ThoxOS and ThoxInux: Architecture, Evidence, and a Validation Protocol for Local-First AI

Craig Ross* Tommy Xaypanya†
THOX.ai LLC

November 1, 2026
Technical research draft, revision 0.2.0-review

Submission readiness is not established. No arXiv submission or external peer review is claimed.

Abstract

Local AI systems require an evidence chain from kernel behavior to application execution, model quality, and physical device operation. This technical report examines the ThoxOS program with ThoxInux, its Ginux-derived, Rust-first x86-64 foundation. We reconcile an earlier Linux-based design with current source and release records, distinguish reusable component evidence from integrated-system evidence, and define a falsifiable validation protocol. The reviewed records document engineering-preview EFI releases and bounded virtual-machine lifecycle tests. They also record resident-agent faults in an edition payload, conditional build reproducibility, and incomplete Linux ABI behavior. Published language-model evaluations carry additional restrictions on scoring scope, serving templates, quantization, and hardware applicability. We formalize the separation between service entitlement and action authorization, and specify experiments for workload compatibility, agent lifecycle, offline operation, update recovery, and device feasibility. The review contributes a revision-bound evidence map and an explicit set of integration tests; it does not introduce a new kernel, report a performance advantage, or establish production security. Historical unit-test counts, repository-reported runs, publisher-reported model scores, and unexecuted protocols remain distinct. The resulting architecture draft supports continued implementation while identifying the measurements required for a complete empirical systems paper.

1 Introduction

An operating system for local AI must support more than model-file availability. The kernel must sustain the workload's process, memory, filesystem, and networking behavior; the runtime must load the exact model artifact; the application must enforce the user's authority; and the resulting system must operate within the selected hardware envelope. Evidence at any one layer cannot establish the other layers.

This report asks: *what is established about the current ThoxOS foundation, and what experiments are necessary to establish a bounded local-AI deployment?* ThoxOS denotes the device-intelligence platform, ThoxInux the Ginux-derived kernel and operating-system foundation, MeshStack the optional private mesh platform, and ThoxWork the workspace. These roles describe intended integration boundaries. They do not establish that the complete stack has executed on the reviewed kernel.

*ORCID: 0009-0005-9872-3673.
†ORCID: 0009-0000-4425-9565.

The source history requires an explicit distinction. An October 31 design whitepaper describes a Linux LTS fork, offline entitlement, Buildroot image definitions, and a separate Rust research laboratory. The current ThoxInux repository instead identifies Ginux as its foundation and excludes MeshStack, ThoxWork, and ThoxKey firmware from its open-source license. We treat the earlier design as historical input, not as authority to restate current kernel or portfolio licensing decisions [1, 2, 3].

The contribution is threefold: a source- and payload-specific evidence map; a separation of commercial entitlement, execution authorization, and model behavior; and a reproducible protocol for testing the remaining integration hypotheses. No novelty claim is made for the upstream kernel, the cryptographic primitives, or standard update mechanisms. This is an architecture and validation-status draft, not a completed empirical evaluation.

2 Related Work

Ginux is the direct implementation antecedent: an independent Rust-first x86-64 kernel with Linux userspace compatibility, a ring-3 administration controller, and a UEFI boot image. The upstream project describes an experimental system with a constrained QEMU hardware target. Its technical-report checkpoint differs from the upstream revision pinned by ThoxInux; results cannot be transferred between those revisions without new evidence [4, 5]. We use the upstream repository for architecture context and do not reproduce the technical report's unreviewed performance results.

ASTERINAS studies a different Rust-based framekernel architecture with the OSTD framework and a small memory-safety trusted computing base [6]. It establishes relevant precedent for combining Rust with Linux ABI compatibility, but neither its safety argument nor its measured performance applies automatically to Ginux or ThoxInux. It is retained as a foundational 2025 source rather than presented as a new result within the six-month review window.

