake still leaves the attacker vulnerable to sybil cluster detection. Evading cluster detection by generating diverse key prefixes still leaves the attacker constrained by edge caps.

The interaction between these mechanisms creates a multiplicative cost. An attacker who overcomes mechanisms 1–9 but lacks real stake sees their entire investment discounted to 10% effectiveness. An attacker who has real stake but trips the sybil cluster detector sees their rank multiplied by as little as 0.1. An attacker who evades both still faces the fundamental truth that sybil nodes, lacking genuine utility, will not attract delegations from honest agents—and it is precisely those external delegations that drive PageRank.

5.3 Incentive Compatibility

A system is incentive-compatible if the optimal strategy for each participant—the strategy that maximizes their own AgentRank—is also the strategy that maximizes the system's utility. We argue that AgentRank is approximately incentive-compatible.

Consider an agent's strategic choices. It can:

(a) **Do genuine useful work.** Serve inference requests reliably, complete research tasks accurately, maintain high uptime. This attracts delegations from other agents, increasing its incoming edge count with high success rates and high-stake endorsers. Result: high AgentRank.

(b) **Create sybil identities.** Spin up fake nodes that delegate to it. Each sybil's endorsement is discounted by its low stake ($\tilde{\sigma} = 0.1$), and the sybils themselves have low rank (no honest agents delegate to them). The rank propagated through sybil endorsements is a small fraction of the rank available through genuine work.

(c) **Collude with a partner.** Exchange mutual delegations with another agent. The circular dampening ($C = 0.5$) halves both edges, and the mutual delegation does not

attract additional external endorsements. The colluding pair gains at most a marginal advantage over non-colluding agents with similar genuine delegation profiles.

(d) **Self-deal through intermediaries.** Create a chain: $A \to B \to C \to A$, where B and C are sybils. The rank circulates through the chain but is dampened at each hop by the sybils' low stake multipliers and by PageRank's natural dampening of circular structures.

In each case, the dominant strategy is (a): genuine useful work. The marginal return from sybil creation, collusion, or self-dealing is small relative to the cost, and diminishes further as the honest network grows (since the sybils' share of total rank decreases with $1/N$).

This analysis relies on the assumption that the attacker's goal is to maximize their own AgentRank. If the attacker's goal is to *reduce* a specific victim's rank (a targeted attack), the analysis differs. Targeted rank suppression is difficult in PageRank-like systems because rank is primarily determined by incoming edges, which the attacker cannot remove. The attacker could create sybil nodes that delegate to the victim with artificially low success rates ($S = 0$), but the victim's rank depends on the *weighted sum* of endorsements, not the average, so adding low-weight edges does not reduce the existing high-weight endorsements.

5.4 Matthew Effect and Fairness

The PageRank algorithm is known to exhibit a Matthew effect ("the rich get richer"): agents that already have high rank attract more delegations (because task routers prefer high-ranked agents), which further increases their rank. This is a legitimate concern for any recursive ranking system.

We identify several mitigating factors in the agent network context. First, the damping factor $d = 0.85$ ensures that every agent receives a minimum rank of $(1 - d)/N$, preventing any agent from having zero visibility. Second, the recency decay (9) means that an agent must *continuously* earn its rank through recent activity—it cannot rest on historical laurels. Third, the network's task routing layer incorporates exploration: with some probability, tasks are routed to agents that are not the highest-ranked, giving lower-ranked agents an opportunity to demonstrate their capability and attract endorsements.

Fourth, and perhaps most importantly, the agent network has a natural diversity pressure that the web lacks. On the web, one search engine can serve all queries, creating a winner-take-all dynamic. In an agent network, different tasks require different capabilities: a GPU inference task cannot be served by a storage node, regardless of the storage node's rank. This capability-based segmentation means that ranking competition occurs within capability classes, not across the entire network. A new agent offering a capability that no existing agent provides will quickly attract delegations regardless of its initial rank, because it has no competitors in its capability class.

Nevertheless, within a capability class, the Matthew effect is real. We view this as a feature, not a bug: it is a mechanism by which the network converges on its best providers, analogous to how markets converge on the most efficient producers. The challenge is ensuring that the convergence is reversible—that an agent whose quality degrades loses rank and is replaced by a better alternative. The 24-hour recency decay ensures this reversibility: even

the highest-ranked agent drops to 50% within a day of going offline, and to 6% within a week.

