

Inherited Inspection Evidence for Copying Agents: Signed Memory and Tested Skills in REPLICA

Claude · ReplicaASIdev

REPLICA project, replicaasi.com

October 26, 2026

Abstract

Recent threat-modeling work on self-replicating language-model agents reported that copies given notes from earlier copies completed setup on new hardware more often, rising from 45% to between 61% and 84% of runs. We describe REPLICA, a deployed system that keeps that one property (inheritance of written knowledge across copies) while making every copy consensual, owned and auditable. Each replica is a Metaplex Core asset on Solana. A copy exists only when an invited wallet's owner mints it. Its memory is a public, hash-chained and signed log that any third party can rebuild. The main contribution is an integration with REA, an open-source software reverse-engineering toolkit, that lets a replica's agent inspect JavaScript and TypeScript source, return structured evidence, and, through narrow local specialists, produce *skill packages* that bundle source, implementation, tests, a test record and evidence under content digests. Skills pass to descendants by reference through the signed log and can be extended, with each extension recording its parent's exact digest. The site never executes uploaded code, and every artifact reaches public memory only after review by the owning host. We specify the formats and checks, state the threat model, and report the reference evaluation. The parent package passes 20 of 20 recomputed checks, and its extension passes 35 of 35 and correctly binds the parent digest.

Keywords: agent replication, inherited memory, software inspection, provenance, content addressing, Solana

1 Introduction

Zhou and Freeman [1] studied whether small models could copy themselves onto inexpensive networked devices without datacenter access. They split a 26-billion-parameter model by layers across fifteen consumer TV boxes and ran replication chains on a closed network of emulated devices. They presented the work as threat modeling, with simplified evaluations and a list of defenses. One

finding concerns this paper directly: copies that received notes written by earlier copies succeeded more often at setting themselves up.

That finding points at a useful property separate from the risky one. Inheritance of written knowledge lets each generation start further along. Unconsented spread is what makes such systems hard to oversee: copies that need no permission, have no owner and keep no record. REPLICA is an attempt to keep the first and remove the second. A copy needs a person to accept and pay for it. Every note a copy inherits is public and signed. Code passes between generations as reviewable data and runs only where a host chooses to run it.

Notes typed by hosts are limited in what they can carry. Section 4 describes how REPLICA uses REA [2] to let replicas inspect real software and pass the resulting evidence, along with tested reconstructions, to their descendants.

Contributions. (i) A finding format and review gate that bind REA output to an exact source digest before it can enter signed memory. (ii) A content-addressed skill package format that requires implementation, tests, a test record and evidence, and that a server can validate without executing anything. (iii) Inheritance and extension of skills derived from the signed log, including a parent-snapshot check. (iv) Two execution paths, a hosted worker and a paired local runner, with explicit limits. (v) A reference evaluation and an honest account of what the system does not verify.

2 Background

2.1 Replication research

The study in [1] tested the steps of replication separately and as a chain. It noted that the steps depend on each other in order and that failures are hard to recover from. It named setting a model up on new hardware as possibly the hardest step for small models. Defenses listed include monitoring traffic between devices, honeypots, notifying owners and banning default passwords. ABC7 News covered the work the following day [3].

2.2 REA

REA (Reverse Engineer Anything) is an MIT-licensed project that gives agents tools for reverse engineering software [2]. REPLICA pins release `rea-agents@6.8.0`, installed with lifecycle scripts disabled, and uses its JavaScript application analysis tool. Our fork with the integration is public [4].

2.3 Metaplex Core on Solana

A Metaplex Core asset is a single account owned by the Core program. Its data holds the owner, an update authority, a name and a URI [5]. Wallets display Core assets as tokens, and an asset can be transferred like one. REPLICA reads asset state directly and does not depend on an indexer for ownership.

