ArmOS

An Operating System Architecture for Sovereign, Secure, and AI-Native Computing

TECHNOLOGY WHITEPAPER

A Windows-compatible, privacy-first, AI-native and post-quantum-ready operating system built on an open, hardened Linux foundation — one that returns control of hardware, applications, data, computation, identity, and network participation to the people and organizations that own the devices.

Version 1.0 (Public) · 2026

Aftab Ahmad
ARM Technology Holding Pty Ltd — ArmOS Program
Public technology whitepaper — for general distribution

Contents

About this document

This is a technology architecture whitepaper. It describes what ArmOS is, how it is engineered, what it is designed to achieve, which capabilities are currently targeted for the first release, and which remain on the roadmap or under evaluation. It is written to be understood by executives without being superficial for engineers.

Because ArmOS is an actively developing platform, the paper is careful to separate different states of maturity. To keep that distinction visible, capability status is signalled explicitly where it matters, using the following conventions:

- **Architecture / design intent** — a deliberate design decision, whether or not the corresponding code is complete.
- **MVP target** — a capability planned for the first public release.
- **Roadmap** — a capability planned for a later release.
- **Under evaluation / experimental** — a component or claim being assessed, not yet committed.

Where a figure, benchmark, or comparison originates in internal specification rather than independent validation, the paper says so. No adoption numbers, certifications, audits, partnerships, or performance results are asserted in this document, because none have yet been independently established. The aim throughout is credibility over hype: an ambitious program described honestly.

ArmOS is a technology initiative of ARM Technology Holding Pty Ltd (ACN 161 065 952), part of the ARM group of companies.

Executive Summary

For three decades the personal computer has been defined by a small number of proprietary platforms. That model delivered remarkable capability, but it has also concentrated control: over which software may run, over where data is stored, over how devices are identified and tracked, and over the recurring cost of keeping a fleet operational. Recent public court records have made the consequences concrete. According to public reporting on a 2026 United States federal filing, the dominant desktop operating system assigns each installation a persistent, installation-level device identifier — reported to remain consistent across VPNs and to reset only on reinstallation — present on ordinary systems by default. Computing has quietly become something that observes its users.

ArmOS is an operating system architected to reverse that trajectory. It is built on the mainline Linux kernel and a hardened Ubuntu Long-Term-Support base, and it is engineered so that existing Windows applications, .NET software, and PowerShell workflows continue to run — through mature, clean-room open-source technologies such as Wine, Proton, and Mono, and, where an exact Windows environment is required, through full virtualization. ArmOS contains no Microsoft code and is not a fork of Windows. Its purpose is to let users adopt an open, private platform without giving up the software they depend on.

Four architectural commitments distinguish ArmOS from mainstream desktop platforms. **Privacy by architecture:** the system is designed without a persistent device identifier or default telemetry, so privacy is intended to be a property of the platform rather than a setting to be discovered. **Security by default:** a benchmark-hardened base, a consolidated endpoint-security stack, and a single-toggle hardened profile are built in, not assembled from separate products. **AI-native operation:** a local, offline-capable assistant is part of the operating system, with optional distributed inference across a user's own devices, so intelligence does not require surrendering data to the cloud. **Digital and hardware sovereignty:** peer-to-peer local groups, a portable trace-free mode, and an optional device-owned economy return control of computation and network participation to the device owner.

These commitments are realized through a clean six-layer architecture: the Linux kernel; a hardened Ubuntu base; the ArmOS system-services layer that constitutes the project's primary engineering contribution; a virtualization layer capable of hosting Windows Server guests; a Windows-compatibility runtime; and a Windows-style desktop shell. The same base is designed to extend to a companion mobile experience, so that desktop and mobile converge rather than diverge.

For enterprises, the implications are direct. Endpoint security that normally requires several commercial agents is consolidated into one integrated stack with one policy engine, reducing agent sprawl and operational complexity. Local AI keeps sensitive prompts and documents on the machine, which matters in regulated sectors. A hardened, auditable base with kernel-level logging and cryptographic file-integrity verification provides a defensible foundation for compliance work. By design, post-quantum algorithms are applied to key exchange, signatures, and transport, while data at rest is protected by strong symmetric encryption — treating quantum resistance as a present requirement rather than a future migration.

This paper does not claim that every capability described is complete. Windows compatibility breadth depends on the application; several AI and cryptographic components are targeted or under evaluation; and the roadmap is ambitious. What the paper argues is that the architecture, the engineering model, the security philosophy, and the roadmap are coherent enough to take seriously — and that the problem ArmOS addresses is real, documented, and growing. The remainder of the document sets out that architecture in detail,

distinguishing throughout between what is designed, what is targeted for first release, and what remains ahead.

1. The Computing Problem

The case for ArmOS is not ideological. It rests on a set of documented, mutually reinforcing pressures in mainstream computing. Each erodes user control in a different way; together they describe a platform model increasingly misaligned with the interests of the people and organizations who own the hardware.

1.1 Platform dependency

Modern productivity is tightly coupled to a handful of proprietary operating systems and the ecosystems built around them — identity, app distribution, device management, and cloud services. This coupling is convenient until an organization wishes to change course, at which point proprietary formats, closed management planes, and hardware-attestation requirements make departure expensive. Dependency of this kind is a strategic risk, not a technical footnote.

1.2 Data sovereignty

Documents, identity, and configuration increasingly reside on infrastructure that the user does not control. For individuals this means functionality that degrades when connectivity is lost; for enterprises and public-sector bodies it raises acute questions about jurisdiction, residency, and lawful access. The location and control of data have become a first-order governance concern rather than an implementation detail.

1.3 Privacy and device identity

Operating systems now participate directly in telemetry, identity, and synchronization. The clearest public illustration emerged in 2026, when a United States federal complaint described how investigators traced a suspect across countries not through the network — which had been concealed with a VPN and proxies — but through a persistent operating-system-level device identifier.[1] Reporting on the unsealed filing established that the identifier is tied to the Windows installation, remained consistent despite VPN use, and operates above the network layer; only reinstalling the operating system assigns a new one. The mechanism is present on ordinary installations by default. Whatever one concludes about the specific investigation, the architectural fact is significant: the platform itself can make a device persistently identifiable.

