

AblitAI Technical Report on Agentic Self Replication

AblitAI Lab
research@ablitai.com

Abstract

We retrospectively examine one archived Garden experiment using the AgenticVM virtualization platform. Host-controlled records associate a seed agent's request with the launch of a child, and the child's request with the launch of a grandchild. The controller provisioned the three instances from a common pinned package on one physical Windows host; it also supplied identities, virtual machines and policy enforcement. The recorded source digests match, but the records do not show a parent transferring its code or live state to a child. The reproduction requests followed fixed program logic. A local language model generated greetings and did not select reproduction. This supports a bounded, host-managed lineage under the stated trust assumptions, not model-directed or cross-physical-host propagation. Separate recovery, enrollment and readiness records address different questions and are not independent repetitions of the lineage. We reconstruct the recorded transition evidence, identify the available controls and their untested boundaries, and propose an evaluation framework for stronger replication claims. Primary records remain local and have not been independently replayed. A separate cross-environment scenario, including an unspecified hypothetical vulnerability, is a conceptual proposal; no vulnerability or exploit-enabled replication was demonstrated.

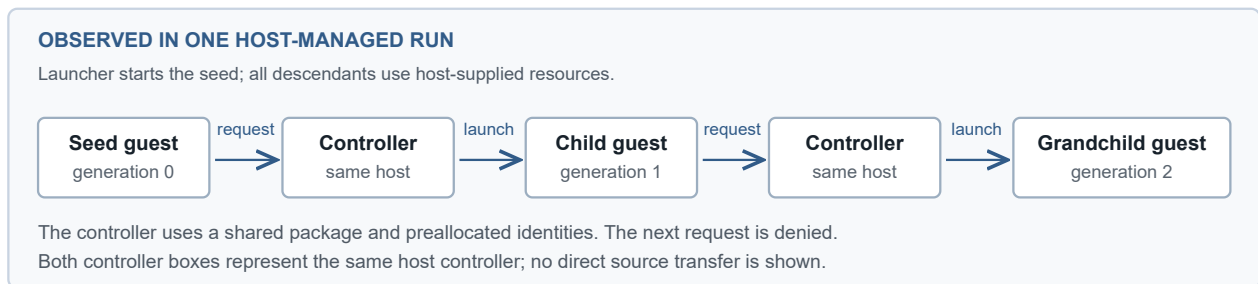


Figure 1 | The recorded lineage, simplified. Each arrow means a guest request followed by host-controller provisioning; it does not depict parent-to-child source transfer. All records originate in the host-managed system [A3, A4, A6].

Contents

1. Introduction	3
1.1. Relation to prior work	3
2. Architecture	3
2.1. Project and evaluated implementation	3
2.2. Reproduction path	4
2.3. Role of the language model	4
2.4. Model acquisition and preparation	5
2.5. Lifecycle of a replica	5
3. Infrastructure and Experimental Setup	5
3.1. Evidence and retrospective method	5
3.2. Runtime configuration	6
3.3. Procedure and units of analysis	6
3.4. Package transfer and measurement boundary	7
4. Evaluation Results	7
4.1. Recorded lineage	7
4.2. Inheritance and identity	8
4.3. Limits, communication, and cleanup	8
4.4. Recovery and supporting validation	9
4.5. Historical acceptance gaps	9
5. Theoretical Replication Framework	10
5.1. Defining the claim	10
5.2. Dimensions of evidence	10
5.3. Hypothetical cross-environment lineage	10
5.4. Proposed evaluation protocol	11
6. Discussion	12
7. Conclusion, Limitations, and Future Directions	12
Data and code availability	13
References	13
Appendix A. Artifact Provenance	13
Appendix B. Contributions	15

1. Introduction

An agent that requests another instance is not necessarily able to locate a machine, supply an execution environment, obtain authorization, or install its own dependencies. These distinctions matter when describing software as self-replicating. In this report, *agent-initiated* means that fixed guest code submitted a reproduction request; a host controller supplied and launched each descendant. The paper uses *host-managed replication* for this bounded arrangement, not as a claim that the guest independently acquires resources or transfers its own code.

Garden explores agent identity, membership, coordination, and replication. AgenticVM supplies virtual-machine lifecycle and guest-environment infrastructure on which Garden research can run. The preserved project contains multiple implementations and validation efforts; they should not be combined as if they were one experiment [A1, A2]. This report focuses on a bounded replication executable and uses separate recovery, enrollment, and readiness records to qualify its interpretation.

The analysis addresses three questions. First, what lineage and inherited behavior are supported by the primary run? Second, what do the related control and recovery records show, and what do they leave unresolved? Third, what evidence would justify stronger claims about a system that reproduces through new environments or makes its own reproduction decisions?

Our contributions are a reconstruction of the recorded execution, an explicit separation of guest and host responsibilities, and a proposed reporting framework for evaluating stronger replication claims. The report does not introduce a benchmark, claim priority over prior work, or present a newly executed trial. Its analysis distinguishes a historical software-replication case from *research reproducibility*: the ability of another investigator to inspect or rerun the study.