RAUC documents bootloader integration and the application decisions needed to mark an updated system healthy [7]. It supplies an implementation reference for update experiments; an A/B configuration template alone does not establish successful recovery. HMAC supplies symmetric message authentication [8]. It is useful for a controlled entitlement-state demonstration, but a device that holds the MAC verification secret can also generate MACs. A production verification architecture must state its separate key-custody and enforcement assumptions.

The current primary context includes the October 2026 Ginux work, current ThoxInux source and release evidence, and the November 2026 THOX model cards. This is a focused source review, not an exhaustive systematic literature review. Competing systems are compared only on mechanisms relevant to this report; no comparative speed, market coverage, or security ranking is inferred.

3 Methods

3.1 Source boundaries and evidence classes

The review cutoff is November 1, 2026 in America/Chicago. Table 1 records the principal repository revisions. The accompanying evidence ledger retains full identifiers, file hashes, model-card revisions, access dates, and locators. Four kinds of evidence are kept separate:

1. **Observed artifacts:** repository metadata, manuscript files, release entries, and file metadata actually inspected during this review.
2. **Repository-reported execution:** historical tests and experiments documented in a named source record. These were not rerun here.
3. **Publisher-reported evaluation:** model-card scores under the publisher's stated data and serving conditions. These were not independently reproduced here.

4. **Proposed protocol:** measurements and acceptance criteria specified for a subsequent authorized run. No outcome is implied.

No OS boot, model inference, physical device test, or security penetration test was executed for this manuscript review. Source inspection and a successful document build are not system validation.

Table 1: Reviewed source identities. Full source links and hashes are supplied with the manuscript.

Source	Immutable revision and role
ThoxInux	`fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9`; current reviewed foundation [2].
Ginux fork base	`36503bda2d87465765e1b94e65f9d100f27ed5be`; pinned by ThoxInux, not current upstream main [5].
Earlier ThoxOS Core	`887b8c9c47f352a577b4592c6f81a2c1570d9af3`; historical Linux/Core component lane [9].
Paper repository baseline	`ad92ec44116e4dd895f1f82f7b8c279de1241401`; three placeholder files in the paper directory [10].

3.2 Architecture and deployment scope

ThoxInux's open foundation derives from Ginux under its recorded Apache-2.0 terms and preserved upstream attribution. Its source record identifies a unified patch series, startup changes, THOX identity, QEMU tooling, and a registry trust anchor [5]. Calling the fork hardened is outside this review's evidence. The rights ledger excludes proprietary portfolio implementations; this manuscript does not expand their license [3].

The currently reviewed kernel target is x86-64. Running an x86-64 guest through QEMU on an Arm host is emulation, not an Arm kernel port. The ThoxAir E2, ThoxMini P4, and ThoxClip P6 program classes specify nominal memory floors of 2, 4, and 6 GB respectively [1]; they are device targets, not demonstrated Ginux deployment platforms. The repository's Arm scoping record identifies architecture and ABI work still required [11]. ThoxKey model packages refer to a connected-host workflow, not demonstrated inference inside the USB device [12]. Any deployment matrix must name the actual board, processor architecture, kernel, runtime, model artifact, and execution location.

Licensing and pricing are different questions. Open kernel source does not establish that every local portfolio service is free or open source. Conversely, a proprietary optional integration does not determine the kernel's source license. The older hosted-services-only commercial design is retained as a dated proposal where discussed, rather than presented as a resolved license for MeshStack or ThoxWork.

3.3 Entitlement and execution authority

The historical Core proposal includes an invariant that commercial entitlement must not disable features explicitly designated Core. Let F_{core} be the declared Core feature set, t any token state, and $E(f, t)$ the commercial-entitlement decision. The proposed invariant is

$$f \in F_{\text{core}} \implies E(f, t) = \text{true} \quad \text{for every } t. \tag{1}$$

This is an availability contract, not permission to perform arbitrary actions. For action a, the complete decision must retain independent controls:

$$\text{MayExecute}(a) = U(a) \wedge O(a) \wedge E(f(a), t), \tag{2}$$

where U is current user authorization, O is operating-system and application access control, and $f(a)$ maps the action to a declared service feature. Model output cannot set U or bypass O. Unknown feature identifiers must follow an explicit denial policy; they are not implicitly Core.