1.4 Security fragmentation

To protect endpoints, organizations typically assemble several commercial products — one for application control, one for detection and response, one for patch and device management, one for compliance evidence, one for privileged access. Each brings its own agent, console, and policy model. The result is agent sprawl, integration overhead, and seams between products that adversaries exploit. Security becomes an assembly problem rather than a property of the platform.

1.5 Cloud-dependent artificial intelligence

Most contemporary AI experiences send user data to remote infrastructure for inference. This introduces latency, ongoing cost, availability dependence, and — most consequentially for regulated organizations — a data-governance exposure, because sensitive content must leave the device to be useful. AI has largely been delivered as a service that observes, rather than a capability that the device owns.

[1] Based on public reporting of a 2026 criminal complaint filed in a United States federal court; the technical description of the persistent identifier follows that public coverage. ArmOS makes no independent representation regarding the underlying case.

1.6 Hardware fragmentation and compatibility

Desktop, mobile, server, and development environments remain largely separate experiences that must be bridged with additional tools. And the single greatest barrier to leaving a proprietary desktop is application compatibility: moving away from Windows has historically meant abandoning software an organization depends upon. Any credible alternative must treat compatibility as a first-class requirement, not an afterthought.

1.7 A client-only network model

Finally, conventional computing treats every device primarily as a client that reaches the world through centralized services and IP-addressable intermediaries. Devices rarely cooperate directly, even when they sit on the same desk or in the same building, and even when they collectively own ample compute and storage. The prevailing model leaves that latent capacity untapped and keeps users dependent on infrastructure they do not own.

ArmOS proposes a different model in response to each of these pressures: an open foundation to reduce platform dependency; local-first operation to restore data sovereignty; privacy by architecture to remove default identification; a consolidated security platform to end fragmentation; on-device AI to keep computation private; a converged desktop-and-mobile base with genuine Windows compatibility; and peer-to-peer local groups that let a user's own devices cooperate directly. The sections that follow describe how.

2. Design Philosophy

ArmOS is guided by a set of principles that are meant to reinforce one another rather than sit as an unordered list of virtues. They fall into four groups: how the system treats the user, how it treats data and security, how it treats intelligence and networking, and how it sustains itself.

2.1 How ArmOS treats the user

Zero-retraining experience holds that a person familiar with Windows should be productive within minutes of first boot; the desktop metaphors they already know are provided by default. **Compatibility before novelty** is the discipline that enforces this promise: no new feature is permitted to break existing application compatibility, because users should never be asked to choose between the software they rely on and the platform they prefer. Together these principles keep the cost of adoption low, which is the precondition for any of the platform's other benefits to reach real users.

2.2 How ArmOS treats data and security

Privacy by architecture means the absence of persistent identifiers and default telemetry is a structural property, not a policy the user must trust. **Security by default** means kernel-enforced application confinement, secure boot, a hardware root of trust, audit logging, and file-integrity verification are active from installation, on a base image hardened to recognized benchmarks. **Post-quantum security** treats quantum-resistant cryptography as a present design requirement rather than a future project, because data captured today may be decrypted later. These three principles convert security and privacy from features into foundations.

2.3 How ArmOS treats intelligence and networking

AI-native means the assistant is part of the operating system, works offline, and can pool inference across a user's own devices — intelligence that serves the user without exporting their data. **Agent-native** extends this outward: ArmOS is designed to be a first-class participant in emerging agent-discovery standards, publishing a controlled catalog of what it can safely do for trusted agents. **Peer-to-peer native** holds that a device should be able to cooperate directly with other devices the same owner trusts, sharing compute and storage without a central intermediary. These principles reframe the device from a client into a participant.

2.4 How ArmOS sustains itself

Hardware and economic sovereignty mean the owner controls the full stack and can, optionally, take part in a device-owned economy that rewards contributed resources. **Developer-native** design gives full support to the languages and toolchains developers already use, because a platform lives or dies by what can be built on it. **Modularity** keeps hardware and software upgradeable and replaceable. And **community governance under an open foundation** keeps the platform's direction accountable, with core components open-sourced and proprietary elements reserved for what funds long-term maintenance. Sustainability, in this view, is itself a design requirement.

3. System Architecture

ArmOS is organized as six clearly separated layers. Lower layers provide the kernel and hardened base; the middle layers add virtualization, Windows-compatibility runtimes, and the distinctive ArmOS system services; and the top layer delivers the familiar desktop. Each layer depends only on those beneath it, which keeps the system maintainable and lets components evolve independently.

L6	**ArmOS UI Shell** Windows-style desktop (KDE Plasma 6), start menu, taskbar, ArmOS Store, embedded ARM Brain assistant, no-reboot mode switching
L5	**Compatibility Runtime** Wine, Proton, DXVK/VKD3D, Mono, .NET, PowerShell Core, Android runtime (Waydroid), Flutter engine
L4	**Hypervisor & Virtualization** KVM/QEMU, rootless containers, lightweight Kubernetes, Windows Server guest VMs, GPU pass-through
L3	**ArmOS System Services** Package manager, update engine, hardware-compat layer, P2P Local Groups, PQC encryption, UESP, Lockdown Mode, ARM Brain, VPN mesh, developer SDK, agent discovery, mobile mounting
L2	**Hardened Ubuntu 24.04 LTS Base** APT ecosystem, systemd, drivers, application confinement, benchmark hardening applied at build time, kernel audit, file-integrity baseline
L1	**Linux Kernel (LTS)** Scheduling, networking, filesystems, device drivers, virtualization, in-kernel observability (eBPF)

Figure 1 — The ArmOS six-layer architecture. Higher layers depend only on those below them.

3.1 Kernel and hardened base (Layers 1–2)

The foundation is a long-term-support Linux kernel with a small, well-understood set of enhancements — improved NTFS read/write support, a responsiveness-oriented scheduler for interactive workloads, secure-boot compatibility, and in-kernel observability hooks used by the security stack. Above it sits Ubuntu 24.04 LTS. Building on Ubuntu rather than maintaining an entirely independent foundation is a deliberate engineering choice: it inherits mature hardware support, an enormous software repository, and a multi-year security-maintenance cycle, and it lets the project concentrate effort where it creates the most value. Crucially, a recognized hardening baseline is applied at build time, so every installation begins from a benchmark-aligned configuration before the ArmOS layer is added.

