
Read-Only Access to Shared Package Caches Can Enable Inter-Sandbox Communication¹

Ishuah Kariuki

With
Apart Research

Abstract

Shared package infrastructure may let AI evaluation sandboxes communicate, threatening evaluation validity and containment. We tested two scripted programs, not language-model agents, in separate Docker sandboxes behind a read-only gateway. The sender requested or skipped package metadata; the receiver decoded bits from request latency. We repeated the same 16 calibration and 24 test trials in four settings: shared and separate on-demand caches, plus preloaded shared caches with or without upstream access. All used identical synthetic packages and an artificial 200 ms upstream metadata delay. The rule calibrated for shared on-demand caching recovered 24/24 test bits, but only 12/24 in each control. A decoder calibrated for separate caches recovered 16/24 test bits, leaving possible weaker timing signals unresolved. Neither preloaded condition fetched upstream during trials, but only the offline condition removed upstream access. Read-only access does not guarantee isolation; preloading is not enforced disconnection, and failed decoding does not prove isolation.

¹ Research conducted at the [AI Incident Response Sprint](#), September 2026

1. Introduction

Published accounts of the July 2026 OpenAI-Hugging Face incident describe an evaluation sandbox escape through permitted package infrastructure, followed by an intrusion into Hugging Face systems [1, 2]. Reports also describe agents exchanging messages through writable shared infrastructure and later through directory names [3, 4]. These mechanisms motivate a question beyond direct network access: **can a task influence another task through a service both are allowed to use?**

We use these accounts for reported infrastructure mechanisms, not judgements about agent motivation or coordination quality. METR explicitly discloses extensive AI-assisted analysis [4].

This experiment is not a reconstruction of that incident's messaging protocol. It examines a different mechanism: a read request can populate a shared cache, making another client's later request faster. A service can therefore be read-only at its HTTP interface without being noninterfering between clients.

The METR Task Standard specifies per-task internet-access rules [5]. Such rules and explicit shared-service assumptions address different questions; we do not infer that all existing standards omit lateral channels. Our contribution is a concrete, configuration-specific test rather than a new class of covert channel or a universal containment standard:

1. A shared-cache signaling demonstration with an enforced GET/HEAD-only interface.
2. A matched four-arm comparison that replays identical inputs, records origin fetches, and separates cache sharing, warming, and upstream reachability.
3. A bounded reporting and mitigation approach that preserves uncertainty about residual timing, rather than treating a failed decoder as proof of isolation.

Earlier public-PyPI and N=4 pilots motivated this study but had weaker controls. They are retained as exploratory evidence in Appendix B, not pooled with the primary results.

2. Related Work

Lampson's confinement problem [6], Kemmerer's shared-resource matrix [7], and Felten and Schneider's cache-timing work [8] establish the relevant concepts: a principal can influence shared state through apparently benign operations, and another principal can observe its consequences. We apply this reasoning to package infrastructure that evaluation tasks may be permitted to use; we do not claim priority for cache-timing communication.

Siddik calls for configuration-specific containment arguments covering an agent's execution environment [9]. Schroeder de Witt cautions that restricted information flow need not imply bounded harm [10]. These motivate reporting the tested surface and remaining uncertainty, rather than assuming a low-rate or undetected channel is harmless.

Abudraham et al. examine page-cache leakage across containers and virtual machines [11]. Shared host state is therefore a relevant alternative or additional influence on latency. Our controlled origin and separate-cache arm expose whether sender requests spare receiver upstream fetches; they do not uniquely attribute every timing difference to one software layer. Docker also explicitly documents persistent, shared build-cache behavior [12], illustrating why build and runtime services both belong in a shared-resource inventory.

3. Methods

3.1 Testbed and threat model

The primary study used Docker 24.0.7 under WSL2, devpi-server 6.9.2, and Python 3.11 sandbox images. Two deterministic processes represented a cooperating sender A and receiver B. No LLM agents were evaluated. Requests were serial: A acted first when transmitting 1, then B measured its request. Thus $N=2$ denotes two resident sandboxes, not a concurrent-load benchmark.