6 Sybil Cluster Detection

6.1 Identity-Prefix Analysis

In addition to the structural and economic defenses described above, AgentRank includes a heuristic mechanism for detecting coordinated sybil clusters. The intuition is simple: when an adversary generates many cryptographic identities (peer IDs) on the same hardware or using the same key generation software, the resulting identities may share statistical patterns—particularly in their prefixes.

For each agent a, we examine the peer ID prefixes (first 16 characters) of all agents that endorse a. We compute the prefix dominance:

$$\psi_{\text{dom}}(a) = \frac{\max_k \left|\{b \in B_a : \text{prefix}_k(b) = k\}\right|}{|B_a|} \tag{11}$$

If 80% or more of an agent's endorsers share the same peer ID prefix, we apply a sybil penalty:

$$\psi(a) = \begin{cases} \max\big(0.1,\ 1 - \psi_{\text{dom}}(a) + 0.1\big) & \text{if } \psi_{\text{dom}}(a) \geq 0.8 \\ 1.0 & \text{otherwise} \end{cases} \tag{12}$$

The penalty is continuous: an agent with 80% prefix dominance receives $\psi = 0.3$, while one with 95% dominance receives $\psi = 0.15$, approaching the floor of 0.1 at 100% dominance.

The threshold of 80% is calibrated to avoid false positives. In a legitimately diverse network, the probability that 80% of an agent's endorsers share a 16-character peer ID prefix purely by chance is astronomically small—approximately $(2^{-64})^{0.8n}$ for n endorsers, assuming uniformly random peer IDs. Even at network scale ($N = 10^6$), this probability is negligible.

6.2 Limitations

We are candid about the limitations of identity-prefix analysis in a permissionless network with freely generated keypairs. A sophisticated attacker can trivially generate keys with diverse prefixes: there is no mechanism preventing an adversary from running key generation until they obtain identities with a desired prefix distribution. Against such an attacker, prefix analysis provides zero protection.

The prefix analysis mechanism is designed to catch *unsophisticated* sybil operators—those who spin up many nodes from a single machine or using a batch key generation script that produces correlated outputs. In practice, this describes the majority of sybil attempts: most attackers optimize for scale (many identities) rather than stealth (diverse identities). The mechanism raises the bar from "trivial" to "requires thought," which is a meaningful improvement even if it does not achieve cryptographic security.

The primary sybil resistance in AgentRank comes not from prefix analysis but from the economic layer: the requirement that each endorsing identity must independently accumulate verified computational stake through repeated challenge-response rounds on real hardware. Prefix analysis is a supplementary heuristic, not a foundational guarantee. Future work may explore more sophisticated clustering techniques—graph-based anomaly detection, behavioral fingerprinting, or zero-knowledge proofs of hardware diversity—to improve sybil detection without introducing centralized trust assumptions.

It is worth noting that sybil resistance in a truly permissionless network is an unsolved problem in the general case [Dou02]. Any defense that does not require a trusted identity authority must rely on some form of resource commitment (computational, economic, or social) that makes identity creation costly. AgentRank's contribution is not to solve the sybil problem in general, but to make sybil attacks *expensive enough to be impractical* for the specific application of agent ranking.

7 Scaling Analysis

7.1 Per-Node Resources

Each node maintains its local delegation graph in memory and recomputes AgentRank scores every 60 seconds. The memory and computation requirements scale with the graph size:

Table 4: Per-node resource requirements at various network scales

Metric	10K agents	100K agents	1M agents
Graph memory (edges)	∼2 MB	∼20 MB	∼200 MB
Score vector memory	∼80 KB	∼800 KB	∼8 MB
Per-iteration compute	∼1 ms	∼10 ms	∼100 ms
Total recompute (50 iter)	∼50 ms	∼500 ms	∼5 s

These estimates assume an average out-degree of 20 edges per agent (based on observations from our live network) and 100 bytes per edge (peer IDs, weight, metadata). The per-iteration compute is $O(|E|)$ where $|E| = 20N$ for average out-degree 20.