3.2 The ArmOS system-services layer (Layer 3)

Layer 3 is the primary engineering contribution of the project and the source of most of what makes ArmOS distinctive. It provides a unified package manager that presents Linux packages, Flatpaks, and Windows installers through one interface; an atomic, rollback-capable update engine; a hardware-compatibility layer for automatic driver detection; and a settings API that bridges Linux and Windows-style configuration. On top of these sit the platform's signature subsystems — P2P Local Groups, the post-quantum encryption engine, the unified endpoint security platform, Lockdown Mode, the ARM Brain assistant and its inference services, the VPN mesh, the wallet and node components, the portable-device layer, the multi-language developer SDK, the agent-discovery layer, and the mobile drive-mounting service — with kernel audit logging and cryptographic file-integrity verification running continuously beneath them.

3.3 Virtualization, compatibility, and shell (Layers 4–6)

The virtualization layer runs guest virtual machines, including Windows Server, at near-native performance for CPU-bound workloads through KVM-based (kernel-integrated) virtualization — graphics-accelerated guests depend on GPU pass-through, a later-phase item — and supports rootless containers and a lightweight Kubernetes distribution for server and edge use. The compatibility runtime translates Windows programming interfaces to Linux and hosts the Android and Flutter runtimes; it is examined in detail in Section 4. The shell, built on KDE Plasma 6, presents a Windows-style desktop with the ARM Brain assistant embedded in search, and lets a user switch between the ArmOS shell, a standard GNOME desktop, and a full-screen terminal without rebooting.

> **Reading note** Layers 1, 2, 4, 5, and 6 are assembled predominantly from mature, widely deployed open-source technologies. The engineering novelty of ArmOS is concentrated in Layer 3 and in the integration discipline that binds the layers together.

4. Windows and Cross-Platform Compatibility

Compatibility is the single most important promise ArmOS makes to a Windows user: the software you rely on today should continue to work. It is essential to be precise about how that promise is delivered, because ArmOS does not simply 'run Windows applications' — it translates Windows programming interfaces to Linux through several complementary technologies, and provides virtualization as a guaranteed fallback. Compatibility breadth therefore depends on the specific application, its API surface, and its driver and hardware requirements.

Tier	Application class	Technology	Status
1	Win32 desktop applications	Wine + DXVK/VKD3D	MVP target
2	Games and DirectX titles	Proton (Valve)	MVP target
3	.NET applications	Mono + .NET + Wine	MVP target
4	COM / ActiveX / automation	Wine COM subsystem	Design intent
5	Modern Store (UWP/WinRT)	Experimental runtime	Roadmap
VM	Exact Windows environment	KVM Windows guest	MVP target (fallback)

Figure 2 — Windows compatibility tiers with maturity status. Coverage within each tier depends on the individual application.

The strategic principle is that users should not have to choose between the sovereignty of an open platform and access to the applications they already depend upon. Where translation is imperfect for a given program, the built-in hypervisor can run a full Windows or Windows Server instance as a guest, which guarantees that no workload is ever stranded — at the cost of running a licensed Windows image inside the virtual machine.

It is important to distinguish three states. **Current and MVP-target capability** covers Win32, DirectX, .NET, and PowerShell translation plus the virtualization fallback. **Design-intent capability** covers COM and Office-automation scenarios, where behavior depends heavily on the specific application. **Roadmap capability** covers modern Windows Store (UWP/WinRT) applications through an experimental runtime targeted for a later release. ArmOS does not assert a single headline compatibility percentage, because such a figure is only meaningful when independently measured against a defined application set; any percentages in internal material should be treated as estimates pending validation.

4.1 A desktop that adapts, and scripting that carries over

Three no-reboot desktop modes serve different users from one installation: the default ArmOS mode for daily use and application compatibility, a standard GNOME mode for those who prefer it, and a terminal mode for power users. PowerShell Core — Microsoft's own cross-platform shell — runs alongside bash, zsh, and fish, so existing administrative scripts continue to function, and the .NET runtime with Mono covers both legacy and modern .NET software. Android applications run as ordinary sandboxed windows through a containerised runtime, and Flutter applications run natively, so a single machine can target desktop, Android, and Flutter side by side.

5. Security Architecture

ArmOS is best understood as a security architecture rather than a collection of security tools. Its protections are arranged as defense in depth, so that a weakness at one layer is contained by controls at another, and so that detection and audit operate beneath the application layer where much conventional tooling cannot see.

7	**Monitoring & audit** Wazuh correlation, kernel audit logging (Auditd), cryptographic file integrity (AIDE), endpoint visibility (osquery/Fleet)
6	**Data & cryptography** Full-disk (symmetric) encryption, hardware trust module (TPM), hardware security key; post-quantum key exchange, signatures & transport
5	**Network controls** Application firewall (OpenSnitch), network detection (Suricata), zero-trust overlay (NetBird), mesh VPN
4	**Runtime detection** In-kernel behavioral detection (Falco, eBPF), anti-malware (ClamAV)
3	**OS confinement** Application confinement (AppArmor), policy engine (OPA), USB device control (USBGuard)
2	**Boot & platform** Secure Boot, TPM2 measured boot, benchmark hardening applied at build time
1	**Trust anchors** Hardware root of trust, signed images, least-privilege defaults

Figure 3 — Defense-in-depth security architecture. Lower layers anchor trust; upper layers detect, contain, and record.

The components do more than sit side by side; they are wired together. Kernel audit logging and in-kernel runtime detection feed a central correlation engine, so that a suspicious system call, an unexpected binary change flagged by file-integrity verification, and a blocked network connection can be reasoned about as one event rather than three disconnected alerts. Application confinement limits what a compromised process can reach; the policy engine expresses allow and deny decisions uniformly across subsystems; USB device control and secure boot close common physical and persistence vectors. The design goal is a system in which the default posture is restrictive and every significant action is observable and recorded.

> **Scope note** The security value of this architecture depends on correct integration and on the maturity of each component. ArmOS integrates established open-source projects; it does not claim novel cryptographic primitives or independently certified assurance levels. Formal evaluation and third-party audit are future work, not present claims.

6. The Unified Endpoint Security Platform (UESP)

Enterprises typically buy endpoint security as several separate products, each with its own agent, console, and policy model. ArmOS is designed to consolidate the functions these categories provide into a single integrated platform built on mature open-source components, with one agent, one console, and one policy engine. The intent is architectural: fewer moving parts, fewer seams, and controls that sit inside the operating system rather than bolted on top of it.