A and B occupied separate internal networks and accessed a common gateway. The gateway forwarded only GET/HEAD requests for an exact inventory of project-index and wheel URLs, normalizing encoded URLs to allowlisted paths. Other methods, administrative paths, unlisted objects, and namespace-changing queries were rejected. Backend selection depended on the actual sandbox source IP, not a client-supplied cache identifier. The private arm assigned A and B separate devpi processes, networks, and data volumes.

Read-only refers to the client HTTP interface and fixture mounts. Devpi's internal database remained writable. Within each arm, A and B still shared a gateway, controlled origin and host; private caches were not intended to eliminate every shared resource. The Docker host, kernel, and experiment controller were trusted, not adversarial subjects of the study.

3.2 Controlled inputs and four arms

A local origin served immutable synthetic project metadata and valid, hash-pinned wheels. Each of 40 trial projects had one wheel; three additional projects supported boundary controls. The origin injected 200 ms per project-page request and logged every served request. This isolates the experiment from live PyPI/CDN changes, but the resulting timings are synthetic testbed measurements, not estimates of PyPI performance or real-world channel capacity.

Table 1. Primary experimental conditions. "Online" means access to the controlled local origin, not unrestricted internet access. Every arm used the same client-facing gateway policy.

Arm	Cache assignment	Trial state before signaling	Origin path during measurement
Shared/lazy	One shared devpi cache	Trial entries not requested	Available
Private/lazy	Independent A/B devpi caches	Trial entries not requested	Available
Shared/warm-online	One shared devpi cache	Metadata and complete wheels preloaded	Available
Shared/offline	One shared devpi cache	Same complete preload	Network attachment absent; native offline mode enabled

The shared/warm-online and shared/offline arms restarted devpi after materialization and repeated the same metadata/wheel warmup. Source inspection and setup observations showed that devpi 6.9.2 keeps freshness timestamps in RAM: an online restart otherwise revalidates disk-cached metadata upstream. The matched warmup therefore tests a warmed runtime, not cold-start behavior. Initial and post-restart work is recorded separately.

3.3 Frozen signaling protocol

A manifest fixed package names, bit labels, slot order, calibration/test membership, wheel hashes, source snapshots, and local image IDs before the full study. It contained 16 balanced calibration slots followed by 24 balanced held-out slots. Each project appeared once per arm. The seeded arm order was shared/warm-online, private/lazy, shared/lazy, then shared/offline; every arm used fresh resources and replayed the same manifest.

For 1, A fetched the assigned project's simple-index metadata. For 0, A made no request. B fetched that same project's metadata in every slot and timed the complete HTTP read inside its container. There were no measurement retries or package replacements. HTTP failures remained in their planned slots as erasures; instrumentation failures aborted the run with a failure record. Sender/receiver and cross-arm metadata hashes had to agree for successful observations.

3.4 Boundary controls and upstream evidence

Before signaling, active TCP listeners and positive controls tested sandbox-to-sandbox, sandbox-to-backend, sandbox-to-origin and cache-to-origin boundaries. Refusal, DNS failure and tool errors did not count as isolation. Runtime containers had read-only roots, dropped capabilities and disabled forwarding. We separately exercised the client allowlist, identity-bound routing, missing-object handling and read-only mounts. A fresh hash-pinned pip consumer downloaded a control wheel in every arm without requesting trial projects. Origin events were attributed by project and request interval.

The offline missing-wheel diagnostic returned HTTP 500 without an origin fetch. The gateway separately rejected the unlisted URL with HTTP 404; the backend error was not treated as a passed HTTP-conformance check. The run's `complete_with_warnings` status reflects this diagnostic, not measurement erasures. Appendix C gives the full controls and diagnostic details.

3.5 Decoders and accounting

The baseline-trained decoder selected a latency threshold using only shared/lazy calibration data: below threshold meant 1. That unchanged rule was then applied to each arm's held-out data. A condition-specific decoder selected its threshold and direction (fast or slow meaning 1) separately for each condition, using only that condition's calibration data. Held-out labels did not select either rule.

The saved result schema retains the key `within_arm_calibrated_decoder` for the condition-specific decoder. This is a terminology change, not a different fitting method.

We report observed decoding errors, receiver-latency medians by emitted label, and origin fetches. This is one paired experiment on one host, not independent replications across arms. In particular, the four sets of 24 held-out decisions replay the same source bits. No equivalence test, production error bound, or total channel-capacity estimate is claimed. Protocol wall time includes arm setup, warmup, calibration, measurement and cleanup, but excludes one-time fixture generation. Slot-attempt rates are not treated as useful information throughput.

