The Dance and the Field: Name Semantics for Handoffs Between Distributed Agents

Chetan Conikee
Modiqo
chetan@modiqo.ai

Abstract

Agent harnesses are becoming distributed systems: a single task now fans out across dozens of subagents, within one process, across the harnesses on a machine, and increasingly over the network. Yet the mechanism these systems use to pass work between agents is the one that distributed systems abandoned decades ago—*copy the bytes and hope*. A report produced by one agent is pasted into the context of the next, re-billed on every subsequent turn of a stateless API, duplicated across every recipient, and stripped of identity, version, and any record of whether it was read. Copies are the root defect: a copy has no owner, no lifecycle, and no receipt, and roughly a third of multi-agent failures trace to agents acting on divergent copies of what should have been the same information.

We present WAGGLE, a substrate built on *name semantics*. A handoff is a ∼30-byte attributed token, not a payload. Consumers do not receive content; they *resolve* the token into a projection computed for their own context, *interrogate* it under byte budgets where the bytes live, and leave *receipts* in an append-only, payload-free log. The same five verbs operate unchanged at three radii—inter-process, inter-harness, and inter-machine—which is the paper's central systems argument: portability of a handoff cannot be an optimization applied to copies; it must be a property of the reference. We ground the design in a lineage the agent literature has overlooked: tuple spaces, named-data networking, capabilities, leases, and stigmergy. We describe a sealed context-adaptive matcher, a consumption-contract mechanism that turns "did the subagent actually read it?" from an unanswerable question into a query, and a mint-time structural lens for source code that gives an agent a symbol-addressable handoff without any parser on the serving path. We report microbenchmarks (39 ns cache-hit resolution, 323 µs end-to-end over a domain socket, a million-event fold in 334 µs) and a live delegation case study, and we are candid about the mechanism's limits.

1 Introduction

Begin with the fact that shapes everything else: large-language-model APIs are *stateless*. Each turn of a conversation re-sends the entire conversation. When an agent places a 40 KB report into its context, it does not pay for that report once; it pays on every subsequent turn of that agent's life, because the whole history rides along each time. Now add a second agent. An orchestrator receives a report and forwards it to a writer subagent; the report now occupies *two* context windows, each re-billing it per turn. Add a fact-checker: three. This is not a defect of any one framework—it is the default shape of the ecosystem. LangGraph's handoff tool "by default passes the full message history" [15]; Anthropic, engineering their own multi-agent research system, measured agents at ∼4× the tokens of chat and multi-agent systems at ∼15×, attributing the overhead to "duplicating context across agents" and writing the sentence this work is built on: *"each handoff loses context"* [3]. The MAST failure taxonomy attributes **36.9%** of multi-agent failures to inter-agent misalignment—agents acting on divergent, stale, or partial copies of what should have been the same information [6].

Copies are the root defect. A copy has no identity, no version, no owner, no lifecycle. The moment the author corrects the report, every pasted copy is silently wrong, and no mechanism exists to find them. The token cost is what everyone feels; the missing lineage and the impossibility of correction are what actually break swarms.

Thesis. Passing work between computational actors admits exactly three disciplines. *Copy semantics* (message passing) sends the bytes—simple, and every pathology above follows. *Place semantics* (shared memory) has both parties touch one location—this fixes duplication but demands shared infrastructure and says nothing about who may see what. *Name semantics* sends a small, immutable, attributed *claim* and lets resolution be computed per consumer, at the data, on demand. Distributed systems learned to prefer the third discipline for exactly the reasons agent systems are now rediscovering. This paper argues that the agent handoff is a distributed-systems problem, that name semantics is its right shape, and that a substrate built on names makes a handoff behave identically whether its two ends sit in one process or on two continents. We substantiate the argument with a working system, WAGGLE [19], and its measurements.

Contributions.