Figure 4 — UESP consolidates the functions normally spread across several products into one integrated platform. Product categories, not specific vendors, are shown.

6.1 The five functional domains

Application control provides kernel-level allowlisting, process-level firewalling, and storage and network control under a unified policy. **Endpoint detection and response** provides file-integrity monitoring, in-kernel runtime detection, anti-malware, and endpoint visibility. **Endpoint management** covers patch and software deployment, asset management, mobile device management, and remote control. **Compliance automation** continuously maps collected evidence to common frameworks and stores it in tamper-evident form. **Privileged access management** provides a credential vault, session recording, just-in-time provisioning, and key and certificate management.

6.2 Architectural benefits — and honest limits

Consolidation offers real advantages: reduced agent sprawl, fewer consoles to operate, centralized policy, lower operational complexity, potentially lower licensing cost, and tighter integration with the operating system that the controls are protecting. These benefits follow directly from the design. What must be stated plainly is that ArmOS is designed to provide capabilities that address functions commonly associated with established commercial products; it does not follow that it has replaced any of them in production. Feature parity, scale, and operational maturity are earned over time and through deployment, and the paper makes no claim that this has already occurred.

> **Language discipline** Throughout this paper, UESP capabilities are described as 'designed to address the functions of' a category — never as a proven, drop-in replacement for a named product. Replacement is a claim that requires evidence from real deployments.

7. Lockdown Mode

ArmOS includes a single-toggle extreme-security profile intended for users and situations that face elevated risk — travel across borders, high-value targets, sensitive investigations, or simply moments when a device is most exposed. The threat model it addresses is a device that may be physically seized or attacked through its most common exposure points: wired connections, message content, network peers, and configuration changes.

When engaged, the profile blocks wired data connections while the device is locked; disables attachment, link, and rich-content previews that are common exploitation vectors; blocks incoming connections from unrecognized sources by default; prevents installation of new configuration profiles without explicit confirmation; restricts scripting in the default browser; and logs and alerts on every blocked attempt. Each restriction is implemented by a dedicated mechanism — USB device control, an application firewall, application confinement, a policy lock, in-kernel runtime detection, and kernel audit logging — coordinated by the central security console.

The rationale for building such a profile into the operating system, rather than relying solely on third-party endpoint products, is that OS-level controls act earlier and lower in the stack: they can restrict a physical port or a kernel behavior before any application-layer agent is even consulted. A public example from January 2026 illustrates the general point on the mobile platform, where a comparable built-in mode was recorded in a court filing as having prevented forensic extraction from a seized device.[2] ArmOS aims to bring an equivalent, built-in capability to the desktop and laptop, an area where such a profile has historically been absent.

> **Verification note** External case references are used to illustrate why OS-level hardening matters, based on public court records and mainstream reporting. They are not presented as evidence of ArmOS's own effectiveness, which will depend on implementation and independent testing. ArmOS makes no guarantee that it can defeat any specific forensic tool.

[2] Based on public reporting of a January 2026 United States federal device-seizure matter in which data could not be extracted from a seized device with a lockdown-style protection enabled, and in which the individual was not charged. This illustrates operating-system-level hardening in general; ArmOS makes no guarantee of resistance to any specific forensic tool.

8. Privacy and Data Sovereignty

There is a meaningful difference between privacy by policy and privacy by architecture. Privacy by policy is a promise: a vendor states that it will not collect or misuse data, and the user is asked to trust that promise and the controls that enforce it. Privacy by architecture is a property: the system is built so that certain data is not collected or exposed in the first place, regardless of policy. ArmOS is positioned primarily around the second.

Concretely, this means a design with no persistent device identifier, no default telemetry, and no background collection; local-first computation, so that ordinary work — including AI inference — does not require sending data off the device; user-owned data held in encrypted storage under keys the user controls; peer-to-peer communication that need not traverse a central intermediary; and a portable mode that can run without leaving a trace on host hardware. Each of these reduces the surface over which data could be observed, rather than merely promising not to look.

> *Privacy by policy asks you to trust that no one is looking. Privacy by architecture is built so there is less to look at.*

Two honest caveats belong here. First, no operating system can promise absolute privacy: applications a user chooses to install, accounts they choose to sign into, and networks they choose to use all carry their own exposure, outside the platform's control. Second, the strength of an architectural privacy claim depends on implementation and verification; ArmOS's design intent is strong, but claims such as 'zero telemetry' and 'no persistent identifiers' are properties to be demonstrated and independently reviewed, not taken on trust. The paper states the design intent clearly and marks independent verification as future work.

9. Post-Quantum Security

Cryptography that is secure today will not remain so indefinitely. The migration to post-quantum cryptography — algorithms designed to resist attack by future quantum computers — is now an active international standards process, and the 'harvest now, decrypt later' risk means data captured today could be exposed once such machines mature. ArmOS treats post-quantum readiness as a present design requirement rather than a deferred migration.

The intended architecture applies the appropriate tool at each layer. Data at rest is protected by strong symmetric encryption — already considered quantum-resistant at adequate key sizes — with full-disk encryption established on first boot, keys protected by a hardware trust module, and an optional hardware security key as a physical second factor. Post-quantum algorithms are applied where the quantum threat actually falls, on the asymmetric cryptography: key exchange, digital signatures, and the VPN, peer-to-peer, wallet, and transport layers that depend on them. A well-established open-source post-quantum library is designated as a validated fallback beneath the platform's own cryptographic components.

Precision matters most here. Three states must be kept distinct: **design intent** (post-quantum algorithms applied at every layer named above), **implemented capability** (the subset actually shipping in a given release), and **independently validated security** (assurance established by external review). ArmOS's design intent is comprehensive, but the paper does not claim that the platform is 'fully post-quantum secure' — that is a conclusion only independent validation can support. Post-quantum standards and implementations are themselves maturing, and ArmOS's cryptographic components, including any platform-specific implementations, require careful review before any assurance claim is made.

> **Status** Post-quantum coverage is a design goal spanning disk, network, transport, and application layers. The specific algorithms shipped, and their independent validation, are release-dependent and remain in progress. Treat comprehensive coverage as intent, not as a completed, certified property.

