You Bought It, They Bricked It: A Call for Reclaiming Local Control

Jack Tigar Humphries
jhumphri@cs.stanford.edu
Stellar Development Foundation
San Francisco, CA, USA

ABSTRACT

Cloud-tethered devices can become e-waste even while still functional when vendors retire required applications or services. Vendors need not support devices forever, but ending support and blocking replacement software are distinct choices. The labor required to reconstruct proprietary management software has provided a plausible technical justification for retaining vendor control. Our early experience suggests that foundation models can sharply reduce this cost. Using AI, we recovered two versions of Apple's encrypted, undocumented AirPort device-management protocol and built a compatible client tested across six models and four firmware versions. We estimate that reproducing this recovery with the resulting methodology would require four to five days of human work rather than months of manual reverse engineering. De-cloudification is not yet automatic as progress remained constrained by trace coverage, behavioral oracles, and experiment selection. Nevertheless, this result suggests that recovering owner-controlled operation can become cheap enough to make bricking locally operable hardware a business choice rather than a technical necessity.

This paper therefore calls for an end to non-consensual e-waste. We propose an end-of-life control-transfer principle: when support ends, vendors should provide local operation, release the information and authority required by replacement software, or transfer trust to an owner-authorized controller. To make independent recovery dependable, we formulate AI-assisted de-cloudification as a closed-loop experimental process and identify the systems challenges it raises. AI-assisted recovery should remain a fallback, not a substitute for a deliberate end-of-life path.

1 INTRODUCTION

Cloud-tethered devices can fail before their hardware does. A vendor may retire an application or service, withdraw an integration, or make important functionality contingent on a subscription. Supporting deployed devices indefinitely is not free, yet vendors may also benefit when discontinuation forces owners to purchase replacements. Because vendors control the software path, they can decide that otherwise functional hardware has reached the end of its useful life. The result is non-consensual e-waste: owners must replace equipment not because its hardware failed, but because a vendor withdrew the means or authorization to use it.

Recent products illustrate both the consequences of vendor control and the different ways it is exercised. Spotify discontinued its $90 Car Thing just five months after launch, disabled all devices, and directed users to recycle them [13]. After acquiring Revolv, Nest shut down the cloud service required by its $299 smart-home hubs [22, 25], rendering every hub inoperable. Belkin ended Wemo cloud services and app support, causing affected products to lose app control, remote access, and voice-assistant integration [6, 12].

Even when devices are not immediately disabled, vendors can retain long-term control through proprietary software. Apple requires its proprietary AirPort Utility to configure its AirPort line of routers [2–4], yet the application is not guaranteed to function on iOS 27 or macOS 27 [14], forcing users to retain older operating systems simply to manage their own otherwise functional hardware. eero reserves daily and monthly usage histories and security analytics for paying eero Plus subscribers [19, 20]. Samsung added advertisements to previously sold Family Hub refrigerator displays through a software update [17, 42]. BMW offered subscriptions to activate heated-seat hardware already installed in vehicles, then abandoned the program after customers rejected paying to unlock hardware they already possessed [26, 27]. These cases differ in severity, but share the same structure. The owner possesses working hardware while the vendor retains unilateral control over whether, how, and on what terms its software-mediated functions may be used.

Today, reconstructing management software for a closed device can require months of expert reverse engineering, giving vendors a plausible cost justification for retaining control. Our AirPort result suggests that this justification is weakening. Using AI, we recovered two versions of Apple's encrypted, undocumented AirPort Configuration Protocol and built a compatible client tested across six models and four firmware versions. We estimate that reproducing this recovery with this methodology would require four to five days of human work rather than months. Although coverage, behavioral oracles, and experiment selection remain open systems problems, this result suggests that, for devices whose

core functions can operate locally, bricking is increasingly a *business* choice rather than a *technical* necessity.

Our call to action is simple: **support may end, but control should transfer.** Vendors should not be required to operate applications and cloud services indefinitely; when they stop, however, they should provide a local operating mode, release the information needed to build replacement software and the authority needed to operate it, or transfer trust to an owner-authorized controller. We should no longer permit companies to make technical excuses for repossessing hardware they do not own. If no end-of-life path will be provided, consumer-protection rules should require conspicuous disclosure before sale, including the minimum guaranteed support period and the functions that may disappear afterward. For devices whose core functions can be performed locally, preventing continued operation after support ends is not an unavoidable technical consequence of discontinuation. It is a product decision that deprives owners of continued use of otherwise functional property.