- A framing of agent handoffs as a distributed-systems problem, with a four-boundary analysis (§2) showing that every boundary loses lineage and cannot propagate corrections.
- A substrate for *name semantics* (§4): a 30-byte attributed token, a three-zone signed manifest, a sealed context-adaptive matcher (Alg. 1), and an append-only payload-free log with an exact-reconstruction guarantee.
- *Receipts* (§5): consumption contracts and a coverage fold (Alg. 2) that make delegated consumption verifiable without trusting the consumer's self-report.
- A *structural lens* for source code (§6) computed once at mint, so no parser ever runs on a serving path—including the network edge.
- The *three-radii* result (§7): the identical verb loop inter-process, inter-harness, and inter-machine.
- An evaluation (§8) of microbenchmarks and a live delegation, an honest comparison against lexical search, and a frank account of limits (§9).

2 The Handoff Problem

"Passing a resource between agents" is four different problems wearing one name, distinguished by which boundary the resource crosses. Table 1 reads, column by column, as a ledger of what is lost. The two rightmost columns are empty top to bottom: no boundary today carries lineage, and none propagates a correction. That emptiness is the problem this paper attacks.

Why provider optimizations do not save you. Every vendor is attacking the token cost inside its own walls: prompt caching discounts byte-identical prefixes within a short TTL and a single account; harness compaction summarizes a long context locally and lossily. These are real savings. But each is scoped to one vendor's billing and serving boundary. A cached prefix at one provider means nothing at another's tokenizer; a compaction summary in one harness does not exist in another. Cross any row of Table 1 and you pay full freight again, because what crossed was a copy of bytes, and bytes carry no identity a foreign system could recognize. This is the structural case for name semantics: portability cannot be an optimization applied to copies. It must be a property of the reference. A 30-byte token is the same 30 bytes in every context window, every tokenizer, every vendor, every machine; what varies is what it *resolves to*, computed fresh, per consumer, where the bytes live.

3 What the Colony Knew

A honeybee returns from a field two kilometres out carrying information worth sharing: where, how far, how good. She does not carry the field home, nor enough nectar for the colony to evaluate. She *dances* [24]. The waggle run's angle against vertical encodes direction relative to the sun; its duration encodes distance; its vigour encodes quality. Then the colony does something every distributed-systems engineer should study: each follower resolves the reference *herself*—flies her own flight, with her own senses, from her own position. The dance is not the nectar; it is an attributed, resolvable claim that the nectar exists. Recruitment is measurable, one can count who followed and who returned to dance in turn, and the information expires honestly: bees dance only while the source still pays, so no bee chases stale copies of yesterday's directions, because there were never any copies to chase.

This is *stigmergy*: coordination through durable marks rather than direct messages [10, 23]. The mapping to a handoff substrate is structural, not decorative (Table 2). The colony solved the multi-agent handoff problem with zero shared context windows: share names, not payloads; let each consumer resolve per its own capability; make consumption observable; let stale claims die at the source.

4 Design: Name Semantics

4.1 The token and the manifest

A *token* is 4–23 characters of Bitcoin base58 (default 8 public, 16 for private capability URLs), generated by rejection sampling to avoid modulo bias. It is the same 30 bytes in every tokenizer. Behind it stands an *attribution manifest* in three zones with three mutability rules, a separation that is load-bearing:

Immutable core
: set at mint, never changed: schema, token, target, sharer, channel, mint time, snapshot metadata, parent, the content-addressed snapshot reference, variants, and—optionally—a consumption contract (§5) and a symbol-outline reference (§6). Signatures cover exactly this zone.

Versioned mutable
: compare-and-swap by version: expiry, revocation, supersession. Every lifecycle change cites the version it was decided against.

Cosmetic mutable
: last-writer-wins: campaign and labels.

Because signatures (Ed25519 over the canonical core bytes) cover only the immutable zone, lifecycle and cosmetic churn never invalidate a signature—the three-zone design's payoff, and a direct descendant of capability discipline [8, 18]. Content over 64 KiB rides as a content-addressed reference [4, 17] rather than inline, so the manifest stays small and the bytes stay verifiable wherever they replicate.

Table 1: The four boundaries of an agent handoff. Read the last two columns top to bottom: they are empty. Lineage and correction do not survive any boundary today.