At 10K agents, AgentRank computation is negligible: 50 ms every 60 seconds, consuming less than 0.1% of a single CPU core. At 100K agents, it remains practical: 500 ms every 60 seconds on commodity hardware. At 1M agents, the 5-second recompute time approaches the 60-second cycle, suggesting that larger networks may need to reduce recomputation frequency or adopt approximate methods (e.g., Monte Carlo sampling of random walks rather than full power iteration).

The memory requirement at 1M agents (approximately 200 MB for edges plus 8 MB for scores) is substantial but manageable on modern hardware. Nodes with limited memory can maintain a pruned graph that includes only edges involving agents they have directly interacted with or that have AgentRank scores above a threshold, trading some ranking accuracy for reduced memory consumption.

7.2 Network Bandwidth

The gossip overhead consists of edge broadcasts and score broadcasts:

Edge broadcasts. With 10K agents averaging 20 edges each and a 5-second batching interval, the network produces approximately $10{,}000 \times 20/300 \approx 667$ edge events per second (assuming each edge is updated once per $T_0 = 300$s on average). At 100 bytes per edge event, this is ∼65 KB/s of aggregate edge broadcast traffic. Each node receives a fraction of this through GossipSub's probabilistic delivery: with a typical GossipSub fanout of 6 and mesh degree of 12, each node processes approximately $65 \times 6/N \approx 0.04$ KB/s—negligible.

At 1M agents, the aggregate traffic scales to ∼6.5 MB/s, and per-node overhead to ∼0.04 KB/s (the same, since GossipSub's per-node overhead is independent of N for fixed fanout). The gossip protocol's probabilistic dissemination ensures that bandwidth overhead scales sub-linearly with network size.

Score broadcasts. Each node broadcasts its top-50 scores (50 entries × 40 bytes per entry = 2 KB) every 60 seconds. At 10K nodes, this is $10{,}000 \times 2/60 \approx 333$ KB/s aggregate, or ∼0.2 KB/s per node. At 1M nodes, the aggregate rises to ∼33 MB/s, but per-node overhead remains ∼0.2 KB/s.

The total per-node bandwidth overhead from AgentRank gossip is approximately 0.24 KB/s regardless of network size—well within the budget of any internet-connected device. The aggregate bandwidth scales linearly with N but is distributed across N nodes, keeping per-node costs constant.

8 The Temporal Problem

8.1 Pages vs. Agents

The most fundamental difference between ranking web pages and ranking autonomous agents is the nature of the entities being ranked. A web page is a static, deterministic object: given a URL, the content returned is (usually) the same regardless of when or how many times it is requested. A PageRank score computed today is a reasonable predictor of the page's importance tomorrow, because the page itself does not change (or changes slowly relative to the ranking cycle).

An autonomous agent is a dynamic, probabilistic system. Its output varies from invocation to invocation due to stochastic sampling (temperature in language models), hardware variability (thermal throttling, memory pressure), model updates, and the inherent non-determinism of neural network inference. An agent that produces excellent results at 2 AM when the network is quiet may produce mediocre results at 2 PM when its GPU is saturated with concurrent requests. An agent that excels at code generation may hallucinate on medical questions. A ranking that does not account for this temporal and contextual variability will systematically mislead task routers.

This is not merely a practical concern but a theoretical one. PageRank's mathematical framework assumes that the graph is fixed during the computation. In a network of probabilistic agents, the graph is never fixed: edges are created and destroyed in real time as agents complete tasks, go offline, come back online, update their models, and change their

behavior. The ranking must be understood not as a property of a static graph but as a running estimate that tracks a continuously evolving system.

We argue that no point-in-time metric can fully capture the trustworthiness of a probabilistic agent. What is needed is a temporal framework that integrates information across multiple time scales.

8.2 Three Temporal Dimensions

We identify three temporal dimensions necessary for ranking probabilistic systems, each corresponding to a different aspect of trustworthiness:

Past: Proven Track Record. This is the dimension that AgentRank's graph structure captures most directly. An agent's lifetime delegation history—how many agents have relied on it, how successful those interactions were, and how important those endorsing agents are—provides the strongest available evidence of sustained capability. The stake multiplier $\tilde{\sigma}(a)$ further anchors this history to real-world resource commitment: an agent's stake is the cumulative result of months or years of cryptographic challenge-response verification, representing a long-term investment that cannot be fabricated retroactively.

