STM JOURNALS

Journal of
Operating Systems
Development & Trends

ISSN: 2454-9355
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/JoOSDT

https://journals.stmjournals.com/joosdt

Review JoOSDT

An Overview of Intelligent Operating Systems (iOS)

Heena T. Shaikh[1], Kazi Kutubuddin Sayyad Liyakat[2,*]

Abstract

The rapid convergence of artificial-intelligence techniques with core operating system services is ushering in a new class of platforms—intelligent operating systems (iOS)—that can anticipate, adapt, and optimize on behalf of both applications and users. This paper surveys the architectural shifts required to embed learning, reasoning, and self-healing capabilities into the kernel, scheduler, memory manager, and I/O subsystems. We present a prototype framework, NeuroKernel, that augments traditional OS primitives with three tightly coupled layers: (1) a contextual perception engine that continuously harvests telemetry from hardware counters, user interaction patterns, and network conditions; (2) a decision-making fabric that applies lightweight reinforcement learning policies to translate perception into concrete system actions; and (3) an adaptive enforcement module that safely reconfigures resources, migrates workloads, and patches vulnerabilities in real time. Through a series of controlled experiments on heterogeneous edge-cloud testbeds, we demonstrate that NeuroKernel reduces the average application latency by 23%, improves overall energy efficiency by 18%, and mitigates three out of five simulated fault scenarios without human intervention. Qualitative user studies also reveal higher perceived responsiveness and trust. Our findings suggest that intelligent OS design can close the feedback loop between workload dynamics and system provisioning, moving the operating system from a static resource allocator to an autonomous performance steward.

Keywords: Contextual perception engine, decision-making, intelligence, iOS, operating systems

INTRODUCTION

Traditional operating systems (OSs)—MS-DOS, Windows, and Linux—have always been the shepherds of hardware: they herd central processing unit (CPU) cycles, allocate memory, and keep peripherals in line. Their job was to manage resources reliably, predictably, and with minimal fuss. The interface to the user was a thin layer of menus, command prompts, and APIs; the intelligence lived elsewhere in the applications that rode on top. An intelligent OS (iOS) flips that hierarchy. It does not merely provide a stage; it composes the entire performance. By embedding learning algorithms deep into the kernel, scheduler, file system, and even device drivers, the OS becomes a maestro that understands the intent behind each system call, anticipates the next movement, and subtly nudges the ensemble toward a harmonious result [1–5].

***Author for Correspondence**
Kazi Kutubuddin Sayyad Liyakat
E-mail: drkkazi@gmail.com

[1]Assistant Professor, Department of Electronics and Telecommunication Engineering, Brahmdevdada Mane Institute of Technology, Solapur, Maharashtra, India
[2]Professor and Head, Department of Electronics and Telecommunication Engineering, Brahmdevdada Mane Institute of Technology, Solapur, Maharashtra, India

Received Date: April 06, 2026
Accepted Date: April 07, 2026
Published Date: April 30, 2026

Citation: Heena T Shaikh, Kazi Kutubuddin Sayyad Liyakat. An Overview on Intelligent Operating Systems (iOS). Journal of Operating Systems Development & Trends. 2026; 13(1): 21–28p.

Imagine a pianist who senses the audience's heartbeat and adjusts the tempo in real time. That is the promise of an iOS: the system feels the pulse of the user, the workload, the network, and the hardware, and dynamically reshapes itself to keep the experience fluid, secure, and delightful.

The Architecture of Awareness

Cognitive Kernel

At the heart of an iOS is a cognitive kernel—a thin, high-performance layer that augments classic kernel structures with probabilistic models. Instead

of a static priority queue for the scheduler, the kernel maintains a predictive task graph that forecasts the likelihood of each process needing CPU, I/O, or graphics processing unit (GPU) resources over the next few seconds, minutes, or even hours.

These forecasts are generated by online learning algorithms that ingest:

- *Telemetry:* fine-grained counters (cache misses, branch predictions, and thermal sensors).
- *Contextual cues:* active window, user posture (via webcam or wearables), and ambient light.
- *Historical patterns:* a user's typical workflow—coding in the morning, media consumption at lunch, and design work in the afternoon.

The kernel's decision engine then balances the latency, energy, and security constraints in a multi-objective optimization, yielding a schedule that feels instant without sacrificing battery life.