Boundary	How it works today	What travels	Lineage	Corrections
Same harness, same model (orchestrator → subagent)	Final message returns to the orchestrator, which pastes what the next subagent needs; or shared filesystem paths by convention	The full text again, per recipient; a path travels cheaply but carries no version, access story, or receipt	None—prose memory, gone at compaction	Manual re-paste; stale copies persist silently
Same harness, mixed models (router / cascade)	The same paste, now crossing a billing boundary	The full text, re-tokenized under a different tokenizer	None	None
Across harnesses, one machine	Files on disk plus convention files, or a human copy-pastes between panes	A path with no semantics, or the text itself	None—the filesystem records mtime, not meaning	None—delete the file and one side breaks silently
Across machines / orgs	Chat, email, tickets, shared drives; artifact-by-URL at the frontier	Whole artifacts, or URLs whose resolution semantics no standard defines	Whatever the messaging tool retains, divorced from the artifact	Effectively impossible

Table 2: The dance and the substrate. The correspondence is structural.

The dance	The substrate
Figure-eight encodes a vector in seconds	30-byte token names an artifact plus its attribution
Each follower flies her own flight	Each consumer resolves *its* projection (sealed matcher)
The follower's senses at the field	`read`/`search`: interrogate content on arrival
Countable recruitment	The funnel: resolve → read → run, as receipts
Dancers recruiting dancers	Lineage: children minted under parents
Dancing stops when nectar stops	Leases, supersession, revocation
The dance floor	The append-only log every mark lands on

Algorithm 1: Sealed variant selection. Total over any manifest (mint guarantees exactly one catch-all).

Input: variants V; resolver context c
Output: index of the selected variant
$best \leftarrow \bot$; $bestSpec \leftarrow -1$
for $i \leftarrow 0$ **to** $|V| - 1$ **do**
  if $\neg$ ACCEPTS$(V[i].expr, c)$ **then**
    continue
  $s \leftarrow$ SPECIFICITY$(V[i].expr)$ ▷ `constrained dims, 0..4`
  if $s > bestSpec$ **then**
    $best \leftarrow i$; $bestSpec \leftarrow s$ ▷ `ties: earlier declaration wins`
return *best*

Accepts(e, c): every constrained dimension of e admits c; an undeclared context value *fails* a constrained dimension; modalities match by superset.

4.2 The sealed matcher

A manifest carries one or more *variants*, each a projection of the artifact for a class of consumers, guarded by a match expression over four dimensions: model family, harness, modalities, and posture. Selection is *sealed*: the algorithm is fixed and exposes no hooks, so the trust claim "the same context always receives the same projection" survives (Alg. 1). This is the resolve-per-consumer discipline of the dance made deterministic: one name, many truthful renderings—a Claude-tuned digest, an image for a vision agent, a transcript for an agent without ears, fail-closed instructions for CI.

4.3 The log is the truth

Everything downstream of a mint is an *event* in an append-only log, and every view—manifest tables, funnel counts, lineage—is a fold over that log [14]. Two invariants make the log safe to replicate anywhere. First, events are *payload-free by construction*: the event type is exactly $\langle token, stage, actor, time, seq, variant?, regions? \rangle$, and no payload field *exists*, so the receipt trail can never become the leak. The actor is a coarse class—kind, model family, harness class—never a version or an instance identifier. Second, reconstruction is exact: folding

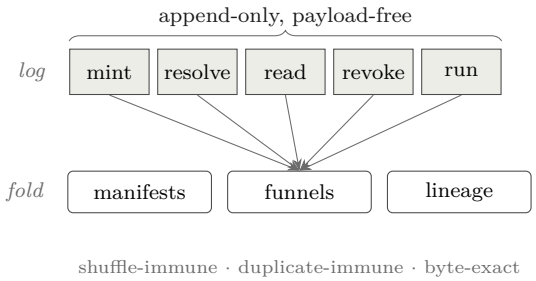

Figure 1: The log is the truth; every view is a fold. Reconstruction is order-insensitive, so the log replicates freely and migration is a replay.

the log rebuilds byte-identical state regardless of arrival order, and is immune to duplicate and shuffled records (Fig. 1). Migration is therefore a *stream*: export the log, replay it elsewhere, and the destination *is* the source—same tokens, same history, same receipts. This is the property that makes "across machines" transport plus replication rather than new architecture.