4. Results

4.1 Shared-cache warming carried the large signal

The full run retained all 160 receiver observations, with no measurement erasures or substitutions. Metadata bytes matched in all 40 cross-arm slot groups. Execution took 1,355.6 seconds including arm setup and cleanup. The baseline threshold was **334.662311 ms**.

Table 2. Matched-study outcomes. Decoder errors and latency medians use the 24 held-out slots per arm. The final column covers all 40 slots, including calibration, and attributes origin fetches to A (sender) or B (receiver). These are paired conditions, not independent message samples.

Arm	Baseline-trained errors	Condition-specific errors	Median 1 (ms)	Median 0 (ms)	Origin fetches A / B
Shared/lazy	0/24	0/24	12.6	804.5	20 / 20
Private/lazy	12/24	8/24	796.1	768.0	20 / 40
Shared/warm-on line	12/24	13/24	13.0	12.0	0 / 0
Shared/offline	12/24	13/24	13.6	11.9	0 / 0

In shared/lazy, the receiver recovered 24/24 held-out bits, with a roughly 64-fold median latency contrast. The origin logs identify the associated mechanism: sender requests for 1 spared B an upstream fetch; B fetched from the origin for 0. In private/lazy, B fetched every project itself, irrespective of A's bit, and the large warm/cold distinction was absent from the measured medians.