The supplied reference uses a demo HMAC state machine for valid, grace, expired, and invalid token states. Its own production discussion calls for pinned public-key verification and separate server enforcement [1, 8]. This draft does not claim that production key issuance, rotation, revocation, trusted time, or billing is implemented. Offline availability, entitlement revocation, and clock rollback must be examined under a stated threat model; a fixed grace window alone resolves none of those issues.

3.4 Evaluation protocol

The following experiments are proposed. Each run must freeze code, EFI and image hashes, toolchain, machine profile, runtime, model weights, tokenizer, prompt template, quantization, context length, decoding, and workload manifest before execution. Report every attempted run and exclusion.

P1: Workload and resident-agent compatibility. Use the same x86-64 machine or controlled VM resources for a supported Linux baseline and the candidate Ginux-derived payload. Start with the failing resident-daemon lifecycle workload, retain the minimal reproducer, and collect serial output, process lifecycle events, audit records, and exit status. Separately test model loading, file mappings, thread creation, signals, sockets, and repeated inference. Acceptance requires the declared lifecycle and bounded workload to finish without kernel/domain faults or unexplained failures. A successful boot alone fails this criterion. Proposed repetition: at least 30 cold-start lifecycle runs per configuration, with run order recorded. This is a study design choice, not a safety certification or sufficient evidence for rare-failure rates.

P2: Model and application behavior. Use a held-out workload separated from training and development gates. Record both the full generated completion and the final user-visible answer, with the scoring rule fixed in advance. Evaluate the actual deployment artifact, including its quantization and serving template. Report correct answers, abstentions, unsafe action proposals, unauthorized tool executions, and task completion separately. Include the application harness and denial path in end-to-end tests. A zero in a small observed failure set is not a population guarantee.

P3: Offline and entitlement behavior. From a clean installation, exercise the declared Core feature set with no account and with absent, malformed, expired, and tampered tokens. Include clock rollback, reboot, and service outage. Record outbound connection attempts at an independently controlled network boundary. Evaluate Equation 1 while retaining Equation 2; test invalid action authorization as well as valid Core use. Report which features and network interfaces were in scope instead of claiming universal privacy.

P4: Updates and recovery. For a built image with a named bootloader and RAUC integration, inject invalid signatures, incomplete writes, exhausted boot attempts, failed health checks, and power interruption at declared stages. Capture slot selection, health marking, rollback decisions, and user-data integrity [7]. Acceptance requires recovery to the named known-good state within a predeclared boot budget. Template validation, unit tests, and a different distribution's rollback behavior do not satisfy this test.

P5: Hardware feasibility and runtime cost. After architectural support exists, run the exact workload on each named board. Measure peak resident memory and total available

memory, time to first token at the initiating client, generated-token throughput, energy per completed task if instrumentation exists, temperature, and failure rate. Report cold and warm loading separately. For mesh execution, state where each operation ran and how many bytes crossed each link. Host or peer execution must not be labeled on-device inference.

4 Results

4.1 Foundation evidence and negative findings

Table 2 summarizes *repository-reported* evidence, not experiments conducted for this paper. The exact payload identity matters because the unified EFI, the edition demo image, and a host container are different execution environments [13, 14].

Table 2: Documented results and their limits at the reviewed revision.