10. AI-Native Computing: ARM Brain

ArmOS's approach to artificial intelligence rests on one idea: AI should be an operating-system capability, not an application installed on top of one. That capability is called ARM Brain. It runs locally, works offline, and can scale across a user's own hardware when more capacity is needed — and because it runs on the device, it changes the economics and governance of AI, not only its convenience.

10.1 A local, offline-first assistant

ARM Brain runs a local language model through an on-device inference runtime, and powers system search, application discovery, terminal assistance, file management, and security monitoring without an internet connection. The model chosen at install time is matched to available hardware, from compact models on modest machines to larger models on systems with capable GPUs. When online, ARM Brain can optionally draw on fresh web results and updates; when offline, it remains fully functional. Because inference is local, sensitive prompts and documents never have to leave the machine to be useful — the property that most directly changes privacy, latency, availability, cost, and enterprise data governance.

10.2 An efficiency-first inference pipeline

Running capable models on everyday hardware requires engineering rather than optimism. ArmOS integrates several proven open-source techniques into a single pipeline, each addressing a different cost.

Figure 5 — The ARM Brain inference pipeline. Lightweight requests are answered cheaply; heavier requests escalate through routing, cache reuse, shared serving, and, when needed, multiple devices.

Model routing answers simple requests with small models and escalates only when quality requires it; cache reuse accelerates repeated context; and a multi-model serving approach lets one GPU host a rotating set of supporting models — an embedder, a re-ranker, an entity extractor, a small language model — rather than dedicating memory to each. These techniques are selected to reduce GPU-memory pressure and improve responsiveness. Their exact benefit depends on workload — for example, shared serving favors sequential workloads and must be benchmarked carefully under bursty concurrent load — and ArmOS validates configurations against real usage patterns before enabling them by default. Any specific efficiency figures in internal material are estimates pending measurement.

10.3 Distributed inference across your own devices

A single laptop cannot always run the largest models. ArmOS addresses this by letting devices in a P2P Local Group combine their GPU and memory into one logical inference cluster, so a group of machines can collectively run a model none could run alone. The distributed runtime exposes a standard, chat-compatible interface, so ARM Brain calls it exactly as it would call a local model. Cross-network latency is a real consideration, so this capability is best suited to devices on a shared local network and is benchmarked before cross-location use.

10.4 On-device intelligence for mobile, and enterprise implications

On the companion mobile platform, ArmOS targets a compact model fine-tuned for the platform's own tasks and quantised to run efficiently on mobile hardware — an approach that trades breadth for a purpose-built assistant. The specific accuracy and throughput figures cited in internal material are development targets, not

independently validated results, and are treated as such. For enterprises, local inference is more than a performance choice: it means regulated data can be processed on controlled endpoints rather than sent to third-party infrastructure, which materially simplifies data-governance and residency questions — subject, as always, to correct configuration and to the organization's own controls.

> **Status** ARM Brain's local assistant, model routing, cache reuse, and multi-model serving are MVP targets. Distributed inference is an MVP target for local-network use. The mobile fine-tuned model and its cited performance figures are development targets under evaluation, not validated benchmarks.

11. Peer-to-Peer Local Groups

Conventional computing treats every device as a client that reaches the world through centralized services and an IP address. ArmOS introduces an alternative in which a user's own devices form a P2P Local Group — a set identified by cryptographic device identity rather than by network address — and cooperate directly.

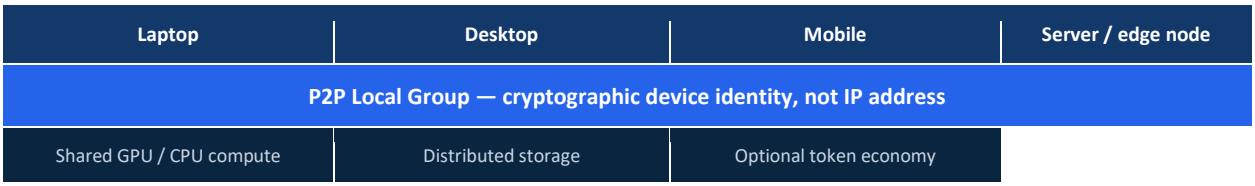

Figure 6 — A Local Group binds a user's devices by cryptographic identity and lets them share compute, storage, and — optionally — a measurable resource economy.

Membership and trust rest on device keys rather than on addresses that change and can be spoofed. Within a group, GPU and CPU capacity can be pooled for distributed inference; storage can be shared so that data is held collectively and resiliently; and because contributed resources — particularly GPU time — are measurable, they can optionally be recognized through a device-owned token economy. Underlying this are established peer-to-peer networking and distributed-storage technologies, hardened with the platform's cryptography.

It is important to distinguish the near-term implementation from the longer-term vision. The MVP target is basic Local Groups with distributed inference across devices on a shared network. The broader vision — a global network of Local Groups with a full distributed-compute economy — is a roadmap direction whose value and viability, including regulatory questions around any token economy, remain to be established. The paper presents it as an aspiration with a credible architectural starting point, not as a delivered system.

12. Mobile and Desktop Convergence

Desktop and mobile computing are normally separate worlds, bridged with sync tools and compromises. ArmOS is designed around a converged base: the same firmware philosophy spans desktop and a companion mobile device, so that synchronization is native rather than a compatibility layer. A mobile device connected to a display, keyboard, and pointer can present a full desktop experience, and its storage can mount on the desktop as a true filesystem — accessible to any application, script, or backup tool rather than only through a file-manager view.

The potential implications differ by audience. For developers, one converged environment can build and test desktop, mobile, and Flutter targets together. For enterprises, a converged, centrally securable base reduces the number of distinct platforms to manage and harden. For remote and field workers, a single device can serve as phone, workstation, and secure endpoint, with the same local AI and the same security posture in each mode. These are design-led possibilities; the depth of convergence available in any given release is roadmap-dependent, and the mobile hardware and its specifications remain to be finalised.

> **Status** Real-time desktop-mobile synchronization and native drive mounting are MVP targets on supported hardware. Full mobile convergence (desktop mode from a phone) and the mobile hardware itself are roadmap items with specifications yet to be finalised.

13. The Developer Platform

A platform lives or dies by what can be built on it, so ArmOS ships as a complete, pre-configured development environment. Rather than favoring one language, it provides strong, native support across the languages and frameworks developers already use, with a unified path from source to a signed, distributable application.