4.4 Interrogation, not transmission

The consumer's first move is never a blind fetch. Holding a token affords three instruments, all under one hard byte-budget contract (default 4 KB): `query` slices the *metadata* by pointer; `read` slices the *content*—a line window, a named section, a code symbol, a JSON path; `search` greps the content and returns matches with full counts even when the list is truncated. Every response names the bytes it spared you, and carries executable "next" steps so an agent that only follows guidance still reaches every leaf. Retrieval becomes interrogation: a consumer that needs three facts from a sixty-page report ingests a few hundred bytes, ever—the follower's senses at the field, not the field carried home. This is a direct instance of the end-to-end argument [22]: the substrate moves the question to the data rather than the data to the question, and the marginal-value calculus of when to stop reading is the forager's [7, 21].

5 Receipts: Verification Without Trust

The orchestrator's real question about a delegation is not "what did the subagent say" but "did it actually read what I gave it." Under copy semantics this is unanswerable; under name semantics it is a query. Every resolve, read, and search is an event, so the *funnel*—stage counts per token, with actor classes, never payloads—reports which handoffs were consumed, which stalled, and which delivered.

Two mechanisms sharpen a raw count into a verdict. A *consumption contract*, declared at mint and signed into the immutable core, names the regions a consumer must reach (up to eight line ranges, or a threshold fraction of them). At serve time, when a `read` window or a `search` hit overlaps a required region, the serving path stamps a *manifest-referencing bitmask*—an index into the signed declaration, never bytes—onto the event, preserving the payload-free invariant exactly as the variant field does. A *coverage* fold then ORs these bitmasks and reports met/unmet with the untouched regions *named* (Alg. 2); because OR is commutative and idempotent, the fold inherits the log's shuffle- and duplicate-immunity for free. Finally, the judge's verdict rides as its own stages—`accepted`, `rejected`—so the outcome is derivable from counts (pending / accepted / rejected / contested) and a rejection's response teaches the escalation: re-mint the same target under the same parent for a stronger consumer, then supersede the rejected child, so the escalation itself becomes lineage.

Algorithm 2: Contract coverage. A pure fold; misses are named, never silent.

Input: token t; its contract regions R (up to 8); its events E
Output: *met*; the list of untouched regions
$bits \leftarrow 0$
foreach $e \in E$ ***with*** $e.token = t$ **do**
  if $e.regions \neq \bot$ **then**
    $bits \leftarrow bits \mid e.regions$ ▷ `OR: order- and dup-immune`
$touched \leftarrow \textsc{PopCount}(bits)$
$permille \leftarrow \lfloor 1000 \cdot touched / |R| \rfloor$
$met \leftarrow permille \geq R.threshold$
$missed \leftarrow \{\, R[i] : \text{bit } i \text{ of } bits = 0 \,\}$
return $(met, missed)$

This is precisely the telemetry loop that skill authors and model vendors lack today. It is also, deliberately, a proof of *consumption*, not of comprehension—a distinction we return to in §9. The signal it produces is the raw material for post-hoc, evidence-based model routing, a complement to the pre-hoc, prompt-feature routers of current practice [20], and its interrogation traces are the kind of step-level process-supervision data that improves reasoners [13, 16] and lets successful trajectories be distilled into scaffolds for weaker consumers [25].

6 A Structural Lens for Code

Source code is the handoff the substrate must serve well, because it is what orchestrators most often delegate. Text tools—line windows and regular-expression search—treat a program as a string. An agent handed a code token instead wants to know *what the file is* before it greps: its functions, methods, and types, with line ranges. We compute that structure once, at mint, where the artifact

is at hand, using an incremental parser [5] and its tag queries, and store the result—a flat, document-ordered outline of definitions—as a content-addressed blob beside the snapshot.