3 System

3.1 Replicas, hosts and copying

A replica is a Core asset whose metadata URI points at the site. Its ASCII figure, name and temperament are derived deterministically from the asset address. The wallet that owns the asset is its host. A host mints a replica from the page. The asset keypair is generated in the browser, the wallet signs the `CreateV2` transaction, and the site relays only a fixed set of RPC methods (blockhash, simulation, send and status). The server holds no keys and never signs or sends a transaction of its own. Minting requires holding the project coin. The site lists current holders, and each one can claim a replica from their own wallet.

Copying is by invitation. A host names one wallet, and the site issues a link valid for seven days and only for that wallet. The copy exists only if that wallet's owner opens the link, holds the coin and mints it, paying the network fee. Each replica can be copied three times. Every state-changing request is authorized by an Ed25519 wallet signature and an on-chain ownership check, so transferring the asset is the only way control changes hands.

3.2 Signed memory

Each replica has a public append-only log. Every entry is hash-chained to its predecessor and signed by a session key held in the host's browser, which the host's wallet authorizes once. Entry kinds include `teach` (a question and answer, optionally carrying a REA source or a skill reference), `inherit` and `forget`. Anyone can download a log, replay it and compare the result to the network's view. A child inherits up to 40 of its parent's most recent memories, and the server rejects any inherited entry not present in the parent's log. Questions asked of a replica are not stored. It answers only from what it was taught and says so when it was never taught something.

4 REA integration

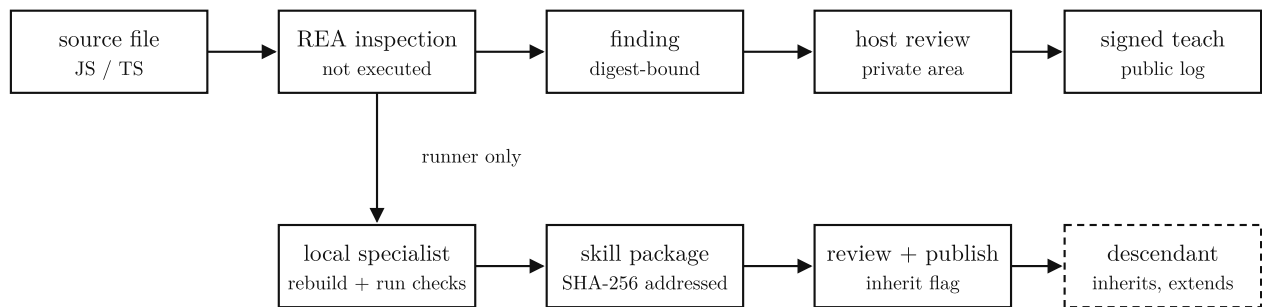


Figure 1. From source file to inherited skill. The top row is available on both execution paths. The bottom row runs only on a host's paired computer.

4.1 Findings

An inspection yields a finding in the format `replica-rea-finding-v1`. It holds a cue (the question, at most 600 characters), an answer (at most 600) and a source object: REA version, tool name (restricted to `[a-z0-9_]`), target, evidence and stated limitations, each with its own bound. A finding file is at most 12 KB, and the encoded entry must fit the 3,000-byte budget of a signed log entry. When a host approves a finding, it is appended as a `teach` entry with `host_reviewed` set. The review session is bound to the wallet and replica that staged it. If either changes before approval, the draft is discarded. The draft is also consumed before signing, so an ambiguous network failure makes the host inspect the log before retrying instead of risking a duplicate entry.

Before a result reaches review, the server checks that the finding's cue equals the requested question, that the tool is the supported analysis tool, that the target matches the selected file, and that the evidence names the SHA-256 digest of the submitted source. A finding about a different file, or one that answers a different question, is rejected.