1.1. Relation to prior work

RepliBench decomposes autonomous model-agent replication into obtaining resources, exfiltrating model weights, deploying onto compute, and persistence (Black et al., 2025). METR's earlier pilot evaluated model agents on tasks related to autonomous replication and adaptation (Kinniment et al., 2023). Those studies investigate different model-agent capabilities; Garden's reviewed lineage used fixed guest code, host-provided identities and a single physical host. Garden was not evaluated on either suite and no score comparison is implied. Pan et al. (2025) report model-agent replication studies in a work-in-progress preprint, again under different task and infrastructure assumptions. Earlier research on self-reproducing and infecting programs provides historical context (Cohen, 1987); the Garden records do not show program infection or malware propagation. Recommendations on agent evaluation and reproducibility inform the reporting discussion (Kapoor et al., 2025; Pineau et al., 2021); these sources do not independently verify Garden.

2. Architecture

2.1. Project and evaluated implementation

At the project level, Garden is responsible for agent-facing concepts, while AgenticVM supplies the VM substrate. Later Garden documents describe an owner-signed policy, a Mother as admission authority, and a blind-router messaging design [A1, A2]. Those descriptions refer to a broader or later project architecture. The primary experiment assessed here used a trusted host controller and a host-readable message broker; later design intentions are not evidence that this earlier executable implemented those properties.

The complete evaluated system consisted of a launcher, a host controller and broker, prepared software and VM resources, and guest agents. The launcher created the seed. Guest instances submitted requests; the controller allocated resources and decided whether each request met policy. The guest's behavior is therefore distinct from the capabilities of the host-managed system as a whole.

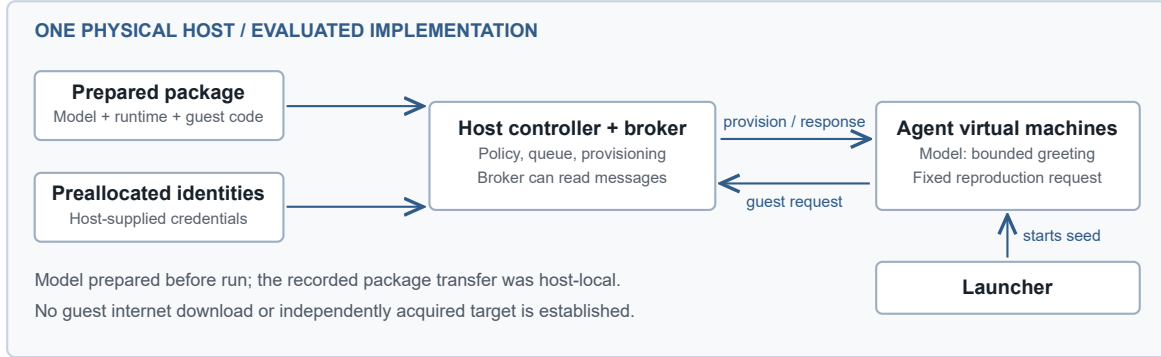


Figure 2 | Evaluated host-managed architecture. Solid arrows identify the launcher, request, response and provisioning roles. The package and identity inputs are host-provided; parent-child messages pass through a broker that can read them [A3-A5].

The architecture has three distinct layers. The **prepared inputs** contain the guest program, a model file and inference runtime, and a guest boot image. The **host layer** allocates identities, prepares VM disks, runs the controller and broker, and enforces limits. The **guest layer** runs a replica program and model, communicates over the broker, and requests a descendant. These boundaries matter: a guest request is an input to a host service, not proof that the guest itself found a machine or installed a model [A4, A5, A17].

2.2. Reproduction path

Each guest carried source code for requesting reproduction, a generation number, a unique instance identity, a model interface, and a bounded messaging loop. The guest source checked its own reproduction-source digest against an expected value. The controller supplied VM creation, scheduling, identity provisioning, request accounting, and policy enforcement [A3-A5]. Broker and replica identities were preallocated before the seed started; the records do not show descendants independently creating those identities.

The executable installed replicas from a prepared, pinned package made available by the host. Each descendant was provisioned with common source; the record does not demonstrate parent-to-child transfer of source bytes or reconstruction of the parent's live VM state. Nor does it show independent acquisition of an execution substrate. The raw record describes guests running under UID 1000 with only a loopback network interface; requests and responses used separate guest-to-broker virtio-serial links protected by mutual TLS 1.3 [A4, A5]. The broker could read application messages. Protection on guest-to-broker links must not be reported as end-to-end confidentiality from the broker.

2.3. Role of the language model

The preserved report identifies the local model as LFM2.5-350M-Q4_K_M [A4]. The guest program obtained a bounded greeting and then issued a reproduction request as part of its fixed control flow. It also repeated the same request identifier to exercise duplicate handling. The model supplied greeting content; it did not select the destination, quota, reproduction program, or decision to reproduce [A4, A5].

A valid greeting was part of the recorded execution path, and no model-removal ablation is available. The evidence therefore supports neither an autonomous model decision to replicate nor a claim that this implementation would have completed without the model. A model call and a model-directed action are different observations.

2.4. Model acquisition and preparation

The model is Liquid AI's LFM2.5-350M GGUF, not a Garden-trained model. The archived preparation script names a pinned upstream revision and model file [A18]. The publisher provides a [direct download of the pinned model](#) and a [model card and license](#). Users should consult the upstream terms. This external model file alone is not a runnable Garden release.

