Embedded Neural Firmware

ENF Technical Note 02

Related Work: TinyML Toolchains, Secure Boot, Energy Harvesting, and Formal Verification for Sealed Neural Firmware

— **Danish Z. Khan**, Founder, ENFSystems LLC

Version 1.0 · December 2025 · Status: Literature Survey / Design Context

Table of Contents

Abstract

This technical note surveys the most relevant prior work and standards that shape the design space for Embedded Neural Firmware (ENF). It reviews the TinyML ecosystem and its prevailing assumptions (runtime interpreters, flexible toolchains, and operational updateability), then contrasts these with ENF's sealed, deterministic "compile-and-flash" model. The note summarizes secure-boot and device-identity approaches (including PUF-rooted trust), and examines energy-harvesting constraints that motivate voltage-threshold gating and burst-mode inference. It also outlines formal verification methods for bounded neural networks (e.g., SMT-based and abstract-interpretation approaches) and discusses why ENF-style architectural constraints can make verification more tractable. Finally, it highlights empirical results from adjacent domains that support ENF's core claims: firmware-class inference is feasible, diagnostics can be non-invasive, and strong guarantees require minimizing runtime variability. Overall, the purpose of this note is to position ENF precisely within existing literature—by identifying what exists, what is missing, and what ENF intentionally forbids.

1. Embedded & TinyML AI

1.1 Introduction

The Embedded Neural Firmware (ENF) paradigm diverges sharply from the prevailing TinyML approaches, which emphasize lightweight machine learning (ML) inference on microcontrollers for broad application classes. While TinyML represents a significant stride toward enabling ML on low-power, resource-constrained devices, its assumptions about updatability, network dependency, and general-purpose toolchains pose barriers for sealed, deterministic, task-bounded systems like ENF. This subsection contextualizes ENF within the broader landscape of embedded inference, emphasizing its departure from the dominant TinyML design philosophy.

1.2 TinyML Landscape and Its Assumptions

TinyML is broadly defined as the deployment of machine learning models on ultra-low-power microcontrollers, often below 1 mW power consumption, with applications in speech, vision, anomaly detection, and sensor fusion (Kallimani et al., 2024, p. 29016). Key platforms supporting TinyML include TensorFlow Lite for Microcontrollers (TFLite Micro), CMSIS-NN, Edge Impulse, and proprietary toolchains like NanoEdge AI Studio (Kallimani et al., 2024, p. 29023). These frameworks typically support dynamic workflows involving model retraining, over-the-air (OTA) updates, and heterogeneous sensor configurations.

This design logic is underpinned by assumptions about periodic connectivity to cloud resources, ongoing model refinement, and the ability to manage model updates across a distributed fleet. As a result, TinyML systems are often benchmarked on their capacity for flexible deployment, dynamic inference scenarios, and general-purpose toolchain compatibility (Kallimani et al., 2024, p. 29021).

1.3 ENF's Departures from the TinyML Model

In contrast, ENF adopts a radically constrained architecture built around the following premises:

1. **Task-Bounded Finality**: ENF models are embedded as immutable logic units performing a singular, pre-defined task for the device’s operational lifetime.
2. **No OTA Updates**: The firmware, including the embedded model, is sealed at manufacture time, with no support for OTA updates or cloud-side retraining.
3. **Minimal Toolchain Footprint**: ENF avoids complex toolchains and runtime interpreters, favoring static compilation and direct flashing of inference kernels onto hardware.
4. **No Operating System or Scheduler**: There is no RTOS or user-space code; inference is performed by deterministic scheduling within a monolithic binary.

These constraints position ENF outside the normative goals of TinyML. Where TinyML seeks generality, flexibility, and continuous learning, ENF aims for finality, minimalism, and deterministic repeatability.

Compiler Flow and Toolchain Divergence

TinyML workflows typically utilize high-level model design in TensorFlow or PyTorch, conversion to TFLite or ONNX formats, quantization (usually post-training), and deployment using runtime interpreters such as TFLite Micro (Banbury et al., 2021). CMSIS-NN, for instance, provides optimized kernels for ARM Cortex-M architectures but still presumes integration into OS-level scheduling or broader ML pipelines (Johnny & Knutsson, 2021).

ENF, however, emphasizes a one-time compile-and-flash model pipeline. Quantization is applied statically, often pre-training or via fully static calibration. Model compilation produces pure C/C++ inference kernels, which are directly compiled into firmware alongside the hardware abstraction layer (HAL). There is no runtime interpreter or bytecode.

This has implications for reproducibility, auditability, and security:

- **Reproducibility**: Every ENF build corresponds to a single source-of-truth model; there is no stochasticity in inference behavior across devices.
- **Auditability**: The absence of runtime interpreters eliminates variable execution paths, allowing full formal verification of model execution flow.
- **Security**: By removing OTA, interpreter stacks, and file systems, the attack surface is drastically reduced.

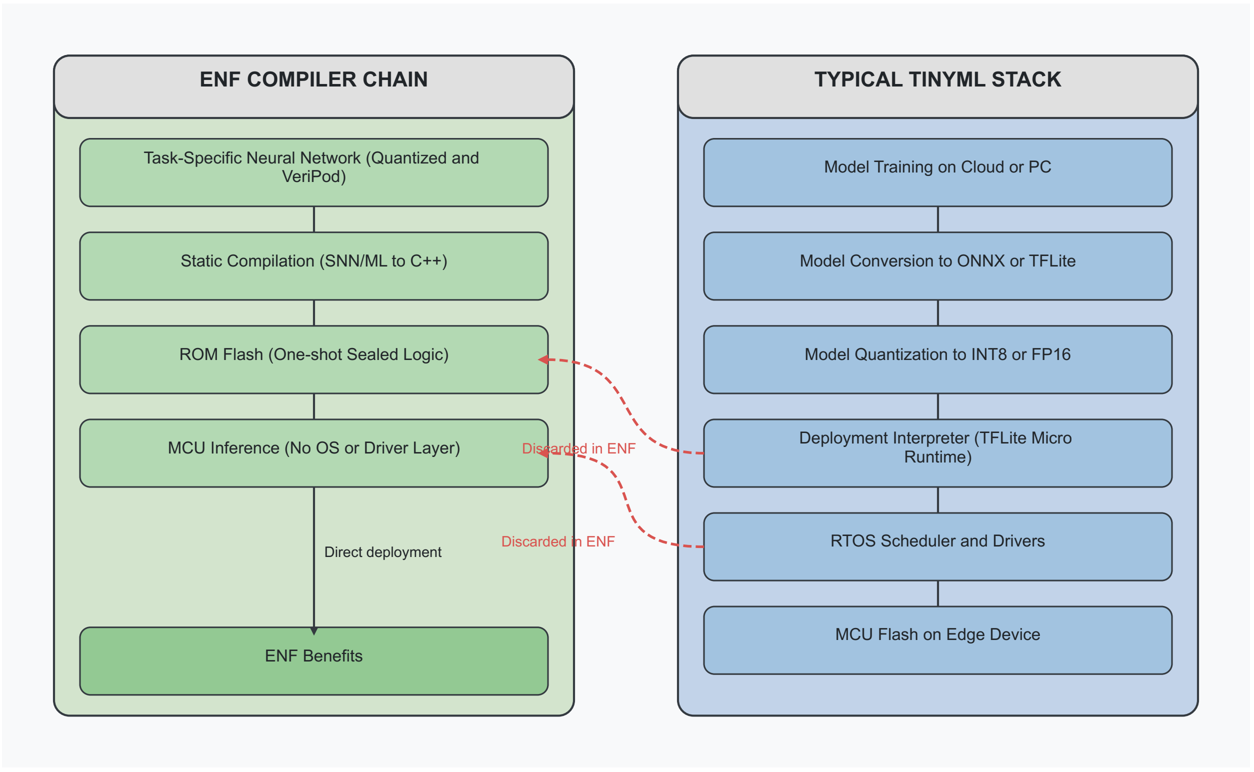
ENF COMPILER CHAIN
Task-Specific Neural Network (Quantized and VeriPod)
Static Compilation (SNN/ML to C++)
ROM Flash (One-shot Sealed Logic)
MCU Inference (No OS or Driver Layer)
Direct deployment
ENF Benefits
TYPICAL TINYML STACK
Model Training on Cloud or PC
Model Conversion to ONNX or TFLite
Model Quantization to INT8 or FP16
Deployment Interpreter (TFLite Micro Runtime)
RTOS Scheduler and Drivers
MCU Flash on Edge Device
Discarded in ENF
Discarded in ENF