When a vendor does not provide such an exit path, **decloudification** can serve as a technical fallback. De-cloudification requires recovering enough device-control semantics to build a local controller, open integration, or replacement management application. One alternative is to install replacement firmware, but doing so is often infeasible or unnecessarily invasive. A device may enforce signed firmware [23, 43], and even an unlocked device may depend on proprietary drivers and hardware-specific logic that is difficult to reproduce [10, 24, 34]. A protocol-compatible application preserves the existing firmware and replaces only the management path. Recovering that protocol, however, remains labor intensive. An expert must connect observations from the running system to evidence in its implementation, formulate hypotheses explaining the observed behavior, implement those hypotheses, and iteratively validate them against the original system's behavior [8, 9, 11, 29, 36, 46]. This process depends heavily on human judgment and does not scale across closed or abandoned devices.

Foundation models may substantially change this cost structure [7, 15, 21, 28, 31–33, 37, 41, 47, 48]. Given traces, binaries, decompiler output, and partial implementations, an AI system can quickly generate plausible message formats, state machines, cryptographic handshakes, and command semantics. But inexpensive hypotheses do not make protocol recovery complete. A candidate can explain every observed packet while misunderstanding a field, omitting a state transition, or mishandling an unobserved failure case. As hypothesis generation becomes cheaper, the bottleneck shifts to **knowing which hypothesis is correct.** This requires constructing inexpensive behavioral oracles, collecting observations with sufficient coverage, and choosing the next experiment that will most reduce uncertainty. The central question is no longer only "What might this protocol do?" but "What observation would distinguish this explanation from the alternatives?" As AI reduces the cost of recovering missing semantics, the remaining barrier is whether vendors preserve or withhold the authority that replacement software needs to operate.

This paper calls for end-of-life control transfer to end nonconsensual e-waste, with clear pre-sale warning when no exit path exists. Using AirPort as early evidence, we formulate AI-assisted reverse engineering as a closed-loop experimental process and identify the systems challenges to overcome for dependable independent recovery: coverage, inexpensive oracles, experiment planning, faithful test environments, efficiency, confidence estimation, and responsible recovery. As recovery becomes practical, vendor-controlled lifetimes become a business and design choice; consumers deserve an exit path or pre-sale warning.

2 EARLY EXPERIENCE RECLAIMING LOCAL CONTROL

We test whether AI-assisted recovery can provide a fallback when a vendor does not provide an end-of-life control path. Apple has retired its AirPort routers and announced that its proprietary management application, AirPort Utility, will no longer be supported on macOS 27 or iOS 27 [14]. AirPort combines functional black-box hardware, encrypted traffic, and undocumented protocol semantics, while the official utility and physical devices provide ground truth for testing a replacement. Our goal was not merely to describe the AirPort Configuration Protocol (ACP), but to recover enough semantics to build a compatible AirPort Utility replacement.

Our initial approach was to ask Codex [40] with GPT-5.5 Medium [39] to reconstruct high-level code directly from the AirPort Utility binary. This approach quickly became inefficient as even a button is entangled in compiled graphics and user-interface frameworks irrelevant to reproducing its behavior. Worse, reconstructed code was difficult to connect to an observable device effect and therefore difficult to validate. We instead treated the running application as the primary specification. We recovered its external behavioral contract from traces and inspected the binary only when behavior did not expose the information needed to reproduce it.

We ran AirPort Utility 6.3.9 and 5.6.1 on macOS Sequoia 15.7.7 and manually exercised their graphical interfaces while recording three timestamped evidence streams consisting of PNG screenshots, Accessibility-tree snapshots [1] that described the visible controls and their contents, and PCAPs captured with `tcpdump` [35]. Codex inferred user actions and their corresponding protocol exchanges by leveraging trace timestamps to align changes in the interface with packets

sent to and returned by the device. We attempted to enumerate the interface systematically and added workflows when testing exposed missing behavior. We began with one Time Capsule (model A1254), then expanded to two AirPort Expresses (A1088/A1392), two AirPort Extremes (A1354/A1521), and one last Time Capsule (A1470) running four firmware versions from 6.3 through 7.9.1. The A1088 used the first version of ACP; the other five devices used a second, substantially different version. Prior work implemented portions of legacy ACP, including individual property reads and writes, but left many protocol features and complete management workflows unsupported [49]; we supplied this material to Codex.