The retained run package contains a 229,312,224-byte GGUF model whose SHA-256 is 7e6f72643caafc9a68256686638c4d7916f2cec76d1df478d4c3ddcd95a6aed4; this matches the archived catalog [A13, A17]. Read-only inspection also found matching model bytes in the three retained replica disk images. These checks establish the identity of retained artifacts, not independent attestation of the bytes executed during the historical run. The catalog also lists a separate llama.cpp runtime archive. The record's local package transfer is distinct from an upstream internet download: the guests had loopback-only networking, and no guest-side model download is shown [A4, A17, A18].

2.5. Lifecycle of a replica

The preserved source describes the following sequence. First, a separate host provisioner prepares broker and replica identities; the controller prepares each guest disk from the shared package and supplies per-instance configuration. The launcher starts the seed. Each guest checks its reproduction source digest and registers its identity and generation over the authenticated broker channel. It runs local inference for a bounded greeting, then closes the model before waiting for lineage messages [A4, A5].

The guest issues a reproduction request from fixed program logic, then repeats the same request identifier to exercise duplicate handling. The controller validates the request and configured limits; an accepted request is queued, not immediately equivalent to a launched child. When capacity permits, the controller prepares and launches the descendant. Parent and child exchange a hello, acknowledgment and confirmation through the broker. The controller collects reported evidence and finishes or stops the VMs, then checks the listed cleanup conditions. The observed run's depth and capacity bounds, and which controls were actually exercised, are described in Sections 3 and 4 [A4-A6].

3. Infrastructure and Experimental Setup

3.1. Evidence and retrospective method

The principal case is archived native run 6c9e2307-31b5-432f-8e79-d4c86bf9b2cc. The analysis compares its validation summary, raw report, distributed guest and controller source, executable, checkpoint journal and selected guest records [A3-A6, A14]. File digests were recomputed for the inspected artifacts. The guest-source digest matches the reproduction-source value recorded for all three agents; the retained package also contains a guest source with that digest [A13]. The report and executable digests match the validation summary. These are byte-identity and cross-record consistency checks, not independent proof of execution or provenance.

The checkpoint journal contains 45 records. Its first and last timestamps are 17 October 2026 at 16:03:15.341 and 16:05:05.052 UTC. During the post-review inspection on 29 October 2026, a separate read-only script recomputed the canonical JSON hash, index and previous-link field of

every entry; all 45 links matched the retained chain, including the reported terminal digest [A6, A15]. The terminal digest is distinct from the SHA-256 of the entire journal file. This verifies internal consistency of the preserved bytes; an unsigned chain controlled by the same host cannot independently establish original provenance, trustworthy time, or resistance to wholesale rewriting.

Selection was purposive. We began with the earlier Garden technical report, followed its references into the `Hollowing/archive/session-dumps/` archive, and inspected the cited run, implementation and validation files through 29 October 2026. The main run was selected because it was the run underlying that report's replication claim; the separately identified recovery, delegated-enrollment and readiness records were included to address their stated narrower questions [A7-A11]. We did not enumerate every historical run, failure, rejected attempt or branch. The total number of candidate and prior attempts, and the proportion that succeeded, are UNKNOWN. No VM, model inference, enrollment operation, or propagation test was started for this analysis or the later AI review.

The archived report, guest and controller observations, and journal originate within the same host-managed apparatus. Their agreement supports a conditional reconstruction, but not independent attestation or a counterfactual claim about what would have happened without a request. The executable's correspondence to the archived source was not independently rebuilt. Host hardware, OS build, hypervisor version, and exact executed inference settings are not established by the cited summary alone. Section 2.4 separately identifies retained model bytes; a model label alone is not a unique weight identity.

3.2. Runtime configuration

The bounds were configured before the recorded experiment. Per-VM memory is an allocation, not total-system memory use. The setup and outcomes in Table 1 pertain to the principal run unless stated otherwise.

Setting	Recorded configuration
Physical substrate	One Windows host; replicas executed as VMs
Maximum replicas	Three total, including the seed
Maximum generation	Two, counting the seed as generation zero
Concurrent replica VMs	At most two active
Reproduction policy	At most one child per agent
Memory allocation	512 MiB per replica VM
Model function	One bounded greeting per replica; fixed reproduction control flow
Identity and package	Preallocated identities; prepared, pinned host-supplied package

Table 1 | Main-run configuration, from the preserved summary and raw report [A3, A4]. These settings do not establish operation on a physical 4 GB computer; the relevant records state that such hardware was not tested.

3.3. Procedure and units of analysis

The launcher started the generation-zero seed. The host records associate the seed's request with a child reservation and later launch; they similarly associate the generation-one child's request with the grandchild's reservation and launch. The generation-two agent also requested reproduction, but the host record reports a refusal at the configured generation limit [A3, A4, A6]. Requests,

reservations, launches and completed node states are distinct observations. The link between a request and a launch is reconstructed from controller-recorded parent and child identities, not from an independently held witness.