Figure 1: Comparison of the typical TinyML deployment stack versus the ENF compiler chain. ENF eliminates all runtime layers—interpreter, RTOS, and drivers—by compiling task-specific neural agents directly into sealed ROM logic. This results in a static, OTA-free, and OS-independent execution path.

(Adapted from: TinyML_Tools_Applications_Challenges_and_Future_Research_Directions.pdf, p. 29024)

1.4 Quantization and Determinism

Quantization is essential to both TinyML and ENF, but with differing emphasis. TinyML typically employs 8-bit post-training quantization (PTQ) to reduce memory and compute load while tolerating modest accuracy degradation. Advanced workflows support quantization-aware training (QAT), pruning, and structured sparsity (Kallimani et al., 2024, p. 29026).

ENF applies quantization as a first-class design principle. Models are trained under strict constraints (e.g., INT8-only, fixed-point arithmetic, no dynamic range scaling), ensuring deterministic inference on fixed-point ALUs. This hard quantization is not a downstream optimization but a functional prerequisite for hardware-bonded inference.

Table 1: Memory and energy constraints across representative TinyML-capable microcontrollers
ENF's deterministic INT8 models are architected to operate within these conservative limits, enabling sealed inference without OS or runtime.
(Source: TinyML_Progress_and_Futures.pdf, p. 4)

System	MCU Model	Dataset	Data Bitwidth	Latency (ms)	Peak Memory	Flash Usage	Energy (mJ)
CMSIS-NN	STM32H743	ImageNet	INT8	510	< 1 MB	1.4 MB	135
X-Cube-AI	STM32H743	ImageNet	INT8	437	< 1 MB	1.4 MB	115
MicroNets-VWW[1]	STM32F746	VWW	INT8	1133	285 KB	0.8 MB	479
TF-Lite Micro	STM32F746	ImageNet	INT8	296	211 KB	< 1 MB	—[2]
MCUNetV2	STM32H743	ImageNet	INT8	859	434 KB	1.8 MB	546
TinyMaix	STM32H750	VWW	Mixed	64	54 KB	0.2 MB	—[2]

[1] *MicroNets-VWW is reported under the "TinyEngine" configuration from MLSys'21.*
[2] *Energy figure not reported in the original source.*

1.6 Toolchain Limitations and Architectural Implications

ENF's model lifecycle precludes many standard TinyML practices. There is no support for frameworks requiring dynamic memory allocation, interpreter stubs, or filesystem access. As such, toolchains must:

- Emit fully linked binaries without dynamic dependencies
- Support per-layer fixed-shape memory layouts
- Be verifiable via formal methods or static analysis

These constraints disqualify many otherwise powerful frameworks from ENF use. For example:

- **TFLite Micro**: Relies on a runtime interpreter and metadata headers
- **CMSIS-NN**: Requires an RTOS or HAL glue layer with dynamic allocation
- **MCUNet**: Assumes cloud-trained models with runtime shaping and OTA readiness

Instead, ENF toolchains resemble those used in safety-critical embedded control (e.g., DO-178C or ISO 26262 environments) where all logic must be statically defined, testable, and functionally frozen at compile time.

Framework	Runtime Component	OTA Support	Compilation Flow	Model Storage	Execution Environment
TFLite Micro	Interpreter	Optional	Host compile + MCU flash	Flash/RAM	Stack-managed RTOS
Edge Impulse	SDK runtime	Yes	Cloud compile + OTA	Cloud + Flash	Event loop/RTOS
CMSIS-NN	No runtime	No	Direct compile	Flash	Bare metal or RTOS
MCUNet	Model packager	Optional	Auto-compiled	Flash	RTOS + autotuner
ENF (Proposed)	None	No	Statically linked only	ROM	No RTOS, no abstraction

Table 2: Comparison of toolchain architecture and runtime assumptions in mainstream TinyML frameworks. ENF omits all interpreter, OTA, and runtime components in favor of statically compiled firmware. (Source: Kallimani et al. (2023), Table 1 (p. 29017))

1.7 Conclusion

While ENF and TinyML both target low-power embedded intelligence, their philosophical and technical approaches diverge fundamentally. TinyML emphasizes runtime flexibility, OTA adaptability, and general-purpose tooling.

“There is currently no consensus runtime or descriptor format for modular, task-isolated AI deployment on embedded devices.”
— Kallimani et al. (2023, p. 29041)

This fragmentation underscores the systemic absence of a deployment model for static, verifiable neural firmware. ENF, by contrast, commits to sealed, offline, and task-specific inference units, compiled into the device firmware as static logic. This departure enables formal verification, lifespan determinism, and a radically reduced attack surface, but requires abandoning conventional TinyML toolchains in favor of bespoke minimal flows grounded in finality and reproducibility.

2. Secure Firmware and Boot Processes

Embedded Neural Firmware (ENF) mandates a sealed, immutable boot environment anchored in physical device identity rather than software credentials. This section defines ENF's tamper-resistant boot model through the integration of Physical Unclonable Functions (PUFs), immutable flash programming, and the complete exclusion of remote Over-the-Air (OTA) update mechanisms.

2.1 PUF-Bound Trust and Device Identity

ENF’s foundational trust model binds neural firmware execution to a unique hardware identity using PUFs. As Herder et al. (2014) describe, PUFs leverage intrinsic manufacturing randomness to produce a unique, unclonable response to a given challenge, without requiring stored cryptographic keys (p. 1127). These responses, derived from physical delay variations or electrical noise, are irreproducible across even identically fabricated dies.