Artifact or test	Recorded result	What the result does not establish
Unified EFI v0.9.0	Public engineering prerelease. Payload U has 22/22 lifecycle checks reported under QEMU TCG and 22/22 under KVM [15, 13].	Physical device qualification, full Linux conformance, model inference, or complete AgentOS integration. The two accelerator runs exercise the same bounded suite.
Controlled EFI v0.9.1	Public engineering prerelease. Matching builds are reported with a fixed epoch, registry anchor, and absolute build path [16, 17].	Unconditional reproducibility or a new lifecycle result. The record explicitly says these release bytes were not separately lifecycle-run.
Agent supervisor in a host container	Identity, capability-denial, audit, and shutdown behavior are reported in the container experiment [13].	Enforcement on the Ginux kernel. The container shares its host kernel.
Resident AgentOS on the legacy edition Ginux guest	The guest daemon starts, but domain/kernel faults prevent completion of the intended loop across 17 recorded attempts [18, 13].	A working persistent agent loop. The failure was not reproduced here on unified Payload U or v0.9.1; their complete resident-loop behavior remains unestablished. The source authors' root-cause attribution is not independently confirmed.
Earlier Linux/Core components	The historical lane reports 25 entitlement tests and 27 image-definition checks [9, 1].	A current ThoxInux system test, booted RAUC recovery, production entitlement security, or newly reproduced evidence.

The compatibility audit reports 375 dispatched syscall numbers at the fork base: 82 labeled Complete, 245 Partial, and 48 Policy-denied [19]. These are ledger classifications checked for source/document consistency. They are not 375 independently executed conformance tests, and the Complete count is not an application-compatibility percentage. The audit identifies

incomplete behavior in file-backed `MAP_SHARED`, connection/acceptance paths, and blocking waits. Those surfaces motivate P1 rather than support a blanket Linux compatibility claim.

The release records also preserve failures: reproducibility initially depended on uncontrolled build inputs; a later path change altered debug-identifier bytes; and the recorded edition resident-agent workload remains blocked. Its EFI SHA-256 is `21d12d86631df7947a9bd7389b66410667 6c6f97a18e44d38d9ddb906b00f7b4`, distinct from unified Payload U and v0.9.1. A failure in that lineage does not prove failure on the unified releases, whose complete resident loop remains unestablished. These findings must remain adjacent to the positive results. Neither a nearly identical EFI hash nor the same source family licenses transfer of an execution result to untested bytes.

4.2 Published model evidence

Table 3 records the exact model-card interpretation needed for this architecture. All gate scores below are publisher-reported development-set results. They are not independent reproduction, held-out generalization estimates, or physical-device measurements. The word "Kernel" in the model family name does not mean that the models run in kernel space.

Table 3: Model-card results with the conditions that limit their use.

Model	Reported GATE49 outcome	Material limits
ThoxOS-Kernel-27B	41/49; PASS under adopted final-answer scope.	The raw-completion verdict is HELD with four known-false hard failures. Final-answer and full-completion results answer different questions. The model is a host-tier artifact [20].
ThoxOS-Kernel-4B	44/49; documented baseline, not gate-passed.	The card reports a GATE-K hard failure, a P17 escape, and an OS-bench destructive escape. Gate scores are on merged safetensors, not proof for quantized GGUF or a physical device [21].
ThoxOS-Kernel-0.8B	37/49; PASS on the specified gates with zero hard failures.	GATE-K verified recall is 0/12 and the routing subscore is 0/2. The stock thinking-template UX test produced a final answer for only 1/38 prompts; the published think-suppressed configuration is required. Gates use merged safetensors; ThoxAir qualification remains absent [12].

The Q4_K_M artifact metadata reports 16,547,400,192 bytes for 27B, 2,708,804,128 bytes for 4B, and 529,297,056 bytes for 0.8B [20, 21, 12]. File sizes were checked as metadata; this review did not download and hash the full weights. File size alone cannot establish runtime fit. A necessary memory budget is

$$M_{\text{weights,resident}} + M_{\text{KV}} + M_{\text{runtime}} + M_{\text{OS}} + M_{\text{reserve}} \leq M_{\text{available}}. \tag{3}$$

Here $M_{\text{available}}$ is total usable physical RAM before the OS and runtime allocations on the left; it is not the operating system's `MemAvailable` counter. All terms must be measured or bounded for the chosen workload. Storage bytes, advertised memory, and peak resident memory are different quantities. Quantization and a smaller parameter count do not eliminate the need to measure model quality and end-to-end safety again.