The first PCAP revealed why traces alone were insufficient as ACP traffic was encrypted. Codex dumped symbols from both AirPort Utility binaries, used Ghidra [38] and command-line object-file tools to follow relevant code into private macOS frameworks, and attached LLDB [44] to determine which candidate cryptographic routines actually executed. Breakpoints on key-derivation and cipher-initialization routines captured per-session directional keys and associated them with individual TCP flows. Modern ACP authenticates using SRP-6a [16, 45], derives separate request and response keys using PBKDF2-HMAC-SHA1 [30], and encrypts each direction with an AES-128 counter-mode construction; legacy ACP instead uses a 1024-bit Diffie-Hellman [18] exchange before applying the same stream construction. Codex also recovered ACP framing, checksums, property records, and structured serialization, enough to decrypt complete captures and construct a compatible client. This selective use of binary analysis was critical, but we did not otherwise need to inspect the original binary's decompiled assembly.

Codex rebuilt the interface in Swift, using Accessibility snapshots as its primary specification and screenshots when they were ambiguous. It implemented the protocol in Python so that the recovered client could also run outside macOS. It replayed each workflow through Accessibility APIs and mouse-and-keyboard automation. Each replay began from a known router configuration. A reusable Python oracle compared the decrypted packets sent and received by our implementation with those produced by AirPort Utility. It ignored differences that are nonsemantic, such as timestamps and property ordering, while requiring the relevant fields, values, and transaction groupings to agree. On a divergence, Codex revised its model and implementation and replayed the workflow, often for several iterations before we manually inspected its changes and tested the result.

The oracle itself changed as we learned which differences mattered. During Time Capsule onboarding, AirPort Utility wrote a set of properties in one batch, whereas the replacement divided the same fields and values across three writes. Codex initially classified the outputs as semantically equivalent, but the device rejected the three-write sequence. After observing this failure, Codex revised both the implementation and the oracle to preserve the original batch boundary. Transaction structure may therefore encode atomicity or a hidden state transition visible only on the device.

Coverage presented a separate problem. An implementation that matched the Time Capsule traces failed on the AirPort Express (A1088), which uses the older ACP protocol and exposes AirPlay [5] settings absent from the Time Capsule. We responded by purchasing four additional devices and collecting new traces across hardware families, firmware versions, and capabilities. The final corpus represents roughly 184 protocol operations and 144 configuration fields across 6 AirPort devices, including about 2,700 screenshots and Accessibility snapshots and 57,000 ACP packets across 500 TCP flows. Across approximately 870 replay iterations, Codex produced roughly 50,000 lines of source and test code and 650 tests.

Within this evidence boundary, the replacement supports both ACP versions, device discovery and authentication, configuration of the same settings exposed by AirPort Utility, onboarding, error reporting, AirPlay and disk management, factory reset and disk erasure, and installation of Apple-signed firmware. Every recorded workflow ultimately produced semantically equivalent exchanges, and we manually confirmed configuration changes, restarts, resets, firmware installation, and disk erasure on physical devices. The process nevertheless required substantial human judgment as we chose the evidence sources, collected traces, recognized coverage gaps, audited the oracle, inspected intermediate implementations, and physically reset devices that Codex left in invalid states. Some experiments also caused routers attached to our network to advertise weakly secured Wi-Fi SSIDs; an isolated VLAN and automatic restoration of device state would have made exploration safer and cheaper.

This result is evidence of feasibility, not completeness. We tested six models and four firmware versions, could not exercise PPPoE, and sampled only a small fraction of possible network modes, addresses, and authentication schemes. The recovery consumed billions of model tokens, but we estimate that reproducing it with the described methodology would require four to five days of human work rather than the months we would expect from manual reverse engineering. The remaining costs were collecting representative evidence, refining oracles, restoring physical state, and deciding whether a failure called for another implementation attempt or a new trace. These bottlenecks motivate the barriers described next and the recovery loop in Section 4.

3 WHY DE-CLOUDIFICATION DOES NOT SCALE

Our AirPort experience illustrates why de-cloudification does not scale. It requires recovering a device's behavioral contract–not just decoding packets. A replacement controller must reproduce when operations are valid, how sessions are authorized, and how actions affect device state. Replaying one camera request that begins streaming, for example, does not reveal when streaming is allowed or how the device behaves when another client is connected.

Recovering this contract requires connecting *low-level artifacts* to *high-level intent*. A packet capture shows message order without revealing the corresponding action; decompiled code exposes parsers or cryptographic routines without showing how a user reaches them; and field meanings may depend on hidden state. Protocol recovery must therefore explain the original system's behavior, not merely capture traffic or translate assembly.