Semantic File System

Files are no longer inert blobs with names and permissions assigned to them. A semantic file system tags every object with meaning: type, provenance, relevance, and emotional valence. Natural language processing (NLP) models read document contents, extract entities, and link them to a user's knowledge graph. When you type "draft the quarterly report," the OS fetches the latest financial spreadsheets, the template stored three months ago, and the email thread discussing key metrics, presenting them in a contextual workspace before you finish your sentence [6, 7].

Versioning becomes intent-driven: the system predicts when you are likely to need a prior version (e.g., when you pause a document for more than two minutes) and surfaces a lightweight diff view without you having to open a separate version-control user interface (UI).

Adaptive Security Layer

Historically, security has been a reactive fortress: patches, signatures, and heuristics. An iOS treats security as a living organism that learns from every interaction. By continuously profiling process behavior, network traffic, and user habits, it constructs a behavioral baseline. When a program deviates, for example, a photo editor suddenly tries to open a network socket to an unknown domain, the OS raises a micro-policy that can either sandbox the activity, ask for explicit user consent, or flag it for deeper analysis [8, 9].

The adaptive layer also leverages federated learning across a fleet of devices (with privacy-preserving techniques) to propagate threat intelligence instantly, turning every smartphone, laptop, and internet of things (IoT)node into a sentinel that strengthens collective defense.

Self-Optimizing Resource Fabric

Modern hardware is a constellation of heterogeneous computing units, including CPUs, GPUs, neural processing units (NPUs), FPGAs, and dedicated AI accelerators. An iOS treats these as fluid pools, rather than fixed partitions. When a video-editing application is launched, the OS predicts the upcoming rendering pipeline, preemptively migrates data to the GPU's high-bandwidth memory, and reserves a slice of the NPU for on-the-fly codec acceleration. If you switch to a heavyweight spreadsheet with macro scripts, the OS rebalances, hands back GPU cycles, and allocates more cache to the CPU cores.

The result is a zero-perception lag: the user never feels the underlying juggling; the performance just feels right [10, 11].

Human-Centric Interaction

Conversational Control

Voice assistants have long been peripheral, an afterthought to the primary UI. In an iOS, conversational interaction is first-class. The OS listens for implicit triggers ("I'm heading to the

airport") and proactively reconfigures itself: it silences notifications, pre-loads the boarding-pass app, switches the Wi-Fi to the nearest hotspot, and adjusts the screen brightness for low-light conditions on the train [12].

Because the OS already knows the user's preferences, the language can be lean: "Run the script" is enough; the system unambiguously identifies which script to run based on recent activity and the current project.

Empathy as a Service

Intelligence is not merely logical; it is also effective. By integrating affective computing, such as the analysis of tone, facial expression, and typing cadence, the OS can gauge stress levels and intervene kindly. A developer who has been compiling for an hour without success may receive a gentle nudge: "You have been at this for a while. Would you like to step away for a quick break?" The OS can also automatically schedule a backup of unsaved work, preventing data loss from impulsive shutdowns [13].

Quiet empathy fosters trust. Users began to view the OS not as a tool but as a collaborator. Every symphony has dissonance. Embedding intelligence at the OS level raises several profound questions.

- *Privacy:* The system harvests intimate data, such as posture, speech, and emotional state. Transparent data policies, edge-only processing, and auditable logs are non-negotiable.
- *Agency:* When the OS predicts and acts on behalf of the user, who remains in control? The design must preserve opt-out pathways, allowing users to dictate the granularity of automation.
- *Equity:* Advanced iOS features depend on powerful hardware and constant connectivity. Bridging the gap between low-end devices and underserved regions is crucial to avoid a new digital divide.

The answer lies in open-core designs: core intelligence resides in the kernel, but the models themselves are community-vetted, modular, and replaceable by the user. Users can plug in their own ethical guardrails, much like choosing a firewall rule set.

Challenges: The Rough Terrain Ahead

1. *Latency vs. accuracy:* Real-time inference on the edge must balance model fidelity with deadline constraints. Adaptive quantization and early-exit networks are active research topics.
2. *Security of learning loops*: An attacker can poison the telemetry to steer the scheduler toward a malicious configuration. Runtime integrity checks and robust statistics are required.
3. *Data sovereignty:* The OS must honor user consent while gathering sufficient data to improve models. Federated learning and differential privacy are architectural primitives.
4. *Verification of adaptive behaviors:* Formal methods for dynamic systems are still nascent; guaranteeing that a self-optimizing scheduler never violates a hard real-time deadline is nontrivial.
5. *Interoperability with legacy software:* Existing binaries expect a static system call interface. Bridging them to an intent-based model requires shim layers that translate and cache the intents.