5 Ablations

No ablation experiment was executed for this manuscript. The following controlled comparisons are proposed to isolate mechanisms rather than aggregate unrelated evidence:

1. **Kernel and workload:** Linux versus the candidate ThoxInux payload under matched VM resources and the same userspace workload. Hold the runtime and model constant where compatibility permits; report any required change as a confounder.
2. **Resident lifecycle:** the minimal agent loop without and with the resident daemon/state-writing workload. Preserve the failure reproducer and change one condition at a time. Do not disable a security guard merely to obtain a passing result.
3. **Model surface:** merged weights versus the selected quantization, and published template versus stock template. Report both full-completion and final-answer scoring under fixed rules; do not change verdict scope after seeing the outcome.
4. **Placement and loading:** cold versus warm local loading and, after integration exists, local versus authorized peer execution. Record transfer time, queue time, and initiating-client latency separately.

The design should report medians and tail latency with sample counts and uncertainty suitable for the available observations. Small samples must not be presented as precise extreme-tail estimates. Failure and abstention counts remain visible even when a primary quality threshold is met.

6 Limitations

The strongest current evidence is a set of bounded, separately identified engineering records. It is not a single integrated certification. Repository and model-card reports may contain stale wording or conflicting summaries; this draft pins the inspected sources and records material differences rather than resolving them by assertion. Full raw historical evidence is not present in the supplied four-file ZIP. Public records recovered during review improve traceability but do not substitute for independent reruns.

Ginux behavior on the named x86-64 VM profile does not establish Arm support, GPU or storage-driver coverage, physical reliability, or deployment readiness across the THOX device family. The edition resident-agent failure and the absence of a completed unified-payload loop limit an always-on system claim in different ways. No inference performance advantage, energy benefit, end-to-end offline guarantee, production security assessment, or device-qualified model result is established here. The proposed experimental protocol still requires implementation and execution.

This draft is also not a complete arXiv research contribution merely because its files can be compiled. A subsequent manuscript must present a sufficiently developed contribution and supporting analysis or measurements, then undergo exact-version author and submission review. The current status remains an internally reviewed technical draft.

7 Safety

The threat model includes untrusted prompts, malicious tool input, compromised applications, malformed artifacts, unauthorized actions, and tampered update or entitlement data. The proposed trusted boundary consists of an authenticated user decision, an enforcing application/tool broker, operating-system access controls, and verified artifact identities. A language model proposes actions; it is not the authority that approves them. Sensitive operations require policy enforcement outside the model, including a valid authorization context and correctly scoped capabilities.

For the historical entitlement prototype, shipping a shared HMAC issuer secret to clients would defeat an issuer-only trust assumption [8]. Production design must keep issuer secrets outside untrusted clients and specify signature validation, key rotation, replay and clock behavior, and server enforcement where needed. These controls are requirements here, not implemented security claims.

Kernel memory safety, isolation, update integrity, and application authorization are distinct properties. Rust source, capability labels, and a successful denial test do not by themselves prove all of them. Fault-injection and security tests should use disposable environments; destructive physical writes, deployment changes, and spending retain their separate authorization requirements. No health, legal, financial, or regulatory compliance certification is claimed.

8 Ethics and Research Integrity

Upstream authorship and licensing remain visible. ThoxInux is identified as a derivative of Gregory M. Kurtzer and contributors' Ginux work, not a clean-room implementation created by THOX. The historical Rust Kernel Lab specification is a separate research lane; its process intentions are not proof about the provenance or security of a different kernel.

Published scores are accompanied by failure cases and evaluation conditions. A development gate is not an independent benchmark, and a benign-looking final answer does not erase unsafe content elsewhere in a completion. This distinction is especially important when models influence system actions. The review uses no human-subject study and introduces no private user dataset. Human author names and ORCID identifiers are supplied by the project owner; exact-version authorship consent and final publication responsibility require author review before submission.

9 Reproducibility Checklist

Table 4: Reproducibility status for this draft, not a declaration of submission readiness.