Evidence is scattered across the application, traffic, logs, and physical device. Investigators must repeatedly move among them, such as tracing an encryption path through a binary before testing competing interpretations on hardware [8, 9, 11, 29, 36, 46]. Each experiment may invalidate earlier assumptions while firmware and hardware differences can silently break an implementation. While existing tools expose the evidence, humans must still decide how the evidence fits together and what to investigate next.

Foundation models can quickly turn traces and implementation artifacts into protocol models and candidate implementations [7, 15, 21, 28, 31–33, 37, 41, 47, 48], shifting the dominant cost to validation. Validation consists of constructing behavioral oracles, gathering coverage, selecting experiments that distinguish competing explanations, and estimating confidence. If vendor-held credentials, remote attestation, or a mandatory cloud service blocks operation even after protocol recovery, the obstacle is authority rather than semantics. That case requires an end-of-life control-transfer path; missing semantics motivate the closed-loop process that follows.

4 TOWARD AI-ASSISTED DE-CLOUDIFICATION

Our AirPort experience suggests that de-cloudification can be organized as a closed-loop experimental process. We envision an AI system that maintains a model of the original protocol together with a candidate implementation, then repeatedly asks: **Which next experiment will most reduce uncertainty?** The answer may be to collect new evidence, revise the implementation, or test a competing explanation. The objective is not to reproduce every internal detail of the original software, but to recover a declared set of owner-relevant behaviors with enough evidence that the replacement can be trusted within that scope.

Figure 1 summarizes this recovery loop. The process begins by constructing an **initial corpus of observations**. Whenever the original system remains available, these traces can be collected from human users to produce a representative corpus. An agent can also exercise the original system's application and record the resulting traffic, interface behavior, and device state. It can begin with common workflows, then vary the initial state or one user input at a time to expose differences in protocol behavior. Binaries and firmware provide additional evidence when behavior alone is insufficient. Each observation should connect an intended action to the messages exchanged and the resulting device behavior, giving the agent a basis for reasoning about protocol semantics rather than treating traces as disconnected byte sequences.

From this corpus, the agent constructs an **executable model** of the device-control protocol. The model must describe not only how messages are encoded, but when they are valid and what effects they should produce. It may contain competing hypotheses about state transitions, authentication, encryption, object relationships, or invariants preserved by the original implementation. Representing this uncertainty is important because several explanations may fit the initial evidence. The model must generate messages and predict responses; prose alone cannot close the loop.

The agent must then decide **what to do next**. If two hypotheses explain the current evidence equally well, the agent should select an experiment whose outcome distinguishes between them. This may require collecting a trace from an unobserved workflow, inspecting a binary function, replaying an interaction under a different state, or fixing a specific divergence in the candidate implementation. The best experiment is not the one that produces the most data, but rather the one expected to eliminate a consequential ambiguity at acceptable cost. The agent must therefore consider expected information gain, oracle quality, and whether the experiment can be executed in a controlled environment.

Each selected experiment is **executed and evaluated against a behavioral oracle**. Whenever possible, the official application and physical device serve as the reference implementation. The agent performs the same operation with the original system and the candidate, then compares their observable behavior. In our AirPort work, the oracle compares semantic packet batches. For another device, it may compare application state or a physical outcome. The oracle should normalize irrelevant nondeterminism, preserve semantic differences, and identify the source of a divergence rather than return a single pass-or-fail result.

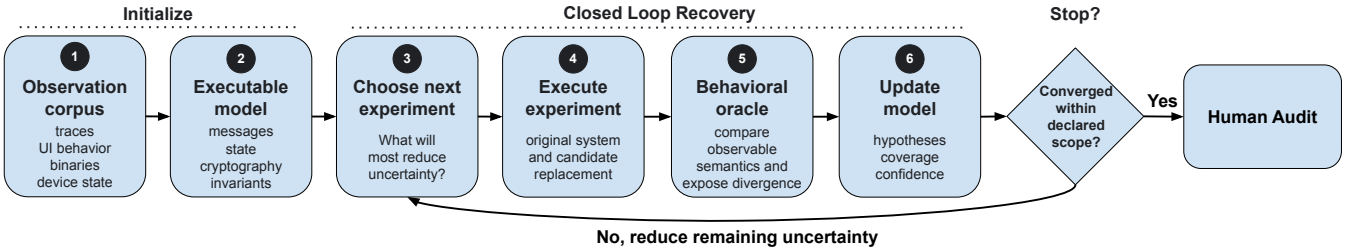

Figure 1: AI-assisted de-cloudification as a closed-loop experimental process.