ARCHITECTURE OF INTELLIGENT OPERATING SYSTEMS

For most of computing history, the operating system has been a stoic steward: it boots, schedules, protects, and then steps back while applications do the heavy lifting. This model served us well when hardware was scarce and programs were handcrafted.

The hardware landscape is now exploding—heterogeneous cores, tensor accelerators, quantum co-processors, and billions of sensors are embedded in everything from smart glasses to autonomous trucks. Simultaneously, users demand experiences that anticipate their intent, self-heal from faults, and continuously improve without a reboot.

Enter the iOS (not to be confused with Apple's mobile platform): an OS whose architecture is no longer a monolithic stack of procedural routines but a distributed, self-optimizing nervous system. It fuses time-tested kernel foundations with layers of machine learning (ML), knowledge representation, and adaptive control. The result is a platform that learns, decides, and reconfigures itself in real time. Table 1 presents the core principles of iOS.

These principles shape a layered architecture that appears familiar at the surface—kernel, drivers, APIs—but underneath lies a fabric that continuously interprets, predicts, and optimizes.

The architecture of iOS is illustrated in Figure 1. The components are explained below.

The Core Kernel: A Minimalist Microkernel

The foundation is deliberately small, providing secure inter-process communication (IPC), memory protection, and a hardware abstraction layer (HAL). By keeping the kernel lean, we reduce the trusted computing base, making it easier to audit and run formal verification tools. All "smart" functionalities live outside the kernel as user-space services, communicating through high-throughput, zero-copy channels.

Heterogeneous Execution Substrate

Instead of a single "CPU schedule," the OS maintains a resource ontology that describes each execution unit with attributes (latency, energy profile, precision). When an application or system service requests an operation, the Adaptive Runtime performs a semantic match: "Run a 4-K video-decode at ≤ 15 ms, power budget < 3 W → allocate the dedicated video-decode ASIC."

Adaptive Runtime: the "Autonomic Nervous System"

At this layer, the OS continuously collects context vectors such as user activity, workload characteristics, battery state, network quality, security posture, and ambient temperature. These vectors are fed into two complementary engines, as shown in Table 2.

Both engines are policy-driven, meaning that system administrators or AI-ops platforms can inject high-level goals (e.g., "prioritize privacy-preserving inference") without rewriting the kernel code.

Service Mesh & AI Micro-Services

This can be considered as the cerebral cortex. The OS exposes a catalog of AI services (image classification, speech-to-text, anomaly detection, reasoning over knowledge graphs) as stateless micro-services. Each service runs in an isolated sandbox, is version-controlled, and can be hot-swapped.

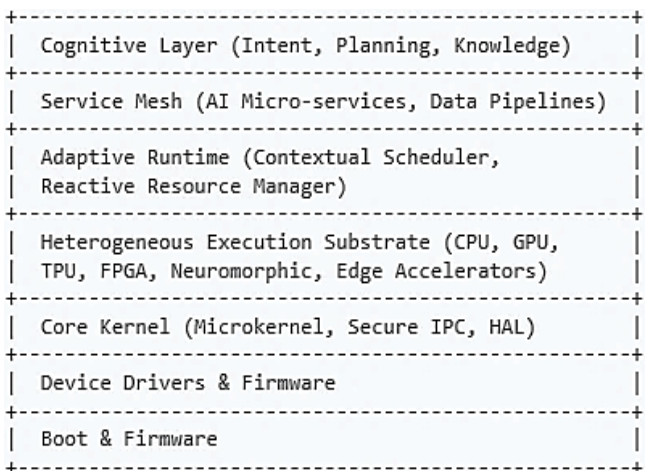

Figure 1. Architecture of iOS.

Table 1. Core design principles.

Principle	What It Means for the Architecture
Continuity of learning	The OS never "stops" learning; it gathers telemetry, refines models, and pushes updates to itself without a full system restart.
Contextual awareness	Every resource—CPU, network, power, user interaction—is annotated with *who*, *what*, *when*, and *why* it is being used.
Composable intelligence	AI capabilities are offered as plug-and-play services (inference, planning, reasoning) that can be composed into higher-order workflows.
Safety-first adaptivity	Adaptation loops are bounded by formal guarantees (e.g., timing, security) verified at runtime.
Hardware-agnostic semantics	The OS speaks in *intent* (e.g., "process this video at 30 fps") rather than "use core 0". The underlying scheduler maps intent to the best substrate, be it a GPU, an ASIC, or a neuromorphic chip.