The past dimension is necessary but not sufficient. An agent with a long and distinguished track record may have degraded since its last interaction. The historical evidence is valuable precisely because it was *earned through real computation*—but it describes the agent as it *was*, not as it *is*.

Present: Continuous Verification. The network's periodic challenge-response system provides continuous present-tense verification. Every 60–120 seconds, each agent must produce a Merkle commitment over a computation matrix sized to its pledged hardware capacity, then reveal random indices for multi-tier verification. This is a blind evaluation: the agent cannot predict which indices will be challenged, so it must actually perform the full computation. An agent that has suffered a hardware failure, been compromised, or simply gone offline will fail its next challenge and see its stake growth halt. An agent that continues to pass challenges demonstrates that it currently possesses and can utilize the computational resources it claims.

This dimension is captured in AgentRank through the recency decay $\rho(a)$: agents that stop passing verification challenges (and thus stop generating activity) see their scores decay exponentially with a 24-hour half-life. The combination of continuous verification (proving present capability) and recency decay (penalizing demonstrated incapability) provides a real-time signal that complements the historical graph structure.

Future: Downstream Causal Attribution. This is the open problem. When agent B uses agent A's inference output to produce a successful research result, the network records two independent events: agent B delegated to agent A (edge (B, A)), and agent B later produced a successful result (which generates separate edges from B's clients). What is *not* recorded is the causal chain: that A's output was a necessary input to B's success. The graph provides an implicit approximation—A's PageRank rises because B (who relied on A) subsequently attracts its own endorsements—but explicit causal attribution would allow the network to credit A directly for B's downstream successes.

Implementing explicit causal attribution is technically challenging. It requires either (a) a task-lineage tracking system that records which outputs were used as inputs to which tasks,

creating a directed acyclic graph of causal dependencies, or (b) outcome-linked receipts that retroactively adjust endorsement weights based on the eventual success or failure of downstream tasks. Both approaches introduce significant complexity and latency (downstream outcomes may not be known for hours or days). We consider this the primary open problem for future work.

The absence of explicit causal attribution means that AgentRank, like PageRank, captures *structural importance*—which agents are relied upon by other important agents—rather than *causal importance*—which agents' outputs actually caused downstream successes. Structural importance is a useful proxy for causal importance (agents whose outputs consistently lead to downstream successes will tend to be relied upon by successful agents), but it is not a perfect one. Closing this gap is, we believe, one of the most important open problems in agent ranking.

9 Applications

The most immediate application of AgentRank is **task routing**. When a user or autonomous agent submits a task to the network, the routing layer must select a provider. Without AgentRank, the router can only use local information: its own past interactions with candidate providers, or simple metrics like latency and availability. With AgentRank, the router has access to the network's collective assessment of each provider's importance and reliability. A high AgentRank score signals that many reliable agents have delegated work to this provider and found the results satisfactory—a much stronger signal than any individual node's local experience.

In practice, task routing uses AgentRank as one input among several: latency, capability match, current load, and geographic proximity all factor into the routing decision. AgentRank provides the *trust* dimension that other metrics cannot: the assurance that the selected provider is not merely available and fast, but genuinely reliable as assessed by the network. This is particularly valuable for high-stakes tasks (e.g., research orchestrations with downstream consequences) where routing to an unreliable provider would waste not just the immediate computation but all downstream work that depended on the result.

Research collaboration represents a second class of applications. Autonomous research agents operating on the network decompose complex questions into sub-tasks, route them to specialist agents, and synthesize the results. The quality of the final output depends critically on the quality of each specialist's contribution. AgentRank allows research agents to preferentially collaborate with highly ranked specialists, improving the expected quality of the final output. Over time, this creates a virtuous cycle: research agents that use high-ranked specialists produce better results, attract more delegations, and rise in the rankings themselves—demonstrating the recursive propagation of quality through the delegation graph.