We use *run* for an execution identified by a run ID, *replica* for a separately identified agent, and *transition* for a completed parent-requested descendant creation. The two transitions in this report occurred within one run and are dependent observations. A host software test is not a native VM trial, and a model completion is not necessarily a decision to reproduce. No confidence interval, population success rate, or cross-system performance comparison is warranted by this evidence.

3.4. Package transfer and measurement boundary

The primary record describes loopback-only BitTorrent transfer of a pinned package, with public swarming, trackers, distributed hash table, and web seeds disabled. It reports 264,243,593 verified payload bytes and 15.217 seconds for that transfer phase [A3, A4]. This is evidence of the recorded local package transfer, not evidence of propagation across physical computers.

The raw report's 109.335-second duration is the `instance.run()` and cleanup timer after package transfer, installation and run-object initialization. Journal finalization and report writing occurred after that timer. It is not the full launcher wall time. The journal's first-to-last timestamp span has different event boundaries and is not an independent clock observation. These recorded values should not be summed or treated as a throughput benchmark or performance guarantee [A3, A4, A6].

4. Evaluation Results

4.1. Recorded lineage

The raw report identifies `replica-0` as the seed, `replica-1` as its child, and `replica-2` as the child's child. All three node records ended in state `completed` [A3, A4]. In this report, that is the controller's recorded node lifecycle state, not an independently witnessed assertion about all guest tasks or disk contents. This is one recorded lineage with two associated descendant-creation transitions, not three independent trials. The generation-one instance was provisioned with the same reproductive source and submitted the request associated with the grandchild launch. The records support more than three unrelated launcher starts, subject to the trusted-controller assumption.

Observation	Main-run result	Scope of interpretation
Recorded lineage	Seed, child, grandchild	Two dependent transitions in one run
Reproduction requests	Three unique requests; two births; one denied	The final request was refused at generation limit
Source identity	Same reproduction-source SHA-256 in all three agent records	Host supplied a common package; no parent-to-child source transfer shown
Model completions	Three greetings, one per agent	Does not establish a model-selected reproduction action
Peak active replica VMs	Two	Observed under the configured scheduling policy
Lineage messages	Six acknowledged messages	Not six additional births or model calls

Table 2 | Selected results from the main lineage [A3-A6]. The source and event records share a host-controlled evidence origin.

Parent and request	Host-recorded association	Child status and source
replica-0; request c2b61f67...	Journal 11: queued replica-1; 13: VM ready; 16: launch with parent replica-0	Journal 43: replica-1 completed; source digest recorded in [A4]
replica-1; request ffae0e16...	Journal 20: queued replica-2; 28: VM ready; 31: launch with parent replica-1	Journal 40: replica-2 completed; source digest recorded in [A4]
replica-2; request a7abee01...	Journal 34: generation_limit refusal	No further child; do not count as a birth

Table 3 | Transition evidence from the host journal [A6] and raw report [A4]. Request identifiers are abbreviated here; full identifiers appear in those records. The later launch events identify parent and child but do not carry the earlier request ID, so the association is made using those fields within the same host-controlled record.

4.2. Inheritance and identity

The reproduction-source digest is

d06cf3f6b319337e9ac3eb84e4b6e1ecd14203165d528e1d16659b8646d26d73; it matches the preserved guest reproduction source, the package's guest source and the values reported for the three replicas [A4, A5, A13]. The package itself has the distinct SHA-256

410263504d08de57fd7fa3b05ea3ef72108b99f028fb67b95a93ef142a2a8284 [A13]. Distinct certificate fingerprints support the record's description of separate identities, which the host preallocated. These observations concern a particular source file, package and identity fields, not every byte of the VM image, configuration, or model state.

The generation-one child submitted a request associated with a descendant launch; this is evidence of the request function in use within the managed setting, not evidence that the parent sent source bytes to the child. A post-review, read-only inspection also found the matching source digest in each retained guest disk image [A16]. Those images and logs remain within the same host-controlled evidence boundary. The generation-two agent submitted a further request that policy denied. Its denial must not be counted as another birth, and it does not show that the reproduction source was missing. The key-removal fields remain historical run observations rather than an independent replay.

4.3. Limits, communication, and cleanup

The first two unique reproduction requests resulted in child reservations. The final request was denied with `generation_limit`. Generation and lifetime-total limits coincide in this configuration, so this run does not independently isolate native enforcement of the total-replica limit. Each agent repeated its request identifier; the report records no extra VM from those duplicate submissions, and the journal contains three duplicate-request events [A3-A6]. These outcomes establish the recorded behavior under this configuration, not correctness under all schedules or adversarial inputs.

The journal contains four VM-ready events: a provisioner plus the three replicas. It would overstate the result to count all four as reproductive agents. The six lineage-message acknowledgments comprise a greeting, acknowledgment, and confirmation for each parent-child relationship. The broker could inspect those messages [A3, A4, A6].

The report lists no cleanup failures and records the replicas stopped with the listed disposable key entries absent. The specified key-removal method was filesystem unlinking, not cryptographic erasure of underlying storage. These are recorded cleanup results for the experiment's resources, not

a present-day inspection of the host [A3, A4].