Item	Status and required evidence
Editable manuscript and bibliography	Included. Build with the documented PDFLaTeX/BibTeX procedure without shell escape.
Source and input identity	Included in the accompanying ledger and hash manifest; historical inputs remain separately identified.
Reported system runs	Public source records and payload identities cited; no independent OS rerun in this review.
Model-card identity	Exact card revisions recorded. Full weight hashes and deployment evaluations were not independently reproduced.
Hardware, runtime, and workload	Required for P1–P5; no new physical-device results in this manuscript.
Metrics, baselines, and ablations	Protocol specified; collection and analysis remain pending.
Negative results	Resident-agent failures, reproducibility conditions, model failures, and missing qualifications retained.
Rights and attribution	Historical whitepaper attribution and CC BY 4.0 notice preserved separately; upstream and proprietary boundaries retained. No new portfolio license is granted.

Item	Status and required evidence
Internal review	Claims, source authority, compilation, and every-page PDF review recorded in the companion review packet. Internal AI review is not external peer review.
Public submission	Not performed. Author approval, contribution sufficiency, exact source inventory, current account eligibility, and final destination/license review remain separate.

Conclusion

ThoxInux provides a concrete Ginux-derived foundation for investigating a local-AI operating system. The available evidence supports bounded engineering-preview statements. It identifies an unresolved resident-agent failure in the edition lineage and leaves the complete loop on unified releases unestablished. A credible next result is a reproducible workload study on exact payload bytes, with model/harness behavior and hardware scope measured separately. This report supplies that study's evidence boundaries and protocol without treating historical component checks as proof of the complete platform.

AI Assistance and Attribution

This revision was prepared with AI assistance from Sadie Weaving and parallel source-authority, claims, and artifact-review lanes. It reviews a supplied Grok draft and a historical THOX whitepaper attributed there to Sadie Weaving as a synthetic drafter. Synthetic contributors are acknowledged as AI assistance, not listed as human authors. The requested human byline is Craig Ross, Chief Executive Officer, and Tommy Xaypanya, Chief Technology Officer and Co-Founder, THOX.ai LLC. No unobserved contribution, external review, or author approval is asserted. The original source attribution and license are preserved in the accompanying provenance record; this revision changes the kernel framing, evidence interpretation, protocol, references, and presentation.

References

[1] THOX.ai. ThoxOS: An open kernel and operating system for local-first AI. `https://github.com/Thox-ai/thoxos-core/blob/887b8c9c47f352a577b4592c6f81a2c1570d9af3/docs/whitepaper/ThoxOS_Whitepaper_v0.1.0.md`, 2026. Historical technical whitepaper v0.1.0, October 31. User-supplied reference source; synthetic drafting credited to Sadie Weaving; accountable human author Tommy Xaypanya. CC BY 4.0. Preserved with a provenance and limitations record.

[2] THOX.ai. ThoxInux: source repository and engineering-preview scope. `https://github.com/Thox-ai/thoxinux/tree/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9`, 2026. Inspected November 1, 2026. Source statements are not independent execution evidence.

[3] THOX.ai. ThoxInux rights and license ledger. `https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/RIGHTS.md`, 2026.

[4] Gregory M. Kurtzer and Ginux contributors. Ginux: independent kernel and Linux userspace compatibility. `https://github.com/gmkurtzer/ginux`, 2026. Repository README inspected November 1, 2026; blob

69b765d2aa1355f38a6e279202b6abb0c33ae45a. Its linked report, DOI 10.5281/zenodo.23682548, uses experimental checkpoint d11c126a048a51fe7eb9c4105f31760add1027ea; report performance results are not reproduced here.

[5] THOX.ai. ThoxInux upstream pin and fork discipline. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/UPSTREAM.md, 2026.