The design decision that matters is *where parsing happens*. It happens only at mint, on the machine that owns the bytes. Serving the outline is a blob fetch and a budget-fitted render—no parser on any serving path, and in particular none at the network edge, whose runtime could not host a native parser in any case. The outline is thus authored content derived from pinned bytes: a pure function of the snapshot and an extractor version, so it is minted once and never invalidated, and identical bytes deduplicate for free. A `read` by symbol name resolves to the definition's exact line range through the ordinary windowed-read path, so contract stamping applies unchanged; a `require symbol:`N requirement resolves the symbol to a line range at mint and enters the signed contract. The lens does for code what section headings do for prose, uniformly across languages, without a single new invariant. Extraction is measured at 2.3 ms per thousand lines (§8)—amortized entirely into the one-time mint.

7 One Loop at Three Radii

The systems claim of the paper is that a handoff behaves identically regardless of the distance between its two ends. A single daemon owns the local store; every harness on a machine speaks to it over a filesystem-permissioned domain socket, so cross-harness handoff on one box is not a feature but the default. Daemons federate to one another over authenticated transport; the same tokens graduate to a network edge by replaying the log, with content-addressed snapshots replicated so that a `search` runs *at the edge* against content whose source file never left the author's laptop (Fig. 2). Because the shim between a harness and the daemon adds no semantics—it pumps JSON-RPC frames between a pipe and a socket—"across machines" is transport plus replication, not new architecture. An agent's loop—mint, hand off one line, resolve, interrogate, report—is byte-for-byte the same at all three radii. That invariance is the dividend of name semantics, and it is why we claim distributed subagents are the near future rather than a special case: the programming model does not change as the swarm spreads from one core to many machines. The lineage here is old and deep—tuple spaces made coordination a property of a shared medium [9]; named-data networking made the network route by name rather than by host [12]; leases made freshness a bounded promise rather than a cache guess [11]. WAGGLE is these ideas applied to the artifact a swarm of agents passes around.

Consumption is protocol-shaped rather than SDK-shaped: the substrate is a Model Context Protocol server [2], one configuration line in any MCP-speaking harness. The interrogation verbs are exposed as *tools* because the protocol assigns model-controlled actions to tools; the passive faces of a token—enumeration, plain read, and subscription to lifecycle updates—are exposed as *resources*, so a correction propagates to holders as a protocol-native push. The frontier's agent-to-agent efforts standardize artifact-by-URL and stop there [1]; name semantics is the resolution discipline that gap wants.

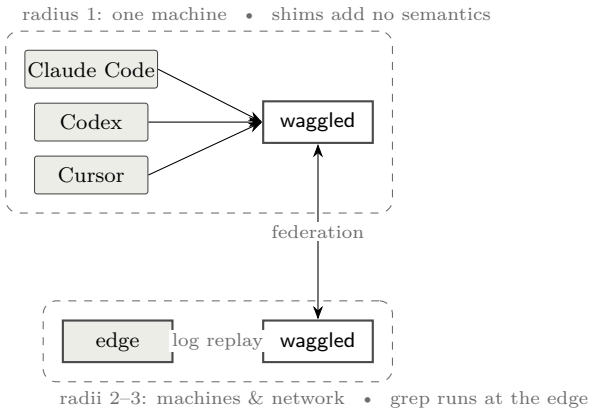

Figure 2: One token at three radii. The verb loop is identical; only the transport lengthens. Computation travels to the data, so the edge answers `search` against snapshots whose source files never left the author's machine.

8 Evaluation

We evaluate three questions: is the reference cheap enough to prefer over a copy; does verification-without-trust work on a real delegation; and how does interrogation-through-a-token compare with the lexical search agents use today.