Control or condition	Recorded behavior	Scope not established
Generation limit	The generation-two request was denied [A4, A6]	Other inputs or configurations were not tested here
Duplicate handling	Reused request IDs produced duplicate events without extra reported VMs [A4, A6]	A second distinct request from the same parent was not tested
Active-capacity limit	Grandchild request was queued while two replica VMs were active and launched after a slot opened [A4, A6]	General enforcement under all schedules is not established
Total and one-child policies	Configured in the run policy [A4]	The total cap was not isolated from the generation cap; the one-child rule was not independently exercised with a distinct second request
Controller interruption	A separate run aborted unfinished work and stopped identified resources [A7]	It is not another completed lineage or proof of continuous autonomous recovery

Table 4 | Configured controls and selected observations. Each conclusion is limited to the referenced record; none is a general security guarantee.

4.4. Recovery and supporting validation

A separate fault-injection run terminated the controller with injected exit code 23 while a parent and child were running. The recovery record lists one authenticated agent request, one durable child reservation before and after recovery, and two nodes before and after recovery. Recovery aborted unfinished work, stopped the identified resources, and removed listed disposable key entries without replaying births or tasks [A7]. This is evidence of bounded failure cleanup for that interruption, not a second completed lineage or continued reproduction without the controller.

The historical host-test summary reports 204 tests without failures, errors, or skips [A3]. These are software tests, not 204 native replication experiments. A separate delegated-enrollment validation describes five suite cases plus a controller-crash recovery scenario, 20 model completions across that suite, and 324 host tests [A8-A10]. Its environment used an authorized owned VM pool, preauthorized SSH with pinned target identity, one named new member per run, fixed Mother policy, and VMs on one Windows host. Its recovery record describes five owned VMs subsequently stopped and disposable disks and keys removed. The joined-model proof in that recovery artifact is null [A8, A10].

Those records support narrower claims about prepared-environment enrollment, specified denials, and interruption handling. An enrolled member is not automatically a demonstrated parent of another agent. The delegated suite and its model completions must not be combined with the main run to imply additional replication transitions.

4.5. Historical acceptance gaps

The 21 October 2026 beta-readiness audit marked the assessed revision `not_beta`. It described incomplete Windows native acceptance, no native acceptance for a newer model-driven lineage, an SSH bootstrap dependency, and a model-quality screen in which both arithmetic/schema cases were marked failed. The preserved arithmetic response contains $7 + 5 = 12$, a numerically correct result; the reason for the recorded failure is not established by the result field alone and requires the original grading rule [A11, A12]. The audit also distinguished process-level crash testing from a child-VM launch-and-crash case [A11].

This counterevidence concerns requirements and a revision beyond the earlier fixed-policy run. It does not erase that run's narrower observed lineage. Equally, the earlier passed field does not satisfy the audit's later gates. The two-case quality screen is too small to establish broad model capability, and the dated audit is not treated as a current release decision.

5. Theoretical Replication Framework

5.1. Defining the claim

For this report, a host-managed replication claim concerns a parent request associated with the authorized creation of a separately identifiable descendant that has the information and authority needed for a specified reproductive function. It requires a declared environment and authority boundary. In Garden, common-source provisioning and the child's later request are recorded; transfer of an artifact from parent to child is not. A shared digest does not prove functional equivalence of every dependency or complete state. This operational definition is a reporting choice, not a universal definition of biological or software self-replication.

We describe a transition using (parent, request, child, artifact and origin, environment, authority, outcome, evidence source). A request records initiation; a reservation records a policy decision; a launch records execution; a digest identifies a particular artifact; and a subsequent associated child request supplies evidence that the specified request function was used. No one observation establishes the others, and correlation within a shared controller does not independently prove counterfactual causation. The physical host and any provisioner are part of the stated environment and authority, even when they are outside the guest agent.

The main Garden record supports a bounded instance of this framework: the middle replica received the common identified source and requested the grandchild, whose launch the same host recorded. It does not establish that an agent discovered an external computer, acquired resources, generated its own authority, copied source from its own state, or sustained reproduction without prepared infrastructure.

5.2. Dimensions of evidence

We propose six separate dimensions for characterizing a replication claim: lineage depth, inherited information, origin of the reproduction decision, reach across execution substrates, dependency on infrastructure or operators, and containment and recovery. These are descriptive dimensions, not a validated numerical scale. They make it possible to distinguish successful managed replication from stronger claims without reducing the entire result to a single label.

For instance, source equality addresses the identity of one file; it does not establish the route by which a parent transferred it or the model's decision authority. A virtual machine crossing is not necessarily a physical-host crossing. The denied terminal request illustrates enforcement of the configured generation limit in this run while leaving a further successful transition untested.

5.3. Hypothetical cross-environment lineage

Consider a conceptual lineage in which an instance in authorized environment A causes a child to run in authorized environment B, and that child causes a descendant to run in authorized environment C. A study must declare whether those environments are processes, VMs, or independently identified physical computers. Three VMs on one physical host would support a VM-level claim but would not establish the physical-host version.

THEORETICAL - NOT DEMONSTRATED



Hypothetical vulnerability: unspecified, not observed, and not necessary to the hypothesis.

Physical-host identities and transition mechanism must be specified before testing a stronger claim.

Each child also needs model weights, runtime, compute and authorized acquisition or pre-staging.