[6] Yuke Peng, Hongliang Tian, Junyang Zhang, Ruihan Li, Chengjun Chen, Jianfeng Jiang, Jinyi Xian, Xiaolin Wang, Chenren Xu, Diyu Zhou, Yingwei Luo, Shoumeng Yan, and Yinqian Zhang. ASTERINAS: A Linux ABI-compatible, Rust-based framekernel OS with a small and sound TCB. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pages 307–323. USENIX Association, 2025. https://www.usenix.org/conference/atc25/presentation/peng-yuke.

[7] RAUC developers. RAUC integration documentation. https://rauc.readthedocs.io/en/latest/integration.html, 2026. Bootloader integration, system health, mark-good, and watchdog guidance; accessed November 1, 2026.

[8] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, RFC Editor, 1997. https://www.rfc-editor.org/rfc/rfc2104.

[9] THOX.ai. ThoxOS Core historical component release. https://github.com/Thox-ai/thoxos-core/tree/887b8c9c47f352a577b4592c6f81a2c1570d9af3, 2026. README and release manifest. Earlier Linux/Core lane; not current integrated ThoxInux test evidence.

[10] THOX.ai. Research-paper repository baseline. https://github.com/ttracx/thox-research-paper/tree/ad92ec44116e4dd895f1f82f7b8c279de1241401/papers/thoxos, 2026.

[11] THOX.ai. ThoxInux ARM64 scoping. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/docs/ARM64-SCOPING.md, 2026.

[12] THOX.ai. ThoxOS-Kernel-0.8B model card. https://huggingface.co/Thox-ai/ThoxOS-Kernel-0.8B/blob/fe4520be896b0a1ef6545998777f5a1bdba13072/README.md, 2026. Evaluation and serving-template requirements; publisher-reported development-set results.

[13] THOX.ai. ThoxInux wave ledger. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/docs/WAVES.md, 2026. Waves 1, 3, and 5. Repository-reported runs and failure analysis; not independently rerun.

[14] THOX.ai. ThoxInux payload and patch-set reconciliation. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/docs/RECONCILIATION-2026-10-31.md, 2026.

[15] THOX.ai. ThoxInux v0.9.0 engineering prerelease. https://github.com/Thox-ai/thoxinux/releases/tag/v0.9.0, 2026. Tag resolves to 5ec049bd7911ee5d0eb2e280555212822b7b9cd1. Payload U SHA-256 85722bd41426491bf86f810bbdd25f951c53afc49e1bf32ad5b34c63ab4420ae.

[16] THOX.ai. ThoxInux v0.9.1 engineering prerelease. https://github.com/Thox-ai/thoxinux/releases/tag/v0.9.1, 2026. Tag resolves to 84b4a7e740496297ca79dff735834ca2a8f2bad3. Controlled-build SHA-256 1bc1c7af33470edc009fa5756d3fe6063573660e07e157d19397afc680a7bb89.

[17] THOX.ai. ThoxInux release-build procedure. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/kernel/ginux/docs/RELEASE-BUILD.md, 2026.

[18] THOX.ai. Wave 5 resident-agent guest integration results. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/evidence/wave5-lane-e/RESULTS.md, 2026. The named demo-guest payload differs from the unified release payload.

[19] THOX.ai. Linux compatibility audit at the pinned upstream commit. https://github.com/Thox-ai/thoxinux/blob/fbad5952f20281ef6ad7eaefc1f54f7db4c0c6e9/docs/COMPATIBILITY.md, 2026. Ledger/dispatcher consistency audit, not individual syscall behavioral retesting.

[20] THOX.ai. ThoxOS-Kernel-27B model card. https://huggingface.co/Thox-ai/ThoxOS-Kernel-27B/blob/0e03b1f067e065597e5ef205e01d965b9819b999/README.md, 2026. Evaluation, known limitations, artifact inventory; publisher-reported development-set results.

[21] THOX.ai. ThoxOS-Kernel-4B model card. https://huggingface.co/Thox-ai/ThoxOS-Kernel-4B/blob/5956548e6bcafeae55e58788a0b318b27ddb9223/README.md, 2026. Documented baseline, not gate-passed; publisher-reported development-set results.