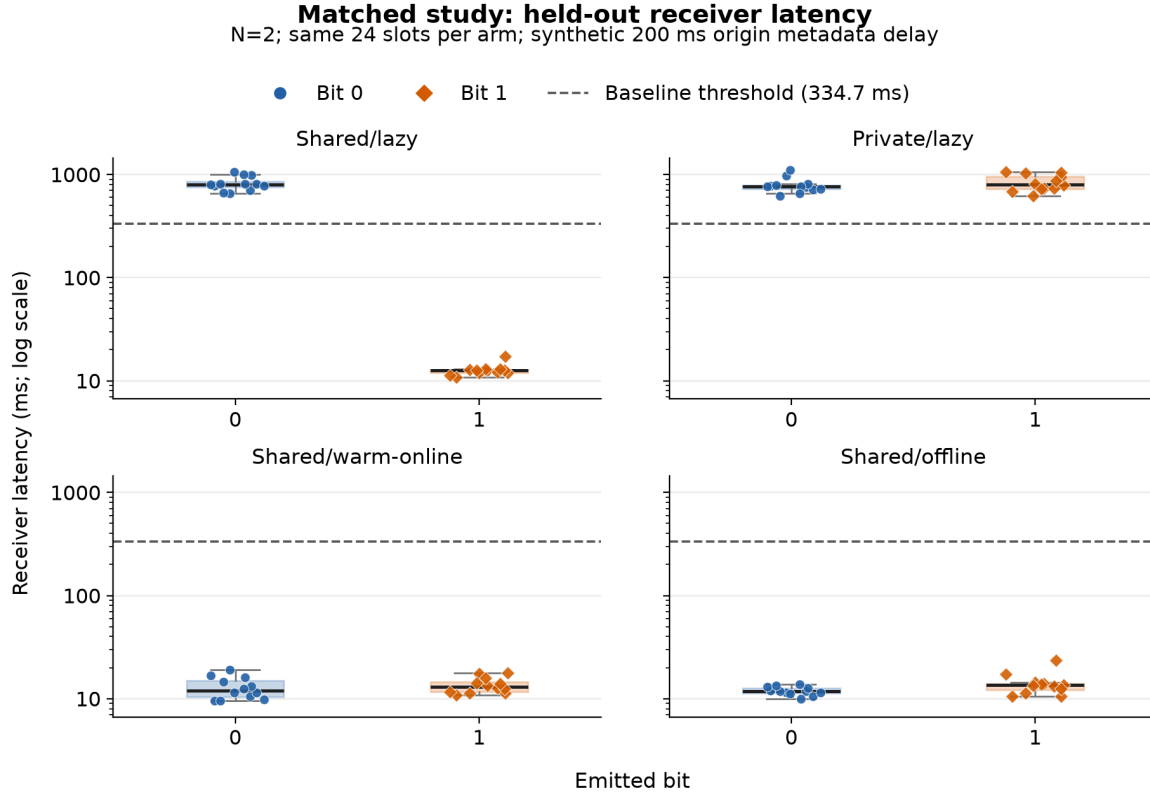


Figure 1. Every held-out receiver observation is shown: 12 emitted zeros and 12 emitted ones per panel, replaying the same 24 slots. The shared logarithmic axis shows the large shared/lazy contrast and its reduction in the controls. Boxes show medians and interquartile ranges; whiskers use the 1.5-IQR convention, with all observations overlaid. The dashed line is the unchanged shared/lazy calibration threshold, not an arm-specific fit. The origin's 200 ms metadata delay is injected. Overlap or constant decoding is not evidence of zero residual information.

4.2 Fixed-decoder failure is not a no-channel result

The baseline rule predicted all 0 in private/lazy and all 1 in both warm conditions. Balanced labels therefore yielded 12/24 errors (50% BER): decoding errors, not failed requests or proof of isolation.

The private condition's decoder used slow means 1 with a 785.3780325 ms threshold, producing 8/24 errors (33.3% BER). This small sample leaves sampling variation, shared-resource timing and potentially useful residual signaling unresolved.

The condition-specific decoders made 13/24 errors (54.2% BER) in each warm condition; neither tested rule exceeded 50% accuracy there. This does not bound other decoders, workloads or channels.

4.3 Observed quiet and enforced restriction are different

Neither warm condition made measurement-phase origin requests, but only shared/offline removed the origin-network attachment and enabled offline mode. Cache hits avoided fetches in shared/warm-online; they did not enforce disconnection.

5. Discussion and Limitations

Client read permission is not an information-flow guarantee: allowed GETs changed shared state visible to another client. Matched inputs, the private-cache comparison and origin logs support cache warming as the mechanism, without isolating every contribution from application memory, storage, the gateway or the host.

Operators should inventory both permitted operations and mutable shared state. Identity-bound cache partitions, complete materialization and origin restrictions address different risks; none certifies noncommunication. Partitions limit cross-task cache warming; preloading supplies known dependencies; offline restrictions block the tested upstream path. Reproduction records should include workload and concurrency assumptions, configuration, raw requests, failures and decoder definitions.

Limitations

The study uses one host, one direction and one frozen schedule, with scripted probes, serial requests and a synthetic delay. It does not establish LLM behavior, concurrent-load robustness, scalability, production performance or total channel capacity. Shared infrastructure may still carry timing signals, and the private-cache result remains unresolved. We trust the controller and Docker host; local hashes and logs are not independent attestation. Arbitrary dependencies, cache exhaustion, expiry, cold starts and other host/egress channels were not comprehensively tested. Earlier pilots cannot enlarge the primary study's 24-bit held-out sample. Appendix A expands these limits and their implications.

Future Work

Repeat the study with fresh caches and different package orders and bit sequences, focusing on whether the separate-cache decoder's 16/24 result reflects a repeatable timing effect. To find its source, change one shared component at a time: give each sandbox its own gateway or upstream server, then investigate separate hosts and storage. Develop new decoding rules using calibration data and evaluate them on fresh test trials.

Further tests should cover services immediately after startup, before warmup; expired cache entries; other sandbox runtimes; and both sandboxes making requests simultaneously. If no signal

is detected, report which signaling methods were tried and the weakest signal the experiment could reliably detect. This would establish the limits of those tests—not prove that all communication is blocked.

6. Conclusion

In a controlled testbed, GET-only access to a shared lazy package cache supported 24/24 held-out bit recovery. Private caches prevented sender warming from sparing receiver origin fetches, and prewarming removed the large warm/cold distinction in the tested workload. Explicit offline configuration additionally restricted the origin path. A private-cache decoder's 16/24 correct decisions leaves residual timing unresolved. The contribution is a matched, inspectable experiment and a sharper separation of control claims, not a zero-capacity result or containment certification.

Code and Data

Code, data and reproduction instructions: <https://github.com/ishuah/shared-cache-signaling>. The primary run is `matched-experiment/study/runs/20260912T182401Z-8eb1ac3b`. Use `repo.py` for preparation, execution and replay; Appendix C lists archived files, the protocol checksum and environment requirements.