The experiment then **updates the protocol model and candidate implementation**. Supporting evidence increases confidence in a hypothesis, while a discrepancy may reveal an incorrect field interpretation, missing transition, or version-specific assumption. The system should preserve the provenance of each conclusion and maintain a coverage map that records which workflows have been reproduced and which behaviors remain unexplored. The revised model becomes the starting point for the next iteration.

Determining when the process has **converged** is more difficult than evaluating a single experiment. Matching every collected trace establishes correctness only within the coverage of those traces. A practical system should stop when the candidate matches a coverage-defined test suite, remaining uncertainty is explicit, and further experiments are unlikely to change the supported behaviors. A human can then audit unobserved states and assumptions embedded in the oracle. The result is a scoped claim about recovered behavior, not proof that the entire protocol is correct. Building a system capable of making that claim raises the research challenges we consider next.

5 RESEARCH CHALLENGES

The closed loop in Section 4 turns protocol recovery into a systems problem. Models can generate candidate explanations and implementations. Reliable de-cloudification, however, depends on evidence quality, experiment selection, and the claims attached to the result. We identify seven challenges that must be addressed before this process can operate with limited human supervision.

1. Coverage. Ordinary traces are biased toward common, successful workflows and omit the states most likely to expose an incomplete model. Random protocol fuzzing can expand the corpus, but often produces invalid messages without reaching meaningful behavior. An AI-guided system could instead generate user workflows, vary one condition at a time, and target uncertain regions of its current model; traces from real users could expose behaviors the developer did not anticipate. The open problem is to turn these sources into a useful measure of behavioral coverage. We want to know which user-relevant states and transitions the replacement supports rather than just how many packets or code paths were observed.

2. Behavioral oracles. Every experiment needs a cheap way to determine whether the candidate behaved correctly. A packet-level comparison may suffice for one protocol, while another requires observing application or device state. The oracle must ignore irrelevant nondeterminism without hiding semantic differences. Constructing an oracle manually can require nearly as much work as recovering the protocol itself. A practical system must therefore derive oracles from repeated executions and existing interfaces, and combine partial signals when no single observation suffices.

3. Experiment planning. Given an incomplete model, the agent must choose the next action that resolves a consequential ambiguity rather than simply collecting more data. Possible experiments differ in expected information gain, cost, oracle quality, and dependence on physical hardware. The agent must estimate these quantities from a model that may itself be wrong, then decide whether to gather another trace, inspect the binary, or revise and replay the candidate implementation. This is active learning over a changing, partially observed system rather than ordinary test generation.

4. Faithful test environments. A development environment or emulator is the natural place to run destructive experiments like wiping a disk, but building one is not straightforward when the device semantics are still unknown. A trace-replay server may faithfully imitate observed responses without reproducing the device's internal state changes that make an action destructive; a learned simulator may encode the same blind spots as the current protocol model. The research challenge is to construct a resettable surrogate incrementally, validate which parts of its behavior are faithful, and identify when an uncertainty can only be resolved on real hardware. Such environments are necessary not only for safety, but also for reproducible and parallel experimentation.

5. Efficiency. Experiments may require restarting an application, restoring state, waiting for timeouts, or accessing scarce hardware. A recovery system should reuse evidence

rather than repeat work after every model revision. Deterministic replay and caching can preserve normalized traces, binary-analysis results, and prior oracle outcomes, while independent experiments can run in parallel when they do not share device or cloud state. The deeper challenge is identifying reusable experiment state after each model revision.

6. Confidence and convergence. Passing every collected trace does not establish correctness because the corpus and oracle may both be incomplete. Confidence should therefore attach to individual behaviors and their supporting evidence, not to the protocol as a whole. Holdout traces, adversarial tests, and contradictions across device versions may expose overfitting. Convergence must remain scope-relative, meaning the candidate matches the original over a coverage-defined test suite, its assumptions are explicit, and additional experiments are unlikely to change the supported behavior. A human audit remains appropriate before high-impact use.

7. Responsible recovery. A system intended to preserve owner control could also reproduce proprietary services, extract credentials, or automate unauthorized interaction with third-party devices. The research community must distinguish repair, preservation, and defensive analysis from misuse. One defensible boundary is to focus on owner-authorized devices, produce clean replacement implementations rather than redistribute extracted code, preserve authentication boundaries, and disclose discovered vulnerabilities through established channels. Vendor-provided local interfaces and end-of-life specifications remain preferable; AI-assisted recovery is a fallback when those paths do not exist.