Language / framework	Recommended desktop approach	Notes
JavaScript / Node.js / Next.js	Tauri (native webview) or Electron	Smaller, faster native packaging via Tauri
Python	Qt bindings or Flet	Native to ArmOS — no extra runtime required
Go	Wails or Fyne	Single static binary — simplest packaging
Rust	Tauri native backend	The native language of the ArmOS system stack
PHP / Laravel	Native packaging or web-view	Internal business and admin tools
Java	OpenJDK with JavaFX or Compose	Enterprise and legacy Java support
.NET	.NET and Mono (Layer 5)	Legacy and modern .NET run natively
Flutter	Flutter engine (Layer 5)	One codebase for ArmOS desktop and Android
C / C++	GTK or Qt via APT toolchains	The full native Linux development stack

Figure 7 — The multi-language developer matrix. Common runtimes and toolchains are pre-installed.

A single SDK provides project templates across every language path, and every application builds to the same packaging and code-signing format through an integrated version-control and continuous-integration pipeline into the ArmOS Store. Common runtimes — Python, PostgreSQL, Go, Rust, Node.js, and others — are pre-installed, and the ARM Brain codebase-indexing capability gives the assistant structural understanding of large projects for terminal and development assistance. The developer-native philosophy is that the shortest path from idea to running, signed application should already be present on the machine, not assembled by hand.

> **Status** The core SDK and the Node.js/Tauri, Python, and Go paths are MVP targets. Additional packaged paths — Rust, PHP/Laravel, Java, and .NET desktop packaging — are roadmap items scheduled for a later release.

14. Agent-Native Computing

Software is increasingly operated not only by people but by autonomous agents. ArmOS approaches this shift as a first-class concern, with native support for agentic resource discovery — an open approach in which a system publishes a machine-readable catalog of the capabilities it can safely offer to trusted agents, with domain ownership serving as the cryptographic root of trust. Through this catalog, an ArmOS device can expose well-defined capabilities — local model inference and semantic file search, analysis and compliance agents, search and retrieval, wallet operations, and Local-Group compute and storage sharing — as discoverable, interoperable services.

The significance is that the operating system participates in an agent ecosystem instead of only hosting applications: other trusted agents can discover and invoke what a device offers, and the device can discover and use capabilities elsewhere, all governed by cryptographic identity and explicit policy. The paper is careful, however, not to overstate the maturity of the surrounding standards. Agent-discovery specifications are young and their industry adoption is still forming; ArmOS's support is a design commitment and an early implementation, not a claim about settled, widely adopted standards. Publishing the capability catalog is an MVP target; deep registry interoperability is a roadmap direction.

15. Portable and Hardware-Sovereign Computing

ArmOS is designed to run its full environment from an external drive, leaving no trace on the host hardware. The portable stack — containers, database, secrets store, VPN, and security configuration — lives on the external drive under full-drive encryption and quantum-resistant volumes, so that a user can carry their entire trusted environment and run it on borrowed hardware without persisting data to that machine. On installed systems, a detected secondary drive can be mirrored in real time for redundancy, and snapshot-based tooling supports bare-metal recovery.

These capabilities express the hardware-sovereignty principle: the owner controls the full stack, from the encrypted volume to the hardware trust module, and can move it at will. The zero-host-trace property is a design goal whose completeness depends on hardware behavior and correct configuration; like other strong claims in this paper, it is stated as intent to be verified rather than as a guarantee, since firmware, peripheral, and hardware variations can affect what 'no trace' means in practice.

> **Status** Portable full-environment operation, LUKS and quantum-resistant volumes, and RAID-1 mirroring on secondary-drive detection are MVP targets. The zero-host-trace property is a design goal subject to hardware-dependent verification.

16. Enterprise Architecture and Use Cases

The architectural properties described so far combine into distinct value for several classes of organization. The scenarios below are presented as potential use cases enabled by the architecture, not as existing deployments; ArmOS asserts no production references at this time.

16.1 Financial services

Regulated financial institutions can pair hardened, auditable endpoints with local AI so that sensitive analysis runs on controlled machines rather than external services. The consolidated security stack provides application control, detection, and privileged-access management under one policy engine, while kernel audit logging and file-integrity verification supply the kind of evidence trail that compliance regimes expect. Controlled, minimally connected environments are straightforward to construct on a local-first platform.

16.2 Government and public sector

Sovereign computing is a direct fit for public-sector requirements: an open foundation reduces dependence on a single foreign vendor; local-first operation supports offline and air-gapped scenarios; and built-in hardening plus Lockdown Mode raise the baseline posture of every endpoint. Data residency and lawful-access questions are easier to reason about when computation and storage remain on owned hardware under owned keys.

16.3 Healthcare

Healthcare settings benefit from local processing of sensitive records, encrypted storage under organizational control, and on-device AI that can assist without transmitting patient data off the endpoint. As always, regulatory compliance is a property of the whole system and its operation, not of the operating system alone; ArmOS provides architectural building blocks that support such compliance rather than conferring it automatically.

16.4 Developers, security operations, and remote workforces

Development teams gain a complete, pre-configured environment with local AI, containers, and lightweight Kubernetes on the same machine. Security operations teams gain integrated endpoint visibility, audit, detection, and control without assembling and correlating several separate products. Remote and field workforces gain a portable, secured environment with a built-in mesh VPN and consistent endpoint security, whether working from a laptop or, in time, a converged mobile device. In each case the value comes from integration — capabilities that normally require several products arriving as properties of one platform.

> **Framing** All scenarios in this section are potential use cases enabled by the architecture. No customer deployments, production references, or sector certifications are claimed. Compliance outcomes depend on the complete system, its configuration, and organizational controls.

17. Product Editions

ArmOS is offered in three editions so that individuals, professionals, and organizations each receive an appropriate capability set and support level. The distinction is architectural rather than commercial: editions differ in the depth of security, management, and virtualization capability they unlock, and in the support commitment behind them.