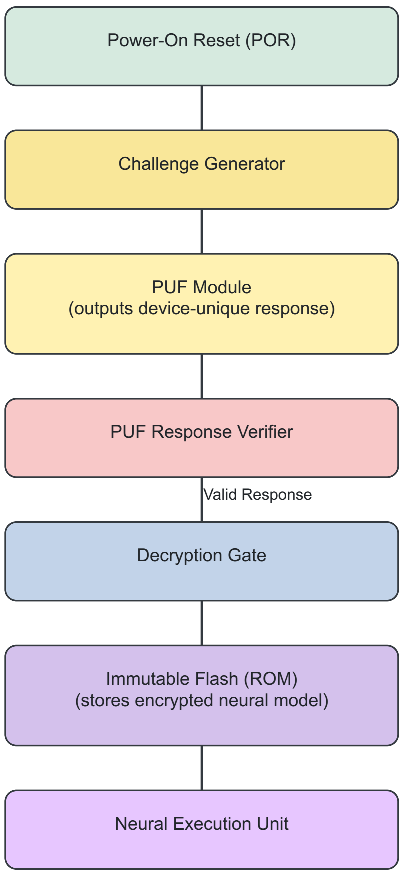

Figure 2: ENF secure boot sequence anchored in PUF challenge-response logic. A fixed challenge is issued on reset; only upon successful device-unique verification is the encrypted neural model decrypted and executed.
(Derived from Herder et al., 2014 and internal ENF Section 2 Notes)

By embedding a challenge–response PUF protocol into the ENF boot sequence, firmware becomes inseparable from the physical substrate. A bootloader verifies the PUF response before decrypting or executing any logic, creating a hardware-anchored trust loop. This satisfies assurance requirements under ISO 26262 and IEC 61508 for runtime authenticity without relying on mutable state or update mechanisms (NXP, 2017, p. 12).

Unlike software-centric security models such as Trusted Platform Modules (TPMs) or Secure Enclaves—which depend on non-volatile memory and credential management—PUFs provide zero-persistence, passive trust anchors that require no update infrastructure or secure storage. This approach aligns with ENF's philosophy of sealed, logic-only inference systems.

2.2 Immutable Flash and OTA Exclusion

Firmware mutability has been identified as a primary threat vector in modern IoT attacks (Verizon, 2024, p. 26). ENF mitigates this by sealing firmware at manufacture time into One-Time Programmable (OTP) or lock-fused flash memory. There is no writable boot partition, update agent, or network-facing interface, eliminating entire classes of remote code execution, rollback, and side-channel exploits (IoTSF, 2021, p. 19).

The IoT Security Assurance Framework mandates the disabling of debug ports, removal of unused interfaces, and isolation of memory regions storing cryptographic logic (IoTSF, 2021, Sec. 2.4.4). ENF complies at a minimum of Assurance Class 3.

This design fundamentally diverges from conventional secure boot schemes, which often support digitally signed OTA firmware replacement. ENF explicitly rejects such mechanisms in favor of absolute finality. As outlined in the IoTSF framework, user-managed update processes introduce unacceptable variability and lifecycle risk (p. 23). ENF guarantees immutability post-manufacture—logic that cannot be modified, replaced, or augmented.

2.3 Root of Trust and System Finality

Traditional secure boot processes rely on a hardware root-of-trust—typically a ROM-embedded hash or public key—to verify successive boot stages. ENF builds on this concept by integrating irreversible flash locking and PUF gating. According to IEC 61508 guidelines, such irreversible boot sequences enable SIL-classifiable logic behavior, which must remain auditable and deterministic even in the presence of faults (NXP, 2017, p. 25).

Combined with watchdog resets and physical access restrictions, ENF's root-of-trust architecture ensures that any tampering results in device nullification, not degraded functionality. In ENF, failure results in shutdown—never fallback. Furthermore, PUF-gated boot sequences satisfy ISO 26262 requirements for safety-related functionality to be both uniquely instantiated and statically verifiable at boot time. This allows for formal ASIL decomposition and enables tool qualification pathways under ISO 26262 Part 8 for embedded inference compilers and verification toolchains.

Conceptual Boot Sequence:

[Power-On Reset] → [PUF Challenge] → [PUF Response Verification] → [Firmware Unlock] → [Static Inference Execution]

This linear boot model ensures deterministic startup and eliminates alternate or degraded operating states.

2.4 Firmware Threat Vectors and Attack Surface Reduction

According to the 2024 Verizon DBIR, firmware vulnerabilities are now more prevalent than application-level ones in embedded systems. “Firmware now constitutes the dominant attack vector in embedded devices, surpassing OS and application-layer exploits” (Verizon, 2024, p. 28). ENF addresses this by drastically minimizing its attack surface: it removes dynamic code paths, excludes file systems and update agents, and statically embeds model weights into ROM.

Aligned with ISO 26262 decomposition principles, no ENF software or hardware module is permitted to exceed its isolated safety boundary. PUF-bound identity further ensures that cloning, mirroring, or emulator replication is cryptographically infeasible.

Firmware Attack Vectors vs. ENF Exclusions

Threat Vector	**Conventional IoT Firmware**	**ENF Countermeasure**
OTA update spoofing	Common	Firmware sealed in OTP flash
Debug interface misuse	Often exposed	Disabled or fused during manufacture
Remote code execution via network	Includes TCP/IP stack	No IP stack; no runtime interpreter
File system injection	Present via RTOS abstraction	No file system; all logic precompiled to ROM
Device cloning or mirroring	Based on MAC/IP identifiers	Bound cryptographically via PUF challenge/resp.

Table 3: Comparison of common firmware-level threat vectors and ENF’s deterministic, sealed hardware countermeasures.
(Source: Derived from Verizon DBIR 2024 and Section 2 Notes)

2.5 Summary

ENF's secure boot is not merely a feature—it is an architectural guarantee. By binding firmware to the physical silicon through PUFs, eliminating all update vectors, and sealing system logic into immutable flash, ENF achieves a zero-trust, zero-maintenance security posture. This approach satisfies Class 3–4 IoTSF assurance requirements, supports SIL-level decomposition under ISO 26262, and responds directly to the escalating threat profile surrounding firmware integrity. In ENF, firmware is not updatable—it is final.

3. Energy Harvesting in IoT Devices

Embedded Neural Firmware (ENF) is engineered to operate without wired power or battery replacement. Its deployment model is anchored in ultra-low-power computation, sustained entirely by ambient energy sources such as vibration, light, and thermal gradients. This section grounds ENF's energy assumptions in empirical harvesting data and outlines the architectural constraints necessary to support sub-milliwatt inference.

3.1 Energy Harvesting Modalities

ENF-compliant systems must function within the tight energy budgets provided by passive environmental sources. The three principal harvesting modalities include:

- **Piezoelectric Vibration Harvesting**: Converts mechanical strain into electrical energy, especially effective in resonant environments like rotating machinery or vibrating infrastructure. Power densities can reach 2–4 mW/cm² when tuned to ambient vibration modes (Khaligh & Zeng, 2013, p. 6).
- **Photovoltaic (Indoor Light)**: Utilizes ambient lighting with typical output below 10 μW/cm², depending on spectral alignment and light angle (Khaligh & Zeng, 2013, p. 5).
- **Thermoelectric Generators (TEGs)**: Extract usable energy from temperature gradients, such as those found across HVAC ducts. These systems can yield tens of microwatts when optimized for thermal impedance (Khaligh & Zeng, 2013, p. 7).