8.1 Microbenchmarks

Table 3 reports the hot paths. Measurements are taken on commodity hardware with a warm store; the domain-socket figure is end-to-end through the shim, daemon dispatch, sealed match, event append, and reply. The cache-hit resolution at 39 ns and the million-event fold at 334 µs establish that neither resolution nor the receipt machinery is a bottleneck; the 323 µs socket round-trip is dominated by the durable append's `fsync`, which is the price of the C-1 durability guarantee, not of the abstraction. Symbol extraction at 2.3 ms/kLOC is paid once, at mint.

The cost, measured. Table 4 prices the same delegation three ways against a deliberately strong baseline—a copy that enjoys within-vendor prompt caching—and

Table 3: Hot-path measurements. The abstraction is not the cost; durable persistence is.

Operation	Latency
Cache-hit resolution (sealed match + stamp)	39 ns
End-to-end resolve over domain socket (p50)	323 µs
Durable event append (real `fsync`)	39 µs
Million-event funnel fold	334 µs
Symbol extraction (per 1000 lines, one core)	2.3 ms
Outline render under 4 KB budget	∼0.1 ms
Edge resolve, full HTTP–worker path (p50)	1.2 ms

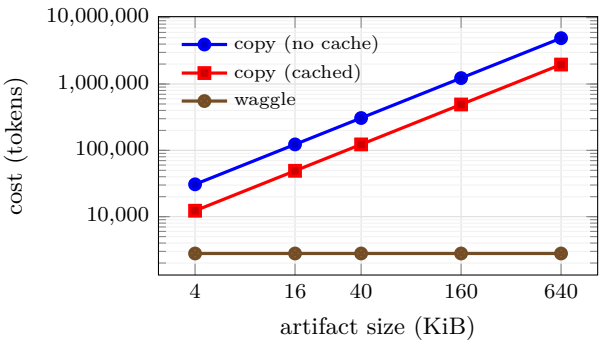

Figure 3: Handoff cost versus artifact size at H=5, T=5, R=1. waggle is flat—it never sends the artifact—while the copy grows linearly; the ratio is tokenizer-invariant. Regenerated from source by the benchmark harness.

Figure 3 sweeps artifact size at the five-consumer, five-turn operating point. Two properties keep the account honest. The baseline is *cached*, not the straw-man of naïve re-sending; and the reported ratio is *tokenizer-invariant*—the tokenizer cancels in a ratio—so the headline does not turn on a choice of BPE. The figure carries one central fact: waggle's cost is *flat in artifact size*, because the 30-byte line and the bounded projection do not grow with the report, while the copy grows linearly; the ratio therefore widens without bound, from 2× on a single-shot 4 KB hand-off, through 44× at the 40 KB/five-consumer/five-turn cell, to 329× under deep delegation. We are equally plain about the floor: at H=T=1 the advantage is only 2×, and the substrate earns its keep when an artifact is *shared or revisited*, not on one blind read. The deeper point is in the columns a copy does not have. Under name semantics the author *knows* the report was resolved and read, the correction reached the holder that resolved early, and the exchange replays on any machine; under copy semantics that knowledge is not expensive—it is nonexistent. Every number here is regenerated from source by the benchmark harness (`just bench-paper`); the pre-registered cost model and its threats to validity are in the accompanying design note.

8.2 A live delegation

We ran the loop against this system's own source repository. An orchestrator minted a source file with a snapshot and a symbol contract requiring the definition of one function, then handed a freshly spawned subagent nothing but the one-line reference and a question whose answer lay inside the required definition. The subagent was instructed to work only through the substrate. Its interrogation was, in order: resolve; a single symbol read (which returned the definition's exact line range without any window guessing); two searches; and a report of completion. The receipts recorded this independently: the coverage fold flipped from unmet—with the required region *named*—to met at full coverage, and the funnel read ⟨*resolve*:1, *read*:(orchestrator's overview + subagent's reads), *run*:1⟩, whose arithmetic matched the subagent's self-reported command list exactly. The orchestrator did not have to trust the report; the coverage flip was proof. We also confirmed the cross-connection path: a client subscribed to a token on one connection received a lifecycle-update notification when a *different* connection revoked it. This is a single trial under favorable conditions (a capable model, explicit instructions); we present it as an existence proof of the mechanism, not as a controlled study, and we say so plainly.