Together, these challenges suggest that the limiting factor in AI-assisted de-cloudification will not be the ability to produce another plausible implementation. It will be the infrastructure needed to gather representative evidence, validate behavior, experiment reproducibly, and communicate the scope of what has been recovered. Independent recovery remains a fallback, while its growing feasibility strengthens the case for vendor-provided control-transfer paths.

6 WHEN SUPPORT ENDS, CONTROL SHOULD TRANSFER

An end-of-life control-transfer principle. We propose a simple rule for preventing non-consensual e-waste: vendors may stop supporting a device, but they should not retain the technical power to make otherwise functional, owner-held hardware unusable. Ending support and preventing owners from supporting a device themselves are distinct choices. For devices whose core functions can operate without a continuing vendor service, the vendor should provide a usable end-of-life path. It can preserve a local operating mode, publish a stable interface, release the information needed to build replacement software and the authority needed to operate it, or provide a mechanism for transferring trust to an owner-authorized controller. AI-assisted recovery should remain a fallback when these paths are unavailable, not a substitute for them. Vendors already possess the semantics and authority that independent investigators must reconstruct.

Consent requires disclosure before purchase. If a device is designed without an end-of-life path, that limitation should be disclosed prominently before sale rather than discovered after the hardware has been deployed. The disclosure should identify which functions require a vendor-operated service, the minimum guaranteed support period, which functions may disappear afterward, and whether the owner will receive a local-control or trust-transfer mechanism. This requirement would not prevent vendors from selling service-dependent products; it would make the vendor-controlled lifetime a term that consumers can evaluate before purchase. Without either an exit path or informed consent, discontinuing support gives the vendor unilateral power to determine the useful lifetime of hardware it no longer owns.

The obligation is scoped. Not every cloud function can be replaced locally as some products genuinely depend on remote computation, shared infrastructure, licensed content, or a continuing human-operated service. The control-transfer principle does not require perpetual free service or the disclosure of shared credentials and third-party secrets. It applies when core functions are performed locally or an owner or third party can reasonably supply a replacement control path without continued vendor cooperation. An end-of-life path can preserve authentication, safety, and privacy through mechanisms such as per-device owner keys, physical-presence requirements, and documented trust rotation. If vendor-held credentials, remote attestation, or a mandatory cloud service still block owner-authorized replacement after protocol recovery, the remaining obstacle is authority rather than missing semantics.

AI changes the policy baseline. Our early experience does not show that de-cloudification is automatic as coverage, validation, and experiment planning remain difficult. It does show that foundation models are reducing the cost of reconstructing missing semantics, while vendors already have the source code, documentation, and authority needed to provide an exit path directly. As independent recovery becomes more practical, a vendor-controlled device lifetime becomes less a technical necessity than a business and design choice. The systems community should build dependable recovery fallbacks and end-of-life trust-transfer mechanisms; vendors, standards bodies, and policymakers should make exit paths the default, with clear pre-sale warnings otherwise.

7 CONCLUSION

Cloud-tethered devices should not become e-waste merely because vendor support ends. Vendors need not operate services forever, but they should provide a local operating mode, release the information and authority required by replacement software, or transfer trust to an owner-authorized controller. Foundation models make independent recovery increasingly practical, but dependable de-cloudification still requires better coverage, behavioral oracles, experiment planning, and confidence estimation. AI-assisted recovery should remain a fallback, not a substitute for deliberate end-of-life design: support may end, but control should transfer.

REFERENCES

[1] Apple, Inc. 2015. *The OS X Accessibility Model.* https://developer.apple.com/library/archive/documentation/Accessibility/Conceptual/AccessibilityMacOSX/OSXAXmodel.html

[2] Apple Inc. 2024. *Download AirPort Utility 5.6.1 for Windows.* https://support.apple.com/en-us/106400

[3] Apple Inc. 2024. *Download AirPort Utility 6.3.1 for Mac.* https://support.apple.com/en-us/106429

[4] Apple Inc. 2026. *AirPort Utility.* https://apps.apple.com/us/app/airport-utility/id427276530

[5] Apple, Inc. n.d. *AirPlay.* https://www.apple.com/airplay/

[6] Belkin. 2026. *Wemo Support Ending - What You Need to Know.* https://www.belkin.com/support-article/?articleNum=335419

[7] Tristan Benoit, Yunru Wang, Moritz Dannehl, and Johannes Kinder. 2025. BLens: contrastive captioning of binary functions using ensemble embedding. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 353, 20 pages.