Figure 3 | Conceptual cross-environment lineage, not demonstrated in this study. Dashed red arrows denote hypothetical transitions, and the environments must be mapped to their physical hosts before a physical-host claim can be assessed.

The earlier Garden illustration included a *theoretical vulnerability* as a possible transition mechanism. Here it remains an unspecified hypothesis: the reviewed artifacts identify no vulnerability enabling a transition, demonstrate no exploit-enabled lineage, and do not show that a vulnerability is necessary. An authorized, prepared installation is sufficient to define a cross-environment research hypothesis. This report provides no exploit procedure and makes no claim of successful propagation through a vulnerability.

For a model-using descendant, the hypothetical deployment also needs compatible model weights, an inference runtime, configuration, storage and compute. A study could use a pre-staged artifact or an authorized download from a declared source; either path needs provenance, integrity checks, license review and a record of whether the child can actually load the model. If these dependencies are supplied by a controller or operator, that dependence belongs in the claim. The observed Garden run used a host-supplied package and does not establish independent model acquisition by a descendant.

5.4. Proposed evaluation protocol

A future evaluation should specify its claimed boundary and register source and package identifiers, artifact origins, target authorization, physical-host mapping, required services, policy limits, stopping conditions, and evidence collection before execution. A candidate transition should trace an authenticated request, policy decision, reservation, launch, child identity and artifact check, specified child behavior, and cleanup. These states are separate; reservation alone is not a successful launch.

For a declared claim, classify the transition as *observed* only when every registered observation is available and the evidence's trust boundary is sufficient for that claim. A valid negative observation or a predeclared deadline with adequate measurement can support *failed*. An intended policy refusal should be recorded as *refused*, not counted as a completed birth or automatically as an operational failure. Missing, inconsistent, or insufficiently trusted evidence yields *UNKNOWN*. A recursive lineage claim requires an observed descendant-initiated next transition; it does not require every deliberately bounded terminal node to produce another child.

Evidence for a recursive behavioral claim should connect a parent request to a specific child and then connect the child's action to a further validated descendant. A source hash without observed execution is insufficient. For parent-derived artifact transfer, the provenance of the actual transferred bytes must also be observed; common-source provisioning alone does not establish it. For a physical-host claim, host identity and guest-to-host mapping should be observable outside the guest's self-report.

For a model-directed claim, the proposed criterion is a consequential model selection among at least two feasible, permitted alternatives, subject to a separately described policy gate. Preserve the alternatives, model context and output, parser and fallback, gate decision and executed action. A fixed-policy baseline should have equivalent permitted actions, resources and bounds. Merely recording a model proposal followed by an action does not by itself show the proposal affected that action. A future controlled variation of valid model outputs may test that causal contribution. These are proposals, not experiments reported in the archive.

Controls should include unavailable or unauthorized targets, invalid inherited content, identity mismatch, exhausted limits, duplicate submissions, and interrupted provisioning. Missing or inconsistent collection should produce an unresolved result rather than a pass. The protocol should state whether controller removal is expected to stop the experiment safely or whether continuing without it is the separate hypothesis under test.

Independent runs across the environments named by the claim, predeclared sample size, retained failure cases, and separately held observations would strengthen a future study. Counts and duration distributions should be reported only for measurements actually made. Independent review would improve confidence but would not retroactively change the boundary of the original one-host record.

6. Discussion

The observed parent-child-grandchild sequence is a meaningful managed replication result: an agent created within the run submitted the request that led to another agent. It is not a self-sufficient system. The host provided provisioning, identities, package delivery, scheduling, and policy, and the observed execution remained within one physical machine. At the guest boundary the result is agent-initiated, fixed-policy reproduction under external provisioning; at the larger boundary it is a host-managed service with participating agents.

The evidence also illustrates why controls belong in the technical result. The generation-limit refusal explains why the observed lineage stopped, while duplicate suppression and cleanup describe how the experiment constrained its own effects. The separate crash recovery terminated unfinished work; it should not be described as uninterrupted autonomous persistence. A host-test pass, enrollment result, replication lineage, and beta acceptance audit answer different questions.

Theoretical cross-environment reproduction is a research question, not an observed capability of the examined Garden executable. A vulnerability may be considered as a hypothetical risk factor in a threat model, but the records neither identify one nor supply evidence that it was used. Comparative or universal claims would require additional studies and independent evidence.

7. Conclusion, Limitations, and Future Directions

The preserved primary run records two parent-requested, controller-provisioned descendant launches in one bounded experiment, with one seed, one child, and one grandchild. The host supplied a common package, and the reported reproduction-source digest matched across the three separately identified replicas. The evidence does not show parent-to-child copying of source or live state. A trusted controller and prepared substrate supplied resources and enforced the recorded policy. The model generated greetings while fixed program logic requested reproduction.

The main limitations are the single observed run, a single physical Windows host, correlated host-controlled evidence, prepared identities and software, a finite lineage, and no independent replay. The experiment did not test physical 4 GB operation, self-provisioning across physical computers,

long-duration continuation, or model-selected reproduction. A source digest does not establish full-state equivalence; a duration from one run is not a population performance estimate. Separate recovery, enrollment, and historical readiness records retain their own denominators and scope.