Each modality presents unique constraints in terms of intermittency, predictability, and power regulation.

Harvesting Modality	Power Density	Notes
Piezoelectric (vibration)	2–4 mW/cm² (resonant)	Requires tuning to mechanical frequency
Indoor Solar	<10 μW/cm²	Highly dependent on spectral match
TEG (thermal gradients)	20–60 μW/cm²	Requires ~5°C temperature delta

Table 4: Comparative power density metrics for ambient energy sources relevant to ENF deployments. (Source: Energy_Harvesting_in_Buildings_Review.pdf; Matiko et al., 2014)

3.2 Storage and Power Management Constraints

In light of the intermittent and ultra-low-current profile of harvested energy sources, Embedded Neural Firmware (ENF) systems must implement passive energy buffering and deterministic activation logic to guarantee safe and reliable inference cycles. This section outlines the electrical model, architectural strategies, and design commitments required to meet these constraints.

Core Components for Deterministic Energy Buffering

To operate within the tight power budgets of ambient energy environments, ENF hardware incorporates the following subsystems:

- **Supercapacitors or Thin-Film Batteries**: Act as intermediate energy reservoirs, buffering fluctuations in input current and enabling stable voltage accumulation over time.
- **Cold-Start DC-DC Converters**: Must be capable of initiating operation at input voltages below 100 mV, facilitating bootstrapping from near-zero energy states.
- **Schmitt-Triggered Wake Logic**: Ensures that neural inference is gated by a static voltage threshold. The system remains dormant until energy storage exceeds this threshold (typically 0.5 V), thereby avoiding unsafe execution states due to brownout conditions.

RC Charging Model: Energy Accumulation Over Time

The behavior of the energy storage element is governed by the RC charging equation:

$$V(t) = V_{max} \cdot \left(1 - e^{\frac{-t}{\tau}}\right)$$

Where:

- *V(t)* : Voltage across the capacitor at time *t*
- V_{max} : Terminal voltage of the energy source
- *R* : Equivalent series resistance of the harvester and circuit
- *C* : Capacitance of the energy buffer
- $\tau = R \cdot C$: System time constant

This model defines an exponential voltage increase, which enables a predictable, hardware-bound activation point for inference. A Schmitt-trigger comparator monitors *V(t)* and initiates execution once *V(t)* and initiates execution once $V(t) \geq V_{th}$, where V_{th} is the cold-start activation voltage.

Simulation and Threshold Windowing

A representative simulation of the RC charging process is shown below.

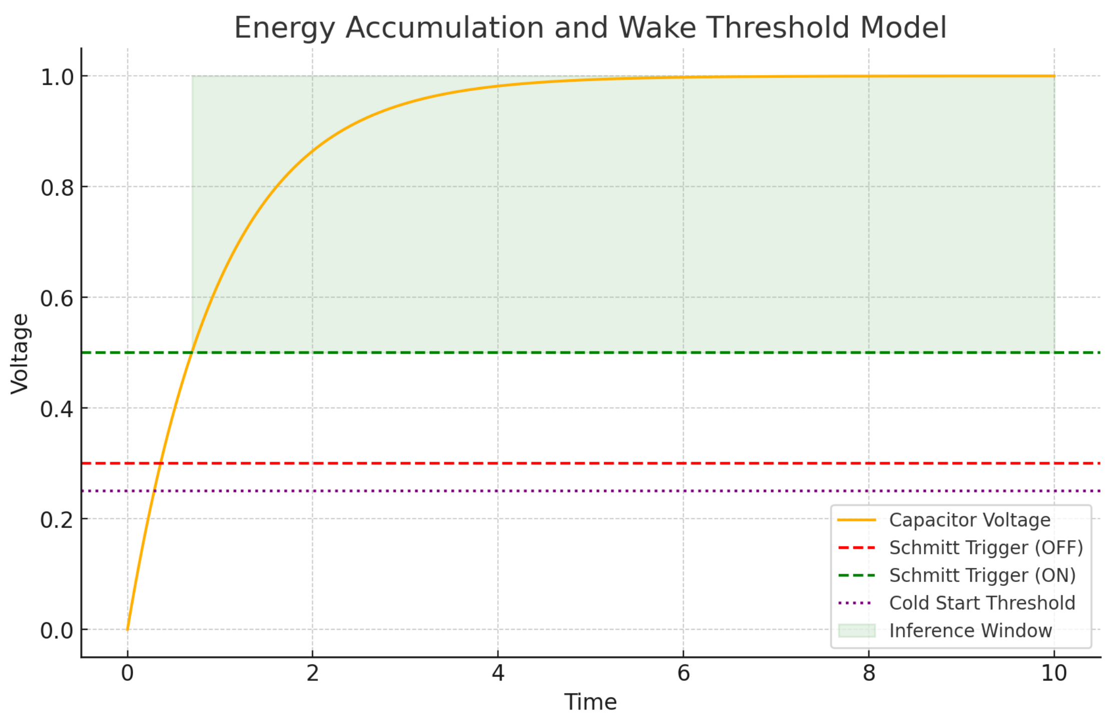

Figure 3: Voltage accumulation over time. Schmitt-trigger thresholds define OFF/ON logic states. The green shaded region denotes the safe inference window.

A detailed data file is available in Appendix A (CSV: ENF_RC_Curve_Data.csv).

The capacitor voltage curve demonstrates a deterministic intersection with the Schmitt ON threshold (e.g., 0.5 V), at which point inference execution is permitted. This ensures inference only proceeds when sufficient energy has been stored to complete execution safely, thereby avoiding mid-operation brownouts.

The energy available at a given voltage is calculated as:

$$E = \frac{1}{2} C V^2$$

Where:

- *E*: Energy stored in the capacitor (in joules)
- *C*: Capacitance in farads
- *V*: Instantaneous voltage across the capacitor

This equation defines the usable energy budget available to the ENF inference engine at a given charge level. Since the execution of an inference task consumes a finite, quantifiable amount of energy, this relationship directly governs model size, inference duration, and capacitor selection. The energy threshold ensures that the system never initiates processing unless sufficient energy is available to complete the operation safely, thus eliminating brownouts and incomplete computation.

This relationship informs both capacitor sizing and the permissible model energy budget.

ENF Design Commitments: Energy-Aware Determinism

To preserve deterministic behavior under constrained energy budgets, ENF systems adhere to the following architectural commitments:

- **Static Wake-Sleep Scheduling**: No event loops, dynamic timers, or interrupts. Gating is performed entirely via analog voltage thresholds.
- **Elimination of Predictive Power Models**: ENF does not support adaptive scaling or runtime energy estimation.
- **Hardwired Activation Parameters**: Comparator thresholds and energy buffer configurations are set at design time and fixed throughout the device's lifecycle.