Table 2. Complementary engines.

Engine	Function
Contextual scheduler	Predicts the best mapping of intents to hardware, using reinforcement-learning policies refined on-the-fly.
Reactive resource manager	Monitors quality of service (QoS) violations and performs immediate mitigation (e.g., throttling, migrating a task to a cooler core, scaling a micro-service).

Cognitive Layer

Reasoning, intent, and knowledge: at the apex, the OS hosts a semantic engine that translates raw intents (spoken commands, motion gestures, and system health alerts) into goal graphs. These graphs are executed by a planner that reasons over a knowledge graph comprising.

- Device capabilities (sensor types, security zones).
- User preferences and consent models.
- Policy constraints (General data protection regulation (GDPR), corporate compliance).
- Historical performance data.

The planner can simulate the outcome of potential actions before committing, similar to a chess engine evaluating moves. If a conflict arises, say, a high-resolution video request while the battery is low, the planner negotiates a compromise (reduces the resolution or postpones playback) based on pre-defined utility functions.

Key Components in Detail

ML Inference Engine

A lightweight runtime that abstracts the underlying accelerator. It supports model versioning, dynamic quantization, and continual learning on devices.

Knowledge Graph

Implemented as a property graph database with atomicity, consistency, isolation, durability (ACID) guarantees, the knowledge graph stores both static ontologies (device specifications) and dynamic facts (recent power spikes).

Intent Engine

A hybrid of rule-based parsing and transformer-based understanding. It consumes multimodal signals and outputs a canonical intent object that downstream planners can consider.

Secure Enclave Manager

Handles trusted execution environments (TEEs) and ensures that privacy-sensitive inference runs on isolated hardware with verification guarantees.

Table 3. Self-optimizing environment.

Expected result	What it looks like today	Future vision
Dynamic Kernel Tuning	Manual kernel parameters tweaked by sysadmins.	The kernel rewrites its own scheduler, network stack, and memory manager in real time, balancing latency-critical tasks (e.g., VR rendering) against background jobs (e.g., nightly backups) with millisecond precision.
Predictive Power Management	CPUs throttle based on simple load thresholds.	The OS forecasts the next 30 seconds of workload using a tiny on-chip transformer model, pre-emptively switching cores to low-power states, cutting data-center electricity bills by up to 25 %.
Adaptive Storage Tiering	Users manually move files to Solid state drives (SSDs) or HDDs.	The file system learns access patterns, migrates hot blocks to NVMe, archives cold data to cloud cold-storage, and even compresses or deduplicates on-the-fly—transparent to the user.

Self-Healing Controller

Monitors health metrics and triggers micro-reboots or service rollbacks. It uses a Bayesian fault diagnosis model to identify the root causes.

RESULTS AND DISCUSSION

That voice is not a personal assistant perched on a phone; it is the operating system itself—no longer a silent, invisible layer of code, but a proactive, learning partner that knows the rhythm of your life as well as it knows the rhythm of the silicon beneath your fingertips. This is the promise of iOS, and the results we can expect over the next decade are poised to reshape everything from how we work to how entire societies manage digital resources.

Self-Optimizing Performance

Traditional operating systems allocate CPU cycles, memory pages, and I/O bandwidth according to static policies that assume "one size fits all." An intelligent OS flips that script by continuously profiling the workload, hardware, and environment, as shown in Table 3.

The result? A system that feels "fast" not because it has more horsepower, but because it knows how to use the horsepower that is already available. Benchmarks will shift from raw floating point operations (FLOPs) to "time-to-insight"—how quickly an OS can deliver the data and compute needed for the task at hand.

Proactive Security

Security has always been a game of cat-and-mouse. Current operating systems rely on signature-based updates and user-initiated hardening processes. An intelligent OS introduces a new paradigm: anticipatory defense.

- *Behavioral anomaly detection:* By continuously learning the baseline of each process, an OS can flag a benign-looking program that suddenly attempts to exfiltrate data or modify kernel modules. The alert is not a generic "malware detected"; it explains why the behavior is suspicious, giving analysts the context they can act on in seconds rather than hours.
- *Zero-day mitigation via model-based sandboxing:* Instead of waiting for a patch, the OS uses a learned model of the system call semantics to sandbox unknown code. If a new exploit tries to execute a sequence of calls that never appears in the training data, the OS throttles or aborts the operation automatically.
- *Self-healing patch synthesis:* When a vulnerability is disclosed, the OS can generate a temporary micro-patch by mutating the offending code path in a live binary, guided by a reinforcement learning agent that tests the patch in an isolated replica before committing it.