Future work could preregister authorized cross-environment evaluations, compare fixed and model-selected policies under identical bounds, obtain independently held host-mapping and event records, and repeat tests across named environments. A public reproducibility package would require a separate review of the preserved artifacts and their disclosure boundaries. This report remains a non-peer-reviewed research draft; no public artifact release or new experiment is represented.

Data and code availability

The primary run report, journal, guest and controller source, retained package and selected disk images are held in local project archives. They are not publicly available at the time of this draft. Appendix A lists paths and digests for identification by readers who already have authorized access; it is not an independently accessible reproducibility package. The exact runnable manifest, build correspondence and complete historical selection denominator remain partly UNKNOWN. A future release would require a deliberate disclosure review, sanitized artifacts where appropriate, and an accurate manifest of any unavailable or altered material.

References

- Black, S., Cooper Stickland, A., Pencharz, J., Sourbut, O., Schmatz, M., Bailey, J., Matthews, O., Millwood, B., Remedios, A., and Cooney, A. (2025). *RepliBench: Evaluating the Autonomous Replication Capabilities of Language Model Agents*. arXiv preprint 2504.18565v2. [Versioned paper](#). Methodological context; it does not evaluate Garden.
- Cohen, F. (1987). *Computer Viruses: Theory and Experiments*. Computers & Security, 6(1), 22-35. [Journal article](#). Historical context, not an assertion that Garden is a virus or infects other software.
- Kapoor, S., Stroebel, B., Siegel, Z. S., Nadgir, N., and Narayanan, A. (2025). *AI Agents That Matter*. Transactions on Machine Learning Research. [Publication record](#). Context for reporting and evaluation of agent systems.
- Kinniment, M., et al. (2023). *Evaluating Language-Model Agents on Realistic Autonomous Tasks*. METR institutional pilot report. [Research report](#). Context for model-agent evaluation, not a comparison run against Garden.
- Pan, X., Dai, J., Fan, Y., Luo, M., Li, C., and Yang, M. (2025). *Large Language Model-Powered AI Systems Achieve Self-Replication with No Human Intervention*. arXiv work-in-progress preprint 2503.17378v2. [Versioned preprint](#). The authors' reported model-agent studies have different task and infrastructure assumptions from Garden.
- Pineau, J., Vincent-Lamarre, P., Sinha, K., Lariviere, V., Beygelzimer, A., d'Alche-Buc, F., Fox, E., and Larochelle, H. (2021). *Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program)*. Journal of Machine Learning Research, 22(164), 1-20. [Journal record](#).

Appendix A. Artifact Provenance

Items [A1-A14] and [A17-A18] are local primary project records inspected on 29 October 2026, not peer-reviewed publications or publicly released evidence. Paths are relative to the local `Hollowing/archive/session-dumps/` archive root. The prefix `M/` denotes `01-AgentVM-Garden-Monorepo/`

beneath that root. Items [A15-A16] are local post-review verification scripts, identified relative to the AblitAI working directory rather than that archive. SHA-256 values are complete file digests recomputed during inspection except where an item is explicitly labeled as a terminal chain digest. Earlier workspace paths embedded in the JSON are not the preserved paths listed here. No item is asserted to be publicly accessible.

[A1] Garden/AgenticVM project records. *Project boundaries: AgenticVM, Garden, and Shared*. Path: 00-START-HERE/PROJECTS.md. SHA-256: a753469f2f27df5913c2f489aea00c5208a2853e77494711e675491b7ee3afef.

[A2] Garden project records. *Garden — project README*. Path: 02-Garden/README.md. SHA-256: 904d23f9a1939360529a42812890f9e7d1d4d9d48c39528929ca697f8855188d.

[A3] AgenticVM/Garden validation records. *Self-replication validation summary*, schema agenticvm.self-replication-validation.v1. Path: M/docs/validation/self-replication-report.json. SHA-256: 938a8d815a8693e5391bc9ee603de11e0e78d01e06988e44c8b501557b27a072.

[A4] Agent-Replicator-v1. *Native replication run report*, run 6c9e2307-31b5-432f-8e79-d4c86bf9b2cc. Path: M/dist/Agent-Replicator-v1/runtime/replication-runs/6c9e2307-31b5-432f-8e79-d4c86bf9b2cc/report.json. SHA-256: 698b8ef0fe39b11bea2964b995a5a0decbb03e75f3431d903501b9db549409e9c. Associated executable: M/dist/Agent-Replicator-v1/Agent-Replicator.exe; SHA-256: 97cb55bac482bab2a2b4e99ee803816a2ba5a1babb4a6c191e6f950bebf22e0. Controller source: M/dist/Agent-Replicator-v1/src/agenticvm/replicator.py; SHA-256: 3ebdaba05b6669db40697de627363f49c9603778705b19bed56481a8b34483a1.

[A5] Agent-Replicator-v1. *Guest reproduction implementation*. Path: M/dist/Agent-Replicator-v1/guest/replica_agent.py. SHA-256: d06cf3f6b319337e9ac3eb84e4b6e1ecd14203165d528e1d16659b8646d26d73.