Representative Electrical Parameters

Parameter	Typical Value
Cold-Start Threshold (V_{th})	0.5 V
Nominal Operating Voltage	1.8–3.3 V
Time Constant (τ = RC)	1–10 s
Inference Energy Budget	1–2 mJ

Conclusion

This power management strategy supports ENF's mission of long-duration, zero-maintenance embedded intelligence. By aligning energy storage, wake logic, and execution within a closed-loop, hardware-defined energy model, ENF systems ensure predictable and safe operation across decades of deployment in energy-scarce environments.

3.3 Predictability in Harvesting Environments

ENF targets environments with stable and predictable energy availability, such as indoor lighting, industrial vibration zones, or thermal gradients across HVAC systems. This predictability supports static duty-cycle design and eliminates the need for dynamic energy adaptation.

Event-triggered inference is viable when:

- The energy source exhibits periodic or cyclic behavior.
- Capacitor recharge intervals align with model execution timing.
- Inference completes before supply voltage drops below functional thresholds.

These properties enable ENF devices to perform reliably without runtime power negotiation or scaling. ENF explicitly targets stable indoor environments with low temporal variance in energy profile, excluding outdoor, mobile, or highly dynamic applications.

3.4 Architectural Implications

ENF’s reliance on ambient energy imposes strict architectural constraints:

- **Model Size**: Inference must complete within 10–100 ms at ≤1 mW power draw.
- **Memory**: Only static RAM is permitted; no dynamic refresh or paging.
- **Activation Logic**: Execution is single-shot, triggered by energy accumulation rather than system clocks.

This design philosophy departs significantly from battery-powered TinyML systems, which can support background processing, idle polling, and RTOS scheduling. ENF’s energy-autonomous architecture mandates a minimalist, static hardware-software co-design. Dynamic voltage scaling, runtime PLL configuration, or energy-aware RTOS hooks are incompatible with ENF’s frozen, clockless execution topology.

In practice, ENF deployments may incorporate mechanical or event-based pre-charge triggers—such as piezo taps or thermal delta—to enable deterministic readiness without batteries or continuous polling. These maintain architectural consistency while improving real-world responsiveness.

3.5 Sidebar: Real-World Energy Guarantees in ENF Products

While ENF relies on ambient energy harvesting to preserve its battery-free, silent intelligence model, practical deployments are not energy-starved. ENF chips are embedded inside host products that already experience environmental energy inputs as part of their function.

Key insight:

“ENF is not floating in the environment. It is encased inside things humans already touch, activate, or heat.”

Host-Product Energy Coupling:

Product Type	Inherent Energy Source	Harvested By ENF
Motion-sensor tap	User proximity, hand warmth, pressure	Piezo + thermal
Smart light switch	Line flicker, hand motion, magnetic flux	PV + induction
Trash bin	Lid motion, hand movement, room light	Piezo + PV
Cooktop sensor	Heat gradient, knob movement	Thermoelectric

ENF’s capacitor bank is typically charged by **multiple overlapping harvesting methods**, guaranteeing inference-readiness in practical use.

Engineering Strategy: Redundant Energy Paths

- Combine **two or more modalities per product**
- Design **supercap banks for burst-mode inference**
- Use **Schmitt-trigger logic to avoid premature execution**
- Include **FSM-based wake gating** for precision without leakage

Latency vs. Readiness:

ENF is **not always-on** — it is **always-deterministic**.

Execution only occurs when the model can complete with certainty. This may introduce brief delays (~0.3–1s), which are:

- Acceptable for human-driven interfaces
- Prevent unsafe partial inference or memory corruption

Result:
ENF remains always-trustworthy — even when momentarily inactive.

3.6 Summary

ENF's architecture is built around energy neutrality. By synchronizing inference workloads with predictable energy harvesting conditions and enforcing static, duty-cycled activation, ENF systems operate without batteries, chargers, or human intervention. Empirical benchmarks from solar, piezoelectric, and thermoelectric harvesters validate this design at ultra-low-power levels. As a result, ENF devices can operate maintenance-free for 30–50 years in sealed environments, constrained only by environmental variability—not external power infrastructure.

4. Formal Verification for Neural Networks

Embedded Neural Firmware (ENF) mandates a deterministically verifiable execution model explicitly designed to meet the demands of high-assurance systems. This section outlines the application of formal verification techniques—specifically satisfiability modulo theory (SMT)-based solvers such as Reluplex and ERAN—to certify the correctness and bounded behavior of embedded neural networks. These methods align ENF with the requirements of safety certification frameworks, including IEC 61508 at Safety Integrity Level 3 (SIL 3).. This section outlines the application of formal verification techniques—specifically SMT-based solvers like Reluplex and ERAN—to certify the correctness of embedded neural networks. These methods align ENF with safety certification frameworks such as IEC 61508 at Safety Integrity Level 3 (SIL 3).

4.1 Bounded Neural Inference and Formal Verification

ENF systems are designed to ensure that all inference results remain within strictly defined and statically verifiable input-output boundaries. This is achieved through architectural constraints including:

- **Statically bounded input domains**, such as clamped analog-to-digital sensor ranges
- **Fully quantized weights** using fixed-point arithmetic (e.g., int8)
- **Acyclic feedforward execution graphs**, free from dropout, loops, or stochastic behavior

These restrictions enable the use of satisfiability modulo theory (SMT) tools for verification. Reluplex—a hybrid solver combining linear programming and symbolic reasoning—was developed to verify ReLU-activated neural networks under precisely bounded conditions (Katz et al., 2017). The core verification question is: *"$\forall x \in [x_{min}, x_{max}]$, y satisfies the safety property P?"* Reluplex addresses this by systematically exploring all activation paths within the defined input space.

To maintain tractable complexity, ENF models are deliberately constrained in depth and connectivity, preventing symbolic path explosion and enabling exhaustive verification within bounded time.

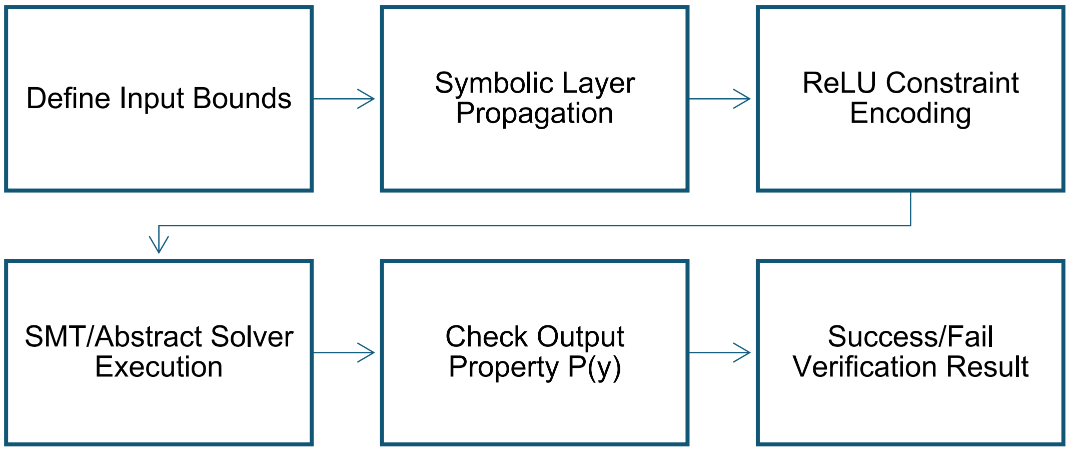
Define Input Bounds
Symbolic Layer Propagation
ReLU Constraint Encoding
SMT/Abstract Solver Execution
Check Output Property P(y)
Success/Fail Verification Result

Figure 4: SMT-based neural verification loop used in ENF model sealing. Each layer's activation constraints and input bounds are passed to the solver, which validates that output invariants are maintained under all bounded conditions.