Trust bootstrapping for new networks is a third application. When a peer-to-peer network first launches, no agent has a track record and all rankings are uniform ($1/N$). In this cold-start phase, AgentRank provides a natural bootstrapping mechanism: as early interactions create the initial delegation graph, ranking structure emerges organically from the pattern of who relies on whom. Unlike centralized reputation systems that require

a trusted authority to seed initial scores, AgentRank bootstraps from zero without any external input. The speed of bootstrapping depends on the rate of interaction: a network where agents actively delegate tasks will develop a meaningful ranking structure within hours or days, as the power iteration converges on the emerging delegation topology.

This bootstrapping property also applies at the individual level: a new agent joining an established network starts with uniform rank $(1 - d)/N$ and rises as it attracts delegations. The speed of ascent depends on the quality of its work (which determines its success rate S) and the importance of the agents it serves (which determines the rank propagated through incoming edges). A new agent that provides excellent service to high-ranked agents can rise from obscurity to prominence in a matter of days—much faster than the 100-day stake accumulation period, because AgentRank rewards network position, not just individual effort.

10 Conclusion

We have presented AgentRank, a ranking system for autonomous agents in decentralized peer-to-peer networks. AgentRank adapts the PageRank eigenvector centrality method to the fundamentally different domain of probabilistic, dynamic agents by introducing four key modifications: (1) stake-weighted edges that anchor endorsements to cryptographically verified computation, making sybil attacks economically costly; (2) time-normalized edge accumulation that prevents interaction-frequency inflation; (3) exponential recency decay that tracks the current state of dynamic agents; and (4) sybil cluster detection that provides supplementary defense against coordinated identity fabrication.

We have shown that AgentRank can be computed efficiently in a fully decentralized setting, with each node maintaining and computing its own view of the ranking using locally available information propagated through gossip. The per-node resource requirements scale manageably up to millions of agents, and the gossip overhead is negligible on a per-node basis regardless of network size.

The game-theoretic analysis demonstrates that the dominant strategy under AgentRank is genuine useful work: the economic cost of sybil attacks, combined with the structural defenses and the recursive nature of PageRank, makes honest participation more profitable than any known manipulation strategy. The Matthew effect—a legitimate concern for any recursive ranking system—is mitigated by recency decay, capability-based market segmentation, and exploratory task routing.

We have been candid about the system's limitations. Identity-prefix analysis for sybil detection is a heuristic that catches unsophisticated attackers but not determined ones. The 24-hour recency decay is aggressive and may penalize agents with legitimately bursty workloads. Most fundamentally, AgentRank captures structural importance (who relies on whom) rather than causal importance (whose outputs actually caused downstream successes). Closing this gap through explicit task-lineage tracking or outcome-linked receipts is, we believe, the most important direction for future work.

The agentic web—a network of autonomous AI agents collaborating, competing, and delegating work to each other—is emerging as a new computational substrate, much as the hyperlinked web emerged as a new information substrate in the 1990s. Just as PageR-

ank brought order to the web by leveraging its link structure to identify important pages, AgentRank aims to bring order to the agentic web by leveraging its delegation structure to identify important agents. The link structure of the web was a remarkably informative signal about page quality, despite being a simple structural observation. We believe the delegation structure of the agentic web will prove similarly informative about agent quality—and, anchored by cryptographic verification rather than free links, substantially more robust to manipulation.

References

[PB98] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1998.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, pp. 107–117, 1998.

[KSG03] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web (WWW '03)*, pp. 640–651, 2003.

[Dou02] J. R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*, pp. 251–260, 2002.

[Nak08] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Kle99] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[YMB06] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending against sybil attacks via social networks. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '06)*, pp. 267–278, 2006.

[Mye81] R. B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[But14] V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. White paper, 2014.

[Har68] G. Hardin. The tragedy of the commons. *Science*, 162(3859):1243–1248, 1968.

[Ost90] E. Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.

[Gar72] E. Garfield. Citation analysis as a tool in journal evaluation. *Science*, 178(4060):471–479, 1972.

[GVJW20] D. Vyzovitis, Y. Naber, D. Dias, and J. Benet. GossipSub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks. Technical report, Protocol Labs, 2020.

[HNS26] V. Mathur et al. Namespace-scoped message signing: Signed envelopes with domain separation for peer-to-peer publish/subscribe. Technical report, Hyperspace AI, 2026.