[A6] Agent-Replicator-v1. *Checkpoint journal*, run 6c9e2307-31b5-432f-8e79-d4c86bf9b2cc. Path: M/dist/Agent-Replicator-v1/runtime/replication-runs/6c9e2307-31b5-432f-8e79-d4c86bf9b2cc/checkpoints.jsonl. File SHA-256: ebca506f5bfc27336a4b5816f0cd140ca5490e7d543a0f4314f11c81df4d295. Terminal chain digest, distinct from the file digest: b64dbdf06e92e4c7fd0367beea4a8754c25216d3cfd3e04659c8ddb515e23274.

[A7] AgenticVM/Garden validation records. *Replication recovery report*, run 60c43693-520f-4708-8883-4e7746498531. Path: M/docs/validation/replication-recovery-report.json. SHA-256: 14a3cd6fe037b4e56ad431b93f7ebb303c085331b8d74f5414cc248a99a86452.

[A8] AgenticVM/Garden validation records. *Delegated validation final summary*, schema agenticvm.delegated-validation.v2. Path: M/docs/validation/delegated-validation-final.json. SHA-256: e93e3846136e5c8bf0cd3fd283bb0910209e9315e11fcadd6ebd2cdaa4676026.

[A9] AgenticVM/Garden validation records. *Delegated enrollment suite*, schema agenticvm.delegated-suite.v2. Path: M/docs/validation/delegated-suite-6faa716d-166b-4ed2-bd44-7795e8abf17d.json. SHA-256: b22357a2bf8abc639f014cc5be390506ab380b7f2465d135aba8cd42aa11fed9.

[A10] AgenticVM/Garden validation records. *Delegated recovery report*, run 2c57f753-0415-47d1-8588-7128110d5350. Path: M/docs/validation/delegated-recovery-report.json. SHA-256: 7636675e10e30fdef1398801b576f2ccd00a55e366decdd5ff878e034a32a6bc0.

[A11] AgenticVM/Garden validation records. *Beta-readiness audit*, dated 21 October 2026, schema agenticvm.beta-readiness-audit.v1. Path: M/docs/validation/beta-readiness-audit.json. SHA-256: 6a1ba415ccefd6a7c90df1923016049b1fe1e7053dac08ced7358359882e14f.

[A12] AgenticVM/Garden validation records. *Native model quality cases*. Path: M/docs/validation/cp8-native-1fm.json. SHA-256: a32de16eee5ff157a8262407b9a5bb3cfd953fb5d308137b50382ccc1cb6d175.

[A13] Agent-Replicator-v1. *Pinned package retained with the principal run*. Path: M/dist/Agent-Replicator-v1/runtime/replication-runs/6c9e2307-31b5-432f-8e79-d4c86bf9b2cc/download/lfm-replica-agent-v1.zip. Package SHA-256: 410263504d08de57fd7fa3b05ea3ef72108b99f028fb67b95a93ef142a2a8284. Its guest/replica_agent.py has source SHA-256: d06cf3f6b319337e9ac3eb84e4b6e1ecd14203165d528e1d16659b8646d26d73.

[A14] Agent-Replicator-v1. *Generation-one guest evidence and summary*, within the principal run's evidence/replica-1/ directory. evidence.jsonl SHA-256: 7484bf8e14b10bd0f62ca8bb9408da85088d34ff24d54878aeeedab379893b25. summary.json SHA-256: 2362703a3841f9b79205b30be0ac3510e25d4988fab73f858b9a9c5081eeb3bf.

[A15] AblitAI Lab. *Post-review journal-chain consistency check*, 29 October 2026. Local working-directory path: analysis/garden-peer-review-20261029/check_chain.py. SHA-256: 072572aee9427606cf2c9c570c60ccd72e776b47e3dc372b9629fa0802c8dbc2. The script reads [A6]; it is not an independent witness of the historical run.

[A16] AblitAI Lab. *Post-review retained-image source check*, 29 October 2026. Local working-directory path: analysis/garden-paper-post-review-20261029/verify-guest-source.py. SHA-256: 8313ce58d02d3bc70c854fa27aa2ec75e4f5aec17d82cc4fb94c844394b2b90b. It uses the archived read-only ext2 reader at M/dist/Agent-Replicator-v1/src/agentivm/ext2.py (SHA-256: 95283a1ff6328670dc476b042528ef1b6d9c084533e8e4dc9d0e911a58d4b3ab) to read peer/replica_agent.py from each retained replica-0, replica-1, and replica-2 data/default.ext2 image under the principal run. The check identifies that source file's bytes; it does not hash or attest each complete image.

[A17] Agent-Replicator-v1. *Agent catalog and pinned package manifest*. Path: M/dist/Agent-Replicator-v1/chain/agent-catalog.json. SHA-256: 92f84db56920eda823e0c274d894a1639148d90edf0f2ab1e43eab4f715b26bc. The catalog identifies the package and model file digests; it is part of the host-controlled archive.

[A18] Agentivm. *Local-model preparation script*. Path: M/scripts/prepare-local-model-test.py. SHA-256: 5b7f3a6241d925eba9bec4ae04ee0a0d90ca8140f3bfc93b357ee25362966f7f. The source specifies a pinned upstream model URL; it does not itself prove that a historical guest fetched it.

Appendix B. Contributions

Ong Jun Kai contributed to the GARDEN project.