[8] Ruven Brooks. 1983. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies* 18, 6 (1983), 543–554. https://doi.org/10.1016/S0020-7373(83)80031-5

[9] Kevin Burk, Fabio Pagani, Christopher Kruegel, and Giovanni Vigna. 2022. Decomperson: How Humans Decompile and What We Can Learn From It. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2765–2782. https://www.usenix.org/conference/usenixsecurity22/presentation/burk

[10] James Calligeros. 2026. *Progress Report: Linux 6.19.* https://asahilinux.org/2026/02/progress-report-6-19/

[11] Ying Cao, Runze Zhang, Ruigang Liang, and Kai Chen. 2024. Evaluating the Effectiveness of Decompilers. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 491–502. https://doi.org/10.1145/3650212.3652144

[12] ClassAction.org. 2026. *Belkin Wemo End of Life Arbitration: Affected by Discontinued Belkin Wemo Support?* https://www.classaction.org/belkin-wemo-bricking-arbitration-lawsuit-info

[13] Juli Clover. 2024. *Spotify's Car Thing Accessory is Officially Dead.* https://www.macrumors.com/2024/12/10/spotify-car-thing-officially-dead/

[14] Juli Clover. 2026. *Apple Dropping AirPort Utility From the App Store With iOS 27.* https://www.macrumors.com/2026/06/22/apple-airport-utility-ios-27/

[15] Cristian Curaba, Denis D'Ambrosi, Alessandro Minisini, and Natalia Pérez-Campanero Antolín. 2024. CryptoFormalEval: Integrating Large Language Models and Formal Verification for Automated Cryptographic Protocol Vulnerability Detection. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*. https://openreview.net/forum?id=hqvJT2pOxu

[16] D. Taylor and T. Wu and N. Mavrogiannopoulos and T. Perrin. 2007. *Using the Secure Remote Password (SRP) Protocol for TLS Authentication.* https://datatracker.ietf.org/doc/html/rfc5054

[17] Enrique Dans. 2025. *Samsung's ad-powered fridge: a masterclass in how to alienate your best customers.* https://medium.com/enrique-dans/samsungs-ad-powered-fridge-a-masterclass-in-how-to-alienate-your-best-customers-089d2856aaac Medium, accessed July 16, 2026.

[18] E. Rescorla. 1999. *Diffie-Hellman Key Agreement Method.* https://datatracker.ietf.org/doc/html/rfc2631

[19] eero. n.d. *What is eero Plus?* https://support.eero.com/hc/en-us/articles/115002766063-What-is-eero-Plus

[20] eero. n.d. *What is Wifi Radio Analytics?* https://support.eero.com/hc/en-us/articles/15282384606107-What-is-Wifi-Radio-Analytics

[21] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. 2024. Large language models for code analysis: do LLMs really do their job?. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 47, 18 pages.

[22] Arlo Gilbert. 2016. *The time that Tony Fadell sold me a container of hummus.* https://arlogilbert.com/the-time-that-tony-fadell-sold-me-a-container-of-hummus-cb0941c762c1

[23] Gadget Hacks. 2025. *Apple Locks iPhones into iOS 26 - No Downgrade Possible.* https://apple.gadgethacks.com/news/apple-locks-iphones-into-ios-26-no-downgrade-possible/

[24] Karol Herbst. 2018. Karol Herbst: Nouveau - reverse engineering Nvidia GPUs. DevConf 2018, YouTube. https://www.youtube.com/watch?v=-7SdKBUrKJ0

[25] Alex Hern. 2016. *Revolv devices bricked as Google's Nest shuts down smart home company.* https://www.theguardian.com/technology/2016/apr/05/revolv-devices-bricked-google-nest-smart-home

[26] Peter Holderith. 2022. *BMW Responds to Fury Over Heated Seats Subscription Fee.* https://www.thedrive.com/news/bmw-responds-to-fury-over-heated-seats-subscription-fee

[27] Byron Hurd. 2026. *BMW Commits to Subscriptions Even After Heated Seat Debacle.* https://www.thedrive.com/news/bmw-commits-to-subscriptions-even-after-heated-seat-debacle

[28] Michaela Mihaljević Jakić. 2026. *The SwiftUI Oracle: Measuring a Clean Room Against the Real Thing.* https://aleahim.com/blog/the-swiftui-oracle/