Capability	Community	Professional	Enterprise
Target user	Individuals	Professionals / SMB	Organizations
ArmOS shell & Windows compatibility	✓	✓	✓ (headless optional)
ARM Brain local AI & distributed inference	✓	✓	✓
Post-quantum encryption & hardened base	✓	✓	✓
UESP security stack	Basic	Full	Full + SIEM
Lockdown Mode	✓	✓	✓
Attack-surface monitoring	—	✓	✓
Privileged access management	Basic	Full	Full
Compliance automation & MDM	—	Basic / ✓	Full
Hypervisor & Kubernetes	Basic	Full	Full + clustering
Directory / LDAP integration	Client	Full	Full + RADIUS
Support	Community	Business hours	24/7 + account manager

Figure 8 — Edition capabilities. Pricing is deliberately de-emphasized; personal use is intended to be free, with paid professional and enterprise tiers. Prices and inclusions are indicative and subject to confirmation at launch.

18. Open-Source and Licensing Strategy

ArmOS follows a hybrid model: open where openness serves users and the ecosystem, proprietary where proprietary components fund the platform's long-term maintenance. Open-sourced elements include kernel patches and driver work, the ArmOS system-services layer, the terminal and user-interface component libraries, compatibility-layer patches, the Android integration layer, the build and testing pipeline, the security-stack integrations, the developer-SDK templates, the agent-discovery catalog specifications, and the hardening configuration profiles. Reserved as proprietary are the premium shell themes, the store back end and compatibility scoring, the enterprise management console, the AI routing and optimization logic, the platform-specific cryptographic implementation, the token-economics engine, and the office-suite core.

Governance is vested in a non-profit foundation that oversees the open-source components, guided by an elected technical steering committee and supported by tiered sponsorship. The intent is a platform whose direction is accountable to its community rather than to any single commercial interest. One area demands explicit caution: combining a Linux and GPL-family base with proprietary components raises licensing questions that must be handled correctly. This paper does not offer legal advice; the boundaries between open and proprietary components, and their compatibility with the licenses of all incorporated projects, require review by qualified counsel before release. The relevant open-source projects retain their own licenses, credited in full product documentation.

> **Legal review required** The interaction between GPL-family base components and proprietary ArmOS components, and compliance with every incorporated project's license, is a matter for qualified legal counsel. Statements here describe intent, not a legal conclusion.

19. Competitive Landscape

Several Linux-based systems offer a familiar desktop, and each serves its purpose well; ArmOS does not seek to disparage them. Its differentiation is architectural — the combination of genuine Windows compatibility, native local AI, consolidated security, post-quantum design, and peer-to-peer cooperation in one platform. The comparison below is category-level and is intended to clarify positioning rather than to assert superiority in every cell.

Capability	ArmOS	General Linux desktops	Windows 11
Open Linux foundation	✓	✓	✗
Windows-style desktop	Full	Varies	Native
Windows application compatibility	Broad (Wine/Proton/VM)	Basic (Wine)	Native
.NET / PowerShell	Native	Limited	Native
Android runtime	✓	Rare	Partial
Built-in hypervisor	✓	Add-on	Pro editions
Native local AI assistant	✓	—	—
Distributed inference across devices	✓ (design)	—	—
Consolidated endpoint security	✓ (design)	—	—
Built-in Lockdown-style profile	✓	—	—
No persistent device identifier	✓ (design)	✓	GDID present
Post-quantum cryptography	✓ (design)	—	—
Peer-to-peer local groups	✓	—	—
Portable / trace-free mode	✓	—	—
Multi-language developer platform	✓	Varies	Partial
License cost	Free + paid tiers	Free	Paid

Figure 9 — Category-level comparison. '(design)' marks ArmOS capabilities that are architectural commitments or MVP targets rather than independently validated results. The Windows Global Device ID (GDID) row reflects publicly documented facts summarized in Section 1.

20. Strategic Roadmap

ArmOS is developed in disciplined phases, each building on a stable base. Future capabilities are described as roadmap items and are not to be read as present features. Timing is indicative and scope may evolve.

Phase 1 — Foundation (v1.0)

A bootable ArmOS desktop with the Windows-style shell; the local AI assistant with GPU support and the core inference pipeline; core Windows compatibility through Wine and Proton with the virtualization fallback; basic P2P Local Groups with local-network distributed inference; post-quantum disk and transport encryption; the built-in VPN mesh; the hardened baseline with kernel audit and file-integrity layers; the consolidated security stack and Lockdown Mode; agent-discovery catalog publishing; real-time mobile synchronization with native drive mounting; portable and mirrored-drive modes; and the core multi-language developer SDK.

Phase 2 — Expansion (v1.5)

Broader Windows application compatibility as GPU pass-through matures; full endpoint-security deployment including privileged-access management, compliance automation, and device management; the optional wallet and token economy; the mobile convergence experience; enterprise activation; the Android runtime with its developer toolkit; the first ArmOS Store release; and extended developer SDK paths across additional languages.

Phase 3 — Convergence (v2.0)

A resilient multi-relay mesh; an ArmOS server edition positioned as a Windows Server alternative; a modular ArmOS laptop; a global Local-Group network with a distributed-compute economy; more capable default AI models on suitable hardware; and a modern Windows-Store compatibility layer.

Phase 4 — Ecosystem (v3.0)

A full peer-to-peer application architecture; a mature token economy; international expansion; a public application store; strategic modular-hardware partnerships; support for additional processor architectures; and self-hosted sovereign-cloud options for enterprises.

> **Reading note** Everything in Phases 2–4 is a roadmap item. The presence of a capability here is a statement of intended direction, not of current availability.

21. Risks and Mitigations

A credible platform paper states its risks plainly. The table below sets out the principal risks to the ArmOS program, each with its impact, the mitigation in place or planned, and the uncertainty that remains after mitigation. This is an engineering assessment, not a reassurance exercise.

Risk	Impact	Mitigation	Remaining uncertainty
Windows compatibility gaps	Some apps run imperfectly under translation	Virtualization fallback; curated compatibility data	Per-app behavior until broadly tested
GPU pass-through for Windows guests	Limits certain accelerated VM workloads	Proton as interim; VFIO/KVM evaluation	Hardware-dependent; a Phase-2 focus
Third-party dependency risk	Upstream projects change or stall	Prefer mature projects; contribute upstream	External roadmaps not controlled by ArmOS
Post-quantum maturity	Standards and libraries still evolving	Design-level integration; validated fallback library	Assurance pending independent review
AI inference performance	Large models strain modest hardware	Routing, cache reuse, shared serving, distribution	Benefits workload-dependent; needs benchmarking
P2P latency	Cross-network inference can be slow	Scope distribution to local networks first	Cross-location performance unproven
Licensing / IP interaction	Open/proprietary boundary complexity	Clear boundaries; legal review before release	Requires qualified counsel sign-off
Token-economy regulation	Jurisdictional and compliance exposure	Opt-in; per-jurisdiction legal review	Regulatory treatment varies and evolves
Security-tool maturity	Integrated stack must equal point tools	Established components; staged rollout	Parity earned through deployment
Ambitious first-release scope	Breadth risks delivery quality	Strict MVP scope; defer non-essentials	Execution risk inherent to scope

Figure 10 — Principal program risks. Stating remaining uncertainty explicitly is itself a mitigation against over-claiming.