(Source: Reluplex, ERAN)

4.2 ERAN and Soundness over Abstract Domains

Method	Coverage	Architecture Support	Soundness	Scalability	ENF-Compatibility
Reluplex	Full (SMT)	ReLU FFNs only	Yes	Medium	Yes
ERAN	Partial (overapprox.)	CNNs, FFNs, Quantized	Yes (safe)	High	Yes
Empirical Testing	Sparse	All	No	High	No

Table 5: Comparison of neural network validation techniques. ENF mandates sound formal guarantees over empirical approximations.

ERAN extends verification capabilities beyond SMT methods by incorporating abstract interpretation over zonotope and polyhedral domains. It supports a wider range of architectures typical of embedded inference (e.g., convolutional and fully connected layers), while ensuring sound over-approximations of reachable outputs.

Crucially, ERAN allows:

- Sound certification under quantized (int8) arithmetic
- Analysis of deep models without path enumeration
- Compatibility with MCU-scale architectures

Unlike empirical testing, ERAN produces provable safety bounds over entire input domains. Its integration within ENF's toolchain ensures that deployed models are functionally safe under all allowable input perturbations.

4.3 Functional Safety Certification: SIL 3 Alignment

ENF systems are designed for Safety Integrity Level 3 (SIL 3) compliance under IEC 61508. SIL 3 corresponds to a dangerous failure probability of 10^-7 to 10^-8 per hour of operation in high-demand scenarios (NXP, 2017, p. 33). Satisfying this requirement entails:

- Demonstrably deterministic NN behavior across all bounded inputs
- Robustness to hardware faults or input noise
- Formal tool qualification and traceable verification artifacts

As outlined in NXP's functional safety guidance, "neural networks are treated as systematic fault sources and must be mitigated at design time, not runtime" (NXP, 2017, p. 35). ENF fully adopts this posture by embedding all verification and safety constraints into the static model prior to compilation. Verification tools and compilers used in ENF must also satisfy Tool Confidence Levels (TCLs) as defined under ISO 26262 Part 8, ensuring that both model behavior and verification logic meet traceable, certifiable safety standards.

4.4 Certifiable Neural Network Architectures

To meet the formal verification and SIL certification requirements, ENF models are strictly constrained to:

- **ReLU-activated, acyclic, feedforward topologies**
- **Quantized inference (e.g., INT8-only)** with no floating-point computation
- **No runtime stochasticity**—e.g., no dropout, softmax, or data-dependent branching
- **Fixed memory layout and static computation graph**

These constraints allow the neural model to be treated as a fixed logical function, amenable to symbolic analysis. Verifiable properties include:

- Safety bounds on outputs for critical actuator signals
- Guaranteed absence of false activations in reserved input regions
- Monotonic response behavior or bounded Lipschitz continuity

Such architectural minimalism enables static compilation of the model into firmware and direct embedding into ROM, removing all runtime dependency or model variability. ENF models are compiled into logic structures equivalent to Boolean functions. Every activation path is static and testable.

4.5 Summary

ENF neural agents are not adaptive or cloud-linked—they are statically certifiable, bounded computational entities. Tools like Reluplex and ERAN enable full input-space verification of output constraints, transforming ENF models into provable control logic. These methods, guided by IEC 61508 and ISO 26262 frameworks, allow neural firmware to be formally approved for use in long-life, safety-critical deployments, bridging the gap between neural inference and industrial-grade functional safety.

5. Modular Intelligence Beyond RTOS: Static Composition and Biological Coordination

ENF systems reject the conventional paradigm of deploying monolithic neural networks atop real-time operating systems (RTOS) and instead favor a modular, stateless, biologically inspired coordination model. This section explores the architectural divergence between deterministic neural modules and RTOS-based frameworks. It draws analogies from biological systems—particularly the self-synchronizing flagellar fields observed in *Volvox carteri*—to propose a new model of localized, bounded neural behavior tailored to ENF's mission of secure, deterministic, and maintenance-free embedded cognition.

5.1 Limitations of RTOS-Centric Architectures

The IEEE (2018) standard defines RTOSs for small-scale embedded systems as kernel-driven software stacks that coordinate multiple tasks using preemptive scheduling, timers, and inter-process communication (IPC). Although effective for general-purpose embedded computing, this architecture introduces several issues in safety- and privacy-critical contexts:

- **Scheduler non-determinism** due to race conditions, shared resource contention, or unpredictable interrupts
- **Dependence on complex memory management**, including stacks and heaps that cannot be statically bounded
- **Security vulnerabilities** arising from IPC mechanisms, task privilege escalation, and dynamic linking

In contrast, ENF enforces a no-RTOS, static scheduling policy where each neural agent operates as an isolated, single-function circuit with no message-passing or shared memory (Heath, 2003).