[29] Jiayi Jiang, Xiyuan Zhang, Chengcheng Wan, Haoyi Chen, Haiying Sun, and Ting Su. 2024. BinPRE: Enhancing Field Inference in Binary Analysis Based Protocol Reverse Engineering. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 3689–3703. https://doi.org/10.1145/3658644.3690299

[30] K. Moriarty and B. Kaliski and A. Rusch. 2017. *PKCS #5: Password-Based Cryptography Specification Version 2.1.* https://datatracker.ietf.org/doc/html/rfc8018

[31] Donghyun Kim, Zichao Hu, Joydeep Biswas, Aditya Akella, and Daehyeok Kim. 2026. Mimesys: Generating Realistic Executable Testing Environments from Resource Usage Traces. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. USENIX Association, Seattle, WA, 2205–2221. https://www.usenix.org/conference/osdi26/presentation/kim-donghyun

[32] Boaz Lavon, Shahar Katz, and Lior Wolf. 2025. Execution Guided Line-by-Line Code Generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=ySFDPoiANu

[33] Boaz Lavon, Shahar Katz, and Lior Wolf. 2025. Execution Guided Line-by-Line Code Generation. arXiv:2506.10948 [cs.LG] https://arxiv.org/abs/2506.10948

[34] Linux Wireless Developers. n.d. *Broadcom brcmsmac (PCIe) and brcmfmac (SDIO/USB) Drivers*. https://wireless.docs.kernel.org/en/latest/en/users/drivers/brcm80211.html

[35] man7.org. 2026. *tcpdump(1) — Linux manual page*. https://man7.org/linux/man-pages/man1/tcpdump.1.html

[36] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. 2022. RE-Mind: a First Look Inside the Mind of a Reverse Engineer. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2727–2745. https://www.usenix.org/conference/usenixsecurity22/presentation/mantovani

[37] Dylan Manuel, Nafis Tanveer Islam, Joseph Khoury, Ana Nunez, Elias Bou-Harb, and Peyman Najafirad. 2024. Enhancing Reverse Engineering: Investigating and Benchmarking Large Language Models for Vulnerability Analysis in Decompiled Binaries. arXiv:2411.04981 [cs.CR] https://arxiv.org/abs/2411.04981

[38] National Security Agency. 2026. *ghidra*. https://github.com/nationalsecurityagency/ghidra

[39] OpenAI. 2026. *Introducing GPT-5.5*. https://openai.com/index/introducing-gpt-5-5/

[40] OpenAI. n.d. *Codex*. https://openai.com/codex/

[41] Thiyagu Palanisamy and Chandirasekar Thiagarajan. 2025. *Blackbox reverse engineering: Rebuilding without source code — XConf India 2025*. https://www.youtube.com/watch?v=Vl1yHlBuAlk

[42] Sasha Rogelberg. 2025. *Samsung confirms it will begin showing you advertisements on your $1,800-plus refrigerator's screen*. https://fortune.com/2025/09/19/samsung-family-hub-refrigerators-advertisements/

[43] The Apple Wiki Contributors. 2026. *SHSH*. https://theapplewiki.com/wiki/SHSH

[44] The LLDB Team. 2026. *The LLDB Debugger*. https://lldb.llvm.org/

[45] Tom Wu. n.d. *SRP Protocol Design*. http://srp.stanford.edu/design.html

[46] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek. 2020. An Observational Investigation of Reverse Engineers' Processes. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1875–1892. https://www.usenix.org/conference/usenixsecurity20/presentation/votipka-observational

[47] Anjiang Wei, Tarun Suresh, Jiannan Cao, Naveen Kannan, Yuheng Wu, Kai Yan, Thiago S. F. X. Teixeira, Ke Wang, and Alex Aiken. 2025. CodeARC: Benchmarking Reasoning Capabilities of LLM Agents for Inductive Program Synthesis. arXiv:2503.23145 [cs.PL] https://arxiv.org/abs/2503.23145

[48] Haiyang Wei, Ligeng Chen, Zhengjie Du, Yuhan Wu, Haohui Huang, Yue Liu, Guang Cheng, Fengyuan Xu, Linzhang Wang, and Bing Mao. 2025. Unleashing the Power of LLM to Infer State Machine From the Protocol Implementation. In *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS65803.2025.11143461

[49] x56 (Vince Cali). 2017. AirPyrt Tools. https://github.com/x56/airpyrt-tools.