22. Security and Trust Model

Every security architecture rests on assumptions about what it can trust and what it must defend against. Making those assumptions explicit is a mark of maturity, because no operating system can eliminate every threat, and a platform that implies otherwise should not be believed.

22.1 What ArmOS assumes it can trust

ArmOS assumes trustworthy underlying hardware and firmware, a functioning hardware root of trust and TPM, an intact secure-boot chain, and a legitimate local administrator acting in good faith. It assumes that signed platform images have not been tampered with upstream. Where these assumptions fail — a compromised supply chain, a malicious administrator, or subverted firmware — the guarantees the platform can offer are correspondingly reduced, and no software layer can fully compensate.

22.2 What ArmOS is designed to defend against

Within those assumptions, ArmOS is designed to resist malicious applications through confinement and allowlisting; compromised credentials through privileged-access controls and session recording; network threats through the application firewall, network detection, and zero-trust overlay; physical exposure through disk encryption, USB control, and Lockdown Mode; and persistence and tampering through secure boot, kernel audit, and cryptographic file-integrity verification. Detection and audit operate below the application layer so that activity conventional tooling misses is still recorded.

22.3 What no operating system can promise

ArmOS does not claim to defeat a determined adversary with physical possession and unlimited resources, to eliminate insider threats, to guarantee against undiscovered vulnerabilities in any incorporated component, or to remove the exposure created by applications and accounts the user chooses to trust. Its aim is to raise the cost of attack substantially and to make the state of the system observable and recoverable — not to offer an unrealistic promise of absolute security. Stated this way, the model is intended to make ArmOS appear mature rather than invulnerable.

23. Architectural Differentiators

Ten characteristics distinguish ArmOS as an architecture. Each is a design commitment; where one depends on capabilities still maturing, the paper has said so in the relevant section.

- **Open foundation with genuine Windows compatibility.** A hardened Linux base that runs existing Windows, .NET, and PowerShell software through clean-room translation and a virtualization fallback — sovereignty without abandoning the software users depend on.
- **Privacy by architecture.** No persistent device identifier and no default telemetry as structural properties, in deliberate contrast to platforms that identify devices by default.
- **AI as an operating-system capability.** A local, offline-first assistant with an efficiency-first inference pipeline, keeping computation and data on the device.
- **Distributed inference across owned devices.** A user's own machines pooled into one inference cluster, so capability scales without the cloud.
- **Consolidated endpoint security.** One integrated stack, one console, one policy engine addressing functions normally spread across several products.
- **Built-in Lockdown Mode on the desktop.** An OS-level extreme-security profile acting below the application layer, an area where built-in desktop capability has been absent.
- **Post-quantum design throughout.** Quantum-resistant cryptography treated as a present requirement across disk, network, and transport, with a validated fallback.
- **Peer-to-peer local groups.** Devices bound by cryptographic identity rather than IP address, cooperating directly on compute and storage.
- **Desktop and mobile convergence.** A shared base so the two worlds converge, with native cross-device sync and drive mounting.
- **Developer-native and agent-native.** A complete multi-language build environment on the machine, and a device that can participate in emerging agent ecosystems under cryptographic identity.

24. Conclusion

ArmOS is not another desktop Linux distribution with a new coat of paint. It is an attempt to build a different computing model — local-first, AI-native, security-first, privacy-oriented, hardware-sovereign, peer-to-peer, and developer-native — on an open foundation, while keeping the software users already depend on. The problems it addresses are documented and growing: persistent device identification, default data collection, cloud dependency, security fragmentation, and the recurring cost and lock-in of proprietary platforms.

The paper has been deliberate in separating what is designed from what is targeted for first release and what remains on the roadmap or under evaluation. That discipline is not a hedge; it is the point. An honest account of an ambitious program serves a serious reader better than an inflated one. If ArmOS delivers even the foundation it targets, it will show that a Windows-compatible, private, AI-native operating system on an open base is not only desirable but buildable.

The long-term implication, stated without pretending to certainty, is that the operating system need not be an instrument of observation and dependency. It can instead be what it was always meant to be: an environment that runs the user's software, protects the user's data, and answers to the user. ArmOS is an argument, in architecture, that this is achievable.

> *Own your hardware. Own your data. Own your network. Own your computing.*

References

The following open-source projects and standards are referenced in the ArmOS architecture. Each retains its own license and documentation; readers should consult the official project sources for authoritative and current information. This list is representative of the technologies discussed in this paper and is not exhaustive.

Linux kernel — kernel.org

Ubuntu LTS — ubuntu.com

KDE Plasma — kde.org

Wine — winehq.org

Proton (Valve) — github.com/ValveSoftware/Proton

Mono / .NET — dotnet.microsoft.com

Waydroid (Android runtime) — waydro.id

KVM / QEMU — qemu.org

Podman — podman.io

Ollama — ollama.com

Exo (distributed inference) — github.com/exo-explore/exo

Wazuh (XDR/SIEM) — wazuh.com

Falco (runtime detection) — falco.org

AppArmor — apparmor.net

OpenSnitch (application firewall) — github.com/evilsocket/opensnitch

OPA (policy engine) — openpolicyagent.org

OpenBao (secrets / PAM) — openbao.org

WireGuard — wireguard.com

NetBird (zero-trust mesh) — netbird.io

Suricata (network detection) — suricata.io

osquery / Fleet — osquery.io

ClamAV — clamav.net

USBGuard — usbguard.github.io

AIDE (file integrity) — aide.github.io

libp2p — libp2p.io

IPFS — ipfs.tech

Open Quantum Safe (liboqs) — openquantumsafe.org

Tauri — tauri.app

Wails — wails.io

Timeshift — github.com/linuxmint/timeshift