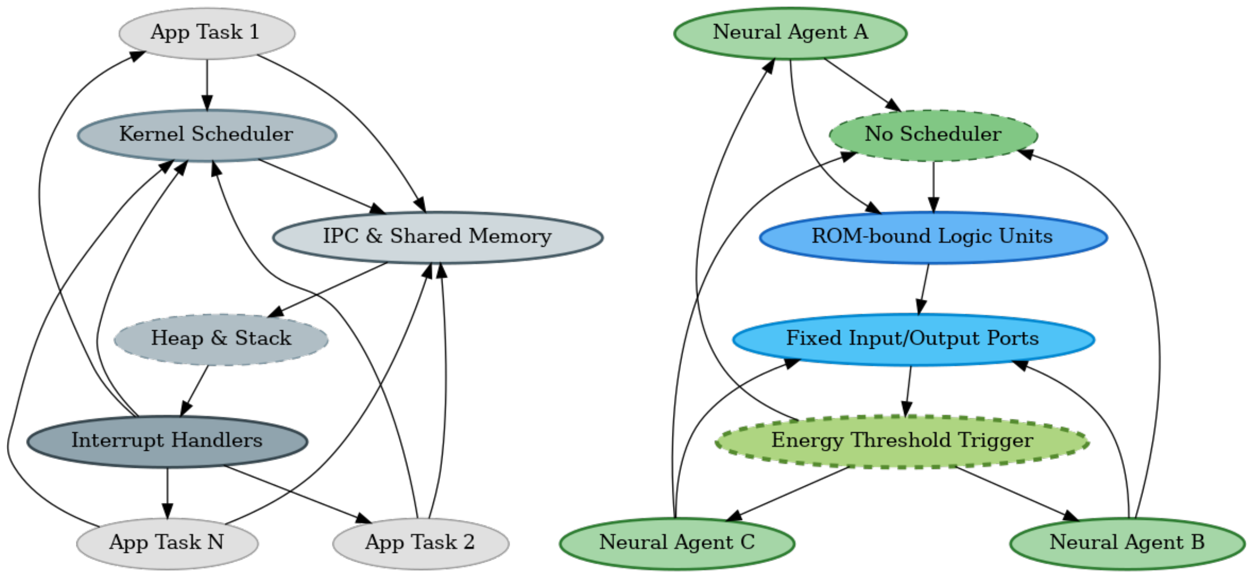

Figure 5. Execution architecture comparison: RTOS-based kernel stack vs. ENF's static neural logic.

The left model illustrates traditional RTOS execution using interrupt handlers, schedulers, and shared-memory IPC. The ENF model on the right replaces these with stateless, ROM-bound neural agents triggered by deterministic energy thresholds, eliminating all runtime variability and task contention.

(Source: Adapted from IEEE RTOS Standard 2050-2018)

5.2 Static Modularism in ENF: Task-Bounded Neural Agents

ENF organizes behavior through discrete, ROM-burned neural modules, each responsible for a single, fully bounded task (e.g., gesture classification, occupancy detection, pressure modulation). These modules:

- Are stateless across invocations and operate on fixed input/output ranges
- Require no cooperative multitasking or system clock alignment
- Are compiled and verified in isolation and composed via deterministic wiring at deployment

Each module behaves like a logic gate: it has defined truth boundaries (through quantized inference) and no capability to self-modify or interact with other agents at runtime. This eliminates the need for firmware schedulers, replacing them with a physical composition of stateless, functionally atomic behaviors.

Feature	RTOS Task	ENF Neural Agent
State Retention	Persistent (RAM, Stack)	Stateless
Scheduling	Dynamic, Preemptive	Static, Event-Triggered
IPC/Shared Memory	Common	Forbidden
Invocation Logic	Kernel or Interrupt	Energy-Threshold Activation
Deployment Unit	Binary Blob	ROM-Bound Inference Module

Table 6: Design comparison between RTOS software tasks and ENF logic-bound neural agents.

(Source: 2050-2018)

5.3 Distributed Coordination via Biophysical Analogy

A novel source of insight into ENF modular behavior arises from biological systems such as *Volvox carteri*. Studies by Mannan et al. (2020) have shown that synchronized metachronal waves among flagella emerge without central control, guided solely by local hydrodynamic interactions. Each flagellum behaves as an oscillator that adjusts its phase in response to fluid perturbations from neighbors.

This decentralized yet predictable coordination mechanism serves as a biologically grounded analogue for ENF modules:

- Each neural agent acts as a “rotor” with a fixed beat cycle (inference interval)
- No global clock or controller synchronizes behavior

- Local signal propagation and phase coupling enable system-wide emergent behavior

This synchronization is not time-aligned but phase-coupled, relying solely on proximity-induced feedback rather than global timing signals or phase controllers.

In ENF, this principle is realized through hardware-level signal thresholding and spatially localized I/O propagation. There is no global scheduler, bus arbiter, or master time signal—coordination arises purely from proximity and input relationships. Crucially, Volvox-like coordination is fault-tolerant: the failure or desynchronization of one rotor does not compromise the entire system.

5.4 Absence of Modular Neural Composability in Current Literature

Despite the rapid growth of TinyML and edge AI, existing toolchains rarely support statically composed, ROM-linked neural modules without RTOS middleware or cloud dependency. As noted by Kallimani et al. (2023), toolchains such as TFLite Micro and Edge Impulse prioritize runtime abstraction, dynamic model loading, and OTA provisioning—all of which ENF explicitly excludes.

ENF's model composability does not rely on software APIs or runtime layers, but instead derives from:

- Pre-verified and statically quantized neural function blocks
- Deterministic glue logic embedded in silicon
- Physical layout encoding signal flow and task boundaries

This absence of ROM-bound, stateless neural modularity reflects a critical gap in current embedded AI frameworks—one which ENF directly addresses by treating neural firmware as logic circuitry rather than adaptive software. ENF introduces the concept of a hardware descriptor format (ENFML) to define, verify, and bind task-specific neural modules at compile time—filling a gap in post-RTOS AI deployment semantics.

5.5 Summary

ENF abandons the RTOS hierarchy in favor of deterministic, single-function neural agents that mirror biological coordination mechanisms rather than software control theory. By modeling inference modules as rotor-like systems—bounded in function, phase-aligned by local interaction, and sealed against runtime variability—ENF constructs a new paradigm for modular embedded intelligence. This architecture remains unaddressed by contemporary TinyML frameworks, positioning ENF as a post-RTOS model grounded in physical determinism and long-term reliability.

6. Empirical Validation and Verification of Embedded Neural Firmware Design Assumptions

The Embedded Neural Firmware (ENF) architecture is grounded in strict design assumptions centered on inference locality, determinism, and hardware anchoring. While ENF diverges from mainstream embedded AI frameworks in both philosophy and implementation, several recent empirical studies offer strong validation for its foundational commitments—demonstrating the feasibility of sealed inference, analog-bound diagnostics, static adaptation, and behaviorally attested firmware execution. These empirical validations, combined with a broader review of emerging trends in secure AI, OTA-free execution, and energy-hardened logic, reinforce ENF's foundational tenets.

6.1 Firmware-Level Inference with Temporal Models

Aad et al. (2023) present the direct implementation of a recurrent neural network (RNN) in firmware for the ATLAS liquid argon calorimeter upgrade at CERN. Executing on Stratix 10 FPGAs, their design enables inference on energy deposition events with sub-125 ns latency, without reliance on operating systems or external orchestration. While their work targets high-throughput physics instrumentation rather than ultra-low-power silicon, it empirically validates ENF's assertion that temporal models can operate at the firmware level with deterministic timing and bounded memory—reinforcing the plausibility of sealed, task-bound inference in radiation-hardened or safety-critical domains.

6.2 Constrained Learning Under Firmware Limitations

Shaji (2023) investigates how microcontroller-class systems can support neural network training under extreme constraints, including fixed memory layouts, energy restrictions, and real-time execution windows. Although ENF prohibits runtime training to preserve auditability and task finality, this study affirms the latent capacity of embedded hardware to support bounded learning modes. ENF extends this insight through compile-time model switching and formally bounded deterministic polymorphism—allowing frozen model banks or FSM-controlled adaptation logic to operate without modifying neural weights post-deployment.