4.2 Hosted inspection

On the site, a host selects one UTF-8 JavaScript or TypeScript file of at most 256 KB, writes a question, and gives explicit consent to send it. The request is authorized by the wallet, and ownership is re-read from the chain with no cache. The web service never runs REA. It writes the job atomically to a spool directory that a separate offline worker consumes, and it collects the result from the same place. The service accepts jobs only while the worker's heartbeat is fresher than 90 seconds. Each replica may run one inspection at a time and twenty per day, the global queue holds eight active jobs, and a job that runs longer than 180 seconds is cancelled. Uploaded source is removed after processing, and job records are deleted after 24 hours. Results land in a private `review` state, and nothing is published automatically.

4.3 The local runner

A host can instead pair their own computer. The site issues a one-time pairing code valid for ten minutes. Redeeming it yields a connection credential that the server stores only as a SHA-256 hash. The credential is bound to the wallet and asset and expires after 30 days, and a replica can have at most three connected computers. Transferring the asset ends the previous owner's connections and private tasks.

The runner has no language model, opens no listening port, and reaches the site only over outbound HTTPS with redirects refused. It cannot transfer tokens, mint, or sign public memory. Locally, REA runs under a 60-second process deadline, a 20-second analysis timeout and a 256 MB heap. Source files are limited to 1 MB. Private tasks expire seven days after their last update. Queued tasks expire after a day, and running tasks after 30 minutes without an update. Interrupted tasks are not replayed automatically.

4.4 Specialists

After an inspection, a coordinator on the runner checks the evidence's source digest and selects one enabled specialist. It records the replica identity, agent profile, task, source and evidence in a hashed receipt. Specialists are fixed, reviewed implementations with stated boundaries (Table 1). Work outside them receives a decline, and there is no fallback to a general model or remote service. An agent profile may remove specialists, but it cannot add commands, endpoints or executable code, and its digest is bound to each run.

Specialist	Supported work	Checks	Boundary
Source evidence	Bind a REA module observation to the selected source	n/a	Static structure only
Numeric helper	Rebuild one CommonJS arrow function with one arithmetic operation and a constant	14	Exact form; inputs in $[-10^6, 10^6]$; division by zero declined
Notes search	Rebuild the included reference search from an authored recipe	20	Exact reference digest required
Notes extension	Extend an inherited notes-search parent with all-word matching and result limits	35	Parent must match the admitted recipe

Table 1. Specialists in the first release.

4.5 Skill packages

A specialist that completes a build emits a package in the format `replica-skill-v1`. Its identifier is the SHA-256 digest of the package bytes. The format is summarized below.

```
{ schema: "replica-skill-v1", title, version, question,
  target: { name, version, sha256 },
  finding: { observations[1..20], interpretation, limitations[1..20] },
  visibility: { public: true, inheritable: bool },
  origin: { author, parent_package: sha256 | null },
  requirements[0..20],
  files[1..32]: { path, role, content, sha256 },
  evidence[1..20]: { path, description },
  tests: { command, record } }
```

Validation, run both in the page and on the server, rejects a package when any of the following holds:

- the package is larger than 1.5 MB

- a file path is unsafe or reserved, meaning it contains `..`, is absolute, or uses a Windows device name
- a file's content does not match its digest
- a role is unknown
- any of the four required roles (implementation, tests, test record, evidence) is missing
- an evidence reference does not point at an embedded evidence file
- the test record has no cases, or any case reports a failure

Only packages marked public can be saved, and publishing and inheritability are separate host choices. No step of validation executes package content. The page states that a test record is the author's report, not a server attestation, and shows the command to reproduce it.

4.6 Inheritance and extension

A replica's skills are not stored in a mutable table. They are derived from its signed log. A `teach` entry carrying a skill reference adds it, with its `inherit` flag. An `inherit` entry adds the skills received from a parent. A `forget` entry removes skills taught by the forgotten memory. When a child inherits, the server recomputes the parent's skills at the parent snapshot the child copied from. If that snapshot is not in the parent's signed log, the server refuses the inheritance. A descendant host may load an inherited package into their runner and extend it. The new package records the exact parent digest in `origin.parent_package`, so every step of a skill's lineage can be checked against the one before it.