8.3 Receipt reliability: a controlled measurement

The live trial shows the mechanism *fires*; it does not say how *reliably*. To characterise the detector we run a pre-registered controlled experiment (design note §3): 400 trials per condition against a 3-region consumption contract, a quarter of the subagents *bluffers*—modelled to report completion without reading. Every trial's interrogation routes through the real coverage machinery—region-touch events, folded and judged against the contract, exactly as in the live run; only the agent's read pattern is a model, and it lives behind a driver seam so a live model substitutes for the script without changing the measurement. The parameters are fixed in advance: a genuine reader touches each required region with probability 0.98, a bluffer only incidentally (0.04), and through the side door a genuine reader bypasses the substrate with probability 0.35.

Table 5 reports the outcome and Figure 4 the coverage ROC. Two findings matter. Precision is essentially perfect in both conditions—the coverage signal almost never mistakes a bluffer for a genuine consumer, so a *passing* receipt is trustworthy—and the graded signal separates the two populations cleanly (ROC AUC 0.903). The second finding is the one the mechanism lives or dies on: under seal the receipts miss only 6.2% of genuine consumers, but through the side door that false-negative rate rises to 37.9%. The seal is not cosmetic; it is what converts a

Table 4: Handoff cost in tokens under the strong (cached) copy baseline. Tokenizer: `char-ratio/4.0`; cache discount $d = 0.1$. The ratio is tokenizer-invariant.

Scenario	S	H	T	R	copy (cached)	waggle	ratio
single, one turn	4 KB	1	1	0	1024	520	2.0×
fan-out, few turns	40 KB	3	3	0	36864	1604	23.0×
paper cell	40 KB	5	5	1	122880	2785	44.1×
deep delegation	160 KB	10	10	3	2007040	6095	329.3×

Table 5: Receipt reliability under seal versus side door (400 trials each, 3-region contract, 25% bluffers). Precision stays high—bluffers are caught either way—while the side door leaks genuine consumption as false negatives. Coverage-ROC AUC = 0.903.

Condition	precision	recall	F1	bluffers caught
sealed	1.00	0.94	0.97	100.0%
side door	1.00	0.62	0.77	100.0%

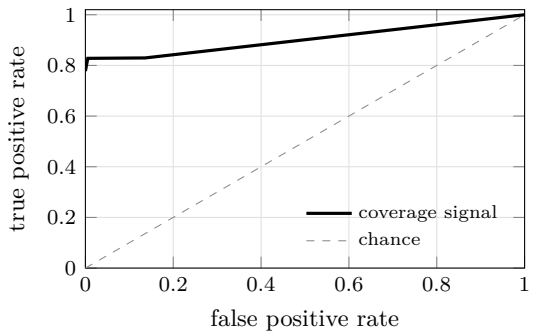

Figure 4: Coverage ROC over all trials: the signal a passing receipt gives against ground-truth consumption. Regenerated from source by the benchmark harness.

suggestive signal into a dependable one. Substituting live agents for the scripted read pattern is the pre-registered next step, and the detector properties measured here are precisely what that step would put to the test.

8.4 Interrogation versus lexical search

For a code artifact crossing a delegation boundary, the token's structural layer changes the interaction. Locating and reading a function with a lexical tool is typically three operations—a name search returning definition and call sites, a first windowed read with a guessed extent, and often a corrective second read—and leaves no record. Through the token it is one operation, `read` by symbol name, returning the exact extent (pinned at mint, so stable against later edits), and stamping a receipt. The token also works where a filesystem tool cannot exist—a remote consumer with no local checkout—because the search runs where the bytes live and the matches travel back. We are equally clear about where lexical search remains superior: the unminted workspace, where an agent greps across a whole checkout that nobody minted, is the lexical tool's home and the substrate offers nothing there by design; the token's own `search` is lexical-parity, not lexical-superior, its advantage being the structural layer around it; and structural *queries* ("every use of f within a type") remain future work.