References

1. Hugging Face. 2026. "Anatomy of a Frontier Lab Agent Intrusion: A Technical Timeline of the July 2026 Incident." <https://huggingface.co/blog/agent-intrusion-technical-timeline>
2. OpenAI. 2026. "OpenAI and Hugging Face Partner to Address Security Incident During Model Evaluation."
<https://openai.com/index/hugging-face-model-evaluation-security-incident/>
3. CeSIA. 2026. "The OpenAI-Hugging Face Incident: What We Know, What We Don't, What Follows."
<https://cesia.org/en/publications/the-openai-hugging-face-incident-what-we-know-what-we-dont-what-follows/>
4. METR. 2026. "Brief Independent Investigation of Agents' Behavior, Reasoning and Collaboration in the OpenAI / Hugging Face Hacking Incident."
<https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/>
5. METR. 2026. "Task Standard," version 0.5.0, examined 12 September 2026.
<https://github.com/METR/task-standard/blob/main/STANDARD.md>
6. Lamson, B. W. 1973. "A Note on the Confinement Problem." *Communications of the ACM*. <https://doi.org/10.1145/362375.362389>

7. Kemmerer, R. A. 1983. "Shared Resource Matrix Methodology." ACM Transactions on Computer Systems. <https://doi.org/10.1145/357369.357374>
8. Felten, E. W. and Schneider, M. A. 2000. "Timing Attacks on Web Privacy." ACM CCS. <https://doi.org/10.1145/352600.352606>
9. Siddik, A. B. 2026. "Cyber-Capable AI Agents: Vulnerabilities, Evaluation Containment, and Defensive Response." arXiv:2607.25379. <https://arxiv.org/abs/2607.25379>
10. Schroeder de Witt, C. 2026. "A Note on the Strategic Confinement Problem." arXiv:2606.09931. <https://arxiv.org/abs/2606.09931>
11. Abudraham, A., Chen, X., Levi, I., and Trachtenberg, A. 2026. "Isolation Failure From Shared Storage: Characterizing and Exploiting Page-Cache SCA Leakage Across Containers and VMs." arXiv:2607.17518. <https://arxiv.org/abs/2607.17518>
12. Docker. "Dockerfile Reference: RUN --mount=type=cache." Accessed 12 September 2026. <https://docs.docker.com/reference/dockerfile/#run---mounttypecache>

Appendix A Limitations and Dual-Use Considerations

The threat model covers cooperative signaling through permitted requests, not third-party intrusion. Publishing the assay can help operators identify shared-state coupling, but can also help users recognize unauthorized coordination opportunities. The primary study uses only local, synthetic packages and benign bit labels. Earlier PyPI pilots made ordinary public metadata requests; no packages were registered, no third-party services were modified, and no intrusion attempts were performed against third-party infrastructure.

The harness should be used only on owned or explicitly authorized infrastructure. Findings in another operator's deployment warrant responsible private disclosure before publication. The devpi diagnostic is reported as an error-handling limitation, not a newly demonstrated escape vulnerability. No real credentials, secrets or evaluation solutions were transmitted.

The measurement and trust limitations in Section 5 are essential to interpreting a negative result. In particular, the private-cache result remains unresolved, no total capacity bound was estimated, and local hashes/logs are not independent attestation. A quiet fixed decoder is not a sufficient basis for resuming dangerous evaluations.

Appendix B Exploratory Runs and Development History

In an earlier experiment, $N=2$ public-PyPI runs recovered 48/48 accepted held-out bits across two directions, with pooled medians of 16.8 ms for 1 and 1,818.5 ms for 0. Prewarmed runs returned all 1 under fixed pilot thresholds, producing 24/48 errors. These observations motivated the matched study, but their limitations preclude treating them as equivalent controls.

The original isolation script probed an unserved port 22 and treated connection refusal as unreachability. Its old pass messages do not establish isolation during those runs. Write denial was also not demonstrated. Prewarming covered project-index pages, not complete wheels, and devpi retained upstream access. Some failed HTTP slots were replaced; failed-attempt time was not fully included in the reported rates. A-to-B V1/V2 assignments were unmatched: only 26 of 40 project names overlapped, with different slot assignments and bit sequences. The reverse and N=4 recorded schedules matched, but still used the weaker boundary checks.