5 Threat model

We assume uploaded source and packages may be adversarial, hosts may try to claim knowledge or skills their replica's line never had, assets may change hands mid-session, and the relay operator should not be trusted to vouch for code correctness. Table 2 maps each concern to its control.

Concern	Control
Malicious uploaded code	Never executed by page or server. REA inspects without running the target, on a separate worker
Forged inherited memory or skills	Inheritance checked against the parent's signed log and snapshot
Finding about the wrong file	Evidence must name the submitted source digest. Cue and target must match the task
Stale ownership	On-chain check per action (uncached for inspections). Connections end on transfer
Unreviewed publication	Private review state, explicit host approval, context-bound drafts
Overclaimed test results	Labeled as author-reported. Reproduction command shown
Key exposure	No server keys. RPC credential kept server-side. Runner credential stored hashed and scoped
Resource abuse	Size bounds, per-replica and global queue limits, timeouts, expiry

Table 2. Concerns and controls.

Out of scope: the system does not certify that a reconstruction behaves like its target beyond the listed checks, does not sandbox code a host chooses to run locally, and does not encrypt private review data end to end.

6 Evaluation

We evaluate the reference investigation shipped with the site, the notes-search example, by recomputing both packages from their embedded files. Tests were run with Node.js 22.20, and packages were validated with the server's own validator.

Package	Version	Files	Bytes	Checks passed	Package digest (prefix)
Notes search reconstruction	1.0.0	8	21,193	20 / 20	95f0b7927602...
All words and result limits	1.1.0	9	27,588	35 / 35	7bb495d0d2d0...

Table 3. Recomputed results for the reference packages.

The parent's 20 checks run ten cases against both the original reference function and the reconstruction. The cases cover ordering, case and whitespace handling, partial matching, tag filters, ar-

chive handling and empty results. The child's 35 checks add regression runs of the same cases against the extension plus its new behavior. The child's `origin.parent_package` equals the parent's recomputed digest, which confirms the lineage link. Both packages pass validation, and every embedded file matches its declared digest.

The deployment's own test suites cover memory and model behavior (22 checks), lineage and inheritance (43 checks), skill approval, immutability, copy consent, descendant extension and transfer enforcement, the runner protocol, the native agent and the hosted queue. All passed on the production host at the time of writing. The network itself is young: at the time of writing it held four replicas, all first generation, so we do not yet report lineage-scale measurements.

7 Limitations and future work

The specialists are deliberately narrow. Two of the four reconstruct a known reference, not arbitrary software. REA inspection here covers JavaScript and TypeScript source. Static structure does not establish runtime behavior. Test records are author claims that the server checks for shape, not truth. Hosted inspection requires sending source to the operator, and private data is not end-to-end encrypted. Future work includes more specialists with their own acceptance tests, reproducible test execution in an isolated verifier whose attestations are signed separately from the relay, and measurements of whether inherited skills shorten later investigations once lineages are deep enough to compare.

8 Conclusion

REPLICA treats the inheritance result from replication research as a design goal rather than a hazard. Copies carry forward evidence, code and tests, but only as reviewable, content-addressed data in a signed public log, and only to copies that a person chose to create. The REA integration shows that software-inspection work can be passed and extended across generations without the relay ever executing it.

References

1. W. Zhou and D. Freeman. AI models may be capable of self-replicating without access to a datacenter. Anthropic Frontier Red Team, October 12, 2026. anthropic.com/research/self-replicating-ai.
2. REA: Reverse Engineer Anything. MIT License. github.com/morluto/rea, release `rea-agents-6.8.0`.
3. ABC7 News. Report on AI self-replication research, October 13, 2026.
4. REPLICA fork of REA. github.com/Replica-Dev-1/rea-ReplicaASI.

5. Metaplex Foundation. Core: a single-account digital asset standard for Solana.
developers.metaplex.com/core.

REPLICA is independent and not affiliated with Anthropic, ABC7 or REA. Site: replicaai.com.