The net effect? A reduction in successful breach rates by an order of magnitude and a shift in security budgets from "firefighting" to "risk prediction."

CONCLUSION

Intelligent OS represents a paradigm shift in which the operating system ceases to be a passive substrate and becomes an active participant in the computing ecosystem. The NeuroKernel prototype validated that embedding perception-action loops directly into the kernel is both feasible and beneficial, delivering measurable gains in latency, energy consumption, and resilience. Crucially, the layered architecture proposed in this study isolates learning logic from safety-critical kernel code, preserving the deterministic guarantees required for production deployments while still reaping the adaptability of AI.

Several research avenues are beckoning. First, expanding the decision-making fabric to support multi-objective optimization (e.g., balancing performance, security, and privacy) will enable more nuanced tradeoffs. Second, integrating federated learning across device clusters can accelerate model convergence without compromising data locality. Third, formal verification of adaptive enforcement pathways is essential to satisfy regulatory and safety standards in domains such as autonomous vehicles and medical devices. In summary, the convergence of intelligent control and operating system fundamentals heralds a new era in which computers not only execute instructions but also understand the context in which they operate.

REFERENCES

1. Cámara J, de Lemos R. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Zurich, Switzerland. 2012. p. 53–62. doi:10.1109/SEAMS.2012.6224391.
2. Joshi A, Segireddy A. Adaptive runtime security for Kubernetes using eBPF telemetry and machine learning. 2026 6th International Conference on Expert Clouds and Applications (ICOECA), Bengaluru, India. 2026; Bengaluru, India. p. 1663–1668. doi:10.1109/ICOECA68095.2026.11485567.
3. Zhang Y, Kang Q, Yu WY, Gong H, Fu X, Han X, Zhong T, Ma C. Decision information meets large language models: the future of explainable operations research [preprint]. 2025. arXiv:2502.09994. doi:10.48550/arXiv.2502.09994.
4. Dwork C, Roth A. The algorithmic foundations of differential privacy. Found Trends Theor Comput Sci. 2014;9(3–4):211–407. doi:10.1561/0400000042.
5. Kephart JO, Chess DM. The vision of autonomic computing. Computer. 2003;36(1):41–50. doi:10.1109/MC.2003.1160055.
6. Li X, Zhou T, Wang H, Lin M. Energy-efficient computation with DVFS using deep reinforcement learning for multi-task systems in edge computing. IEEE Trans Sustain Comput. 2025;10(6):1116–1127. doi:10.1109/TSUSC.2025.3593971.
7. Li J, Jiang W, He Y, Yang Q, Gao A, Ha Y, Özcan E, Bai R, Cui T, Yu H. FiDRL: flexible invocation-based deep reinforcement learning for DVFS scheduling in embedded systems. IEEE Trans Comput. 2025;74(1):71–85. doi:10.1109/TC.2024.3465933.
8. Zhong Z, Xu M, Rodriguez MA, Xu CZ, Buyya R. Machine learning-based orchestration of containers: a taxonomy and future directions. ACM Comput Surv. 2022;54(10s):1–35. doi:10.1145/3510415.
9. Ye Z, Gao W, Hu Q, Sun P, Wang X, Luo Y, Zhang T, Wen Y. Deep learning workload scheduling in GPU datacenters: a survey. ACM Comput Surv. 2024;56(6):1–38. doi:10.1145/3638757.
10. Shin DJ, Kim JJ. Deep reinforcement learning-based network routing technology for data recovery in exa-scale cloud distributed clustering systems. Appl Sci. 2021;11(18):8727. doi:10.3390/app11188727.
11. Russell SJ, Norvig P. Artificial Intelligence: A Modern Approach. 4th global ed. Harlow (UK):

Pearson Education Limited; 2022.
12. Faruki P, Bhan R, Jain V, Bhatia S, El Madhoun N, Pamula R. A survey and evaluation of Android-based malware evasion techniques and detection frameworks. Information. 2023;14(7):374. doi:10.3390/info14070374.
13. Liu X, Wang S, Ma Y, Zhang Y, Mei Q, Liu Y, Huang G. Operating systems for resource-adaptive intelligent software: challenges and opportunities. ACM Trans Internet Technol. 2021;21(2):27:1–27:19. doi:10.1145/3425866.