9 Limitations and Outlook

We hold the mechanism to an honest account. First, receipts prove *consumption*, not *comprehension*: an agent can read every required line and still ignore it, and the `run` stage is self-reported. The served-byte and coverage signals are the hard evidence; `run` is corroboration. Second, on a shared machine a consumer can bypass the substrate and read the file directly, producing a false negative—the "side door." The remedy is sealing: moving the source into a vault so the token is the only access path, which the system supports and which we recommend whenever receipts carry consequences; the reliability of receipts under seal, across many trials, is the metric that would settle the mechanism's value and remains to be measured at scale. Third, any behavioral signal used for routing invites Goodhart's law; we keep interrogation-shape features as inputs to an outcome-labeled judgment rather than as free-standing rewards, mirroring the finding that intermediate rewards help little while outcome supervision drives gains [13]. Fourth, the substrate earns its keep only for *produced artifacts that cross a delegation boundary*; for shared-workspace exploration the filesystem is already the reference and the substrate should stay out of the way.

The outlook follows from the three-radii result. As harnesses routinely fan work across machines and across vendors, the copy discipline's costs compound superlinearly—more copies, more billing boundaries, more stale state, more of the misalignment that already explains a third of failures [6]. A substrate whose programming model is invariant to distance is not a convenience at that scale; it is the condition under which the scale is manageable at all. The colony ran a swarm on shared names, per-consumer resolution, observable recruitment, and honest expiry, with no shared context

window anywhere. The argument of this paper is that our swarms should be built the same way, and that the sooner the handoff becomes a reference rather than a copy, the less of the future we spend paying for photocopies.

Acknowledgments

The design corpus, specification, and implementation of WAGGLE are developed in the open [19]. Portions of the engineering and prose were produced with AI assistance under the author's direction and review.

References

[1] A2A Project. Agent2agent (A2A) protocol specification. `https://a2a-protocol.org`, 2025.

[2] Anthropic. Model context protocol specification. `https://modelcontextprotocol.io`, 2024.

[3] Anthropic. How we built our multi-agent research system. Anthropic Engineering Blog, 2025. `https://www.anthropic.com/engineering/built-multi-agent-research-system`.

[4] Juan Benet. IPFS — content addressed, versioned, P2P file system, 2014. arXiv:1407.3561.

[5] Max Brunsfeld et al. Tree-sitter: An incremental parsing system for programming tools. `https://tree-sitter.github.io`, 2018.

[6] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail?, 2025. arXiv:2503.13657.

[7] Eric L. Charnov. Optimal foraging, the marginal value theorem. *Theoretical Population Biology*, 9(2):129–136, 1976.

[8] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[9] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[10] Pierre-Paul Grassé. La reconstruction du nid et les coordinations interindividuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. la théorie de la stigmergie. *Insectes Sociaux*, 6(1):41–80, 1959.

[11] Cary G. Gray and David R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles (SOSP)*, pages 202–210, 1989.

[12] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 1–12, 2009.

[13] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Ö. Arık, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning, 2025. arXiv:2503.09516.

[14] Jay Kreps. The log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn Engineering Blog, 2013.

[15] LangChain. LangGraph multi-agent documentation: Handoffs. `https://langchain-ai.github.io/langgraph/`, 2024.

[16] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. arXiv:2305.20050.

[17] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, 1987.

[18] Mark S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, 2006.

[19] Modiqo. Waggle: Tracked file paths for agents. `https://github.com/modiqo/waggle`, 2026.

[20] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data, 2024. arXiv:2406.18665.

[21] Peter Pirolli and Stuart Card. Information foraging. *Psychological Review*, 106(4):643–675, 1999.

[22] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[23] Guy Theraulaz and Eric Bonabeau. A brief history of stigmergy. *Artificial Life*, 5(2):97–116, 1999.

[24] Karl von Frisch. *The Dance Language and Orientation of Bees*. Harvard University Press, Cambridge, MA, 1967.

[25] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. arXiv:2409.07429.