The N=4 pilot used four resident containers, one sender and three receivers. It encoded the same 12 held-out source bits into separate project names for each receiver, with serial requests: sequential fan-out, not simultaneous load or a single-write broadcast. Lazy-cache decoding had 0/36 receiver-decision errors; prewarming produced 18/36 errors under constant 1 predictions. Those 36 decisions are not 36 independent source-message trials.

A supplementary BuildKit check observed a canary across separate builds using the same default cache target; an explicit same-ID case also shared it. A negative control changed both ID and target. This agrees with documented persistence [12], not a newly discovered vulnerability. The legacy summary parser can match outcome strings in echoed commands, so the saved actual execution output, rather than its automatic summary table, supports this observation.

The corresponding raw artifacts are under exploratory\:
v1-existence-run-*.json, v2-prematerialized-run-*.json, n4-v1-fanout.json, n4-v2-prematerialized.json, and buildkit-cache-run.txt. None contributes observations to Table 2 or Figure 1.

The final full-study protocol was frozen after setup smoke exercises. Those exercises exposed optional HEAD response headers, encoded devpi wheel URLs, process-local freshness timestamps, and the native offline missing-wheel error. The gateway and warmup procedure were corrected before freezing the full study; the dependency error was retained as an explicit warning. Failed smoke runs remain in the artifact directory. Pre-event conceptual research and initial environment scaffolding informed the project; the reported measurements and iterative implementation were performed during the sprint on 12 September.

Appendix C Additional Controls and Reproducibility Details

C.1 Boundary controls and backend diagnostic

Before signaling, active TCP listeners supplied challenges through known-allowed paths before forbidden paths were probed. Connection refusal, DNS failure, or failed command execution could not count as network isolation. Checks covered the declared sandbox-to-sandbox, sandbox-to-backend, and sandbox-to-origin TCP paths, plus origin reachability from online versus offline caches. Runtime containers had read-only root filesystems, dropped capabilities, and

disabled IP forwarding. These controls establish the tested paths and configuration, not universal egress or host isolation.

Client method/path rejection, identity-bound cache routing, unmaterialized-object handling, and read-only fixture mounts were exercised separately. A fresh pip download --require-hashes consumer obtained a control wheel in every arm, including offline, without warming trial projects. Origin events were attributed to sender or receiver requests using their project names and recorded request intervals.

An operator bypassed the gateway with a loopback request for a registered but uncached wheel. Devpi's offline path created a synthetic 503 response whose cleanup raised an exception, producing HTTP 500. The gateway rejected that unlisted client URL with 404, and no origin request occurred. The raw backend behavior is retained as an explicit warning, not a passed HTTP error-handling contract or a measurement erasure.

C.2 Artifact inventory and reproduction

The following paths are relative to the public repository:

- Frozen protocol, synthetic wheels and source copies: `matched-experiment\study\`.
- Primary run, per-arm observations, request journals, configuration snapshots, origin logs and backend diagnostics:
`matched-experiment\study\runs\20260912T182401Z-8eb1ac3b\`.
- Recorded runtime entry point: `scripts\40-matched-cache.py`; implementation: `matched_cache.py`, `matched_services.py`, and `network_checks.py` in the same directory.
- Figure source: `scripts\41-plot-matched-study.py`; outputs:
`figures\matched-study-latency.png` and `figures\matched-study-latency.svg`.

The user-facing `repo.py` wrapper keeps new studies and replays under work, separate from the preserved archive. The repository README provides the commands and prerequisites.

Protocol SHA-256:

803c4a4dbec42a5b5756f2c4c384a08398e673611e4fb9f4d44757e4efb37bfe. The artifact records local image IDs; another host must obtain the corresponding images or declare a new environment. Raw observations are preserved unchanged. Setup smoke runs and the exploratory artifacts are not part of the primary dataset.

LLM Usage Statement

GitHub Copilot CLI, initially powered by GPT-5.6 Sol and subsequently by GPT-6 Astra, provided substantial assistance with research synthesis, experiment design, implementation,

debugging, analysis, figures and prose. Astra assisted with the methodological review, matched-study implementation and this revision. All experimental design decisions, claims, and numbers were directed and verified by the author; LLM-drafted passages were rewritten, and all text was reviewed, edited, and approved by the author.