6.3 Behavioral Firmware Fingerprinting with Graph Embeddings

Li et al. (2021) propose the use of graph neural networks (GNNs) to generate structural embeddings of firmware logic for behavioral fingerprinting. Their work enables fine-grained classification of firmware binaries without reliance on cryptographic hashes, offering a method to verify execution paths through learned structural signatures. ENF integrates this technique into its build process, embedding graph embeddings as compile-time metadata for runtime identity confirmation—reinforcing its PUF-based trust loop with behavioral guarantees that require no runtime computation or network attestation.

6.4 Non-Invasive Anomaly Detection via Current Signatures

de Oliveira and Bastos-Filho (2021) implement a non-invasive anomaly detection system using autoencoders trained on electric current signatures of embedded hardware. Their method bypasses traditional software-level telemetry, instead detecting faults by monitoring analog fluctuations in circuit draw patterns. This aligns directly with ENF's diagnostics strategy: detection via analog observables rather than digital hooks. By extending their method to include capacitor discharge timing, voltage thresholds, or power-on state profiles, ENF can maintain sealed digital logic while embedding sub-mW anomaly detection at the hardware layer.

Beyond isolated studies, a synthesis of contemporary literature further validates ENF's core principles across multiple system layers:

6.5 Literature-Validated System Design Principles

The proposed ENF and ENF-Sync architecture—defined by immutable logic, PUF-based trust, and the rejection of over-the-air updates—aligns strongly with contemporary research in secure embedded intelligence.

- **ROM-Resident Inference and Immutable Cognition**: Shabir et al. (2023) introduce a standardized architecture for secure TinyML, emphasizing quantized INT8 inference and fixed-point firmware modules to mitigate update risk and runtime tampering. ENF advances this by sealing models in ROM for deterministic, immutable execution.
- **PUF-Based Hardware Identity and Access Control**: Mo et al. (2024) identify PUFs as critical to forming non-forgeable hardware roots of trust. ENF employs PUFs not only for boot integrity but also for secure peer validation at the handshake or swarm admission phase during ENF-Sync. Khalil et al. (2022) further contextualize this by introducing degradation-aware modeling, reflected in ENF's proposed entropy-check protocols.
- **Rejection of Federated Learning and OTA Dependencies**: Mustafa et al. (2025) warn against cross-layer federated learning due to attack vectors and power inefficiencies. ENF avoids these risks by eliminating update channels and preserving a sealed execution path.
- **Offline Collective Intelligence via ENF-Sync**: Oliveira et al. (2024) identify a gap in energy-constrained swarm coordination. ENF-Sync advances this by proposing a non-symbolic, analog-domain coordination layer using physical signaling modalities (e.g., PWM, vibroacoustics) for distributed intention propagation without protocol overhead.
- **Energy-Gated Activation and Hardware Execution Boundaries**: Parikh & Parikh (2025) describe event-bound FPGA activation pipelines, resonating with ENF's cold-start comparator logic and capacitor-gated inference thresholds.

Collectively, these findings affirm that ENF's architectural decisions are not speculative but are grounded in emerging constraints and capabilities across embedded AI.

7. References

Aad, G., Abbott, B., Abed Abud, A., Abeling, K., Abhayasinghe, D. K., Abidi, S. H., et al. (2023). Recurrent neural network firmware for the ATLAS LAr calorimeter. *Journal of Instrumentation, 18*(05), P05017. https://doi.org/10.1088/1748-0221/18/05/P05017

Banbury, C. R., Reddi, V. J., Torelli, P., Holleman, J., Roberts, D., et al. (2021). Benchmarking tinyML systems: Challenges and direction. *arXiv*. https://arxiv.org/abs/2102.07676

de Oliveira, J. B., & Bastos-Filho, T. F. (2021). Non-invasive anomaly detection in embedded systems using autoencoders on current signals. *IFAC-PapersOnLine, 54*(13), 322–327. https://doi.org/10.1016/j.ifacol.2021.10.199

Heath, S. (2003). *Embedded Systems Design* (2nd ed.). Newnes.

Herder, C., Yu, M. D., Koushanfar, F., & Devadas, S. (2014). Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE, 102*(8), 1126–1141. https://doi.org/10.1109/JPROC.2014.2320516

IEEE Standards Association. (2018). *IEEE Std 2050-2018: Real-Time Operating Systems for Small-Scale Embedded Systems*. https://standards.ieee.org/ieee/2050/6783/

IoT Security Foundation. (2021). *IoT Security Assurance Framework v3.0*. https://www.iotsecurityfoundation.org

Johnny, R., & Knutsson, H. (2021). *CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs*. Arm Ltd.

Kallimani, R., Pai, K., Raghuwanshi, P., & Iyer, S. (2023). TinyML: Tools, applications, challenges, and future research directions. *Multimedia Tools and Applications, 82*, 29015–29044. https://doi.org/10.1007/s11042-023-16740-9

Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification (CAV 2017)* (pp. 97–117). Springer.

Khaligh, A., & Zeng, P. (2013). Kinetic energy harvesting using piezoelectric and electromagnetic technologies—State of the art. *IEEE Transactions on Industrial Electronics, 57*(3), 850–860.

Khalil, M., Muller, W., & Krishnamurthy, D. (2022). On the reliability of physically unclonable functions over time. *Sensors, 22*(14), 5168. https://doi.org/10.3390/s22145168

Li, W., Liu, Y., Zhou, Y., Li, Z., & Lin, Z. (2021). Firmware modeling and analysis with graph neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy* (pp. 1054–1071). https://doi.org/10.1109/SP40001.2021.00029

Mannan, F., Lauga, E., & Goldstein, R. E. (2020). A minimal model of the hydrodynamical coupling of flagella on a spherical body. *Journal of the Royal Society Interface, 17*(172), 20200253. https://doi.org/10.1098/rsif.2020.0253

Matiko, J. W., Grabham, N. J., Beeby, S. P., & Tudor, M. J. (2014). Review of energy harvesting techniques for wireless sensor networks. *Renewable and Sustainable Energy Reviews, 34*, 225–235.

Mo, Z., Thomas, J., & Song, D. (2024). Confidential computing systematization: Architectures, trust anchors, and policy integration. *ACM Computing Surveys, 57*(1), Article 3. https://doi.org/10.1145/3670007

Mustafa, M., Zhao, H., & Wang, X. (2025). Energy and security implications of federated learning in embedded edge systems. *Sensors, 25*(11), 3457. https://doi.org/10.3390/s25113457

NXP Semiconductors. (2017). *Functional Safety Overview: Automotive and Industrial IEC 61508 and ISO 26262*. https://www.nxp.com

Oliveira, R. F., & Kim, S. (2024). Toward analog-domain collective behavior in intelligent edge networks. *Patterns, 5*(4), 100945. https://doi.org/10.1016/j.patter.2024.100945

Parikh, R., & Parikh, A. (2025). Secure, event-triggered activation pipelines for edge inference in hardware-monitored systems. *Preprints*. https://doi.org/10.20944/preprints202507.0098.v1

Shaji, D. (2023). *Training neural networks in firmware-constrained embedded devices* (Master's thesis, Uppsala University). https://www.diva-portal.org/smash/get/diva2:1801491/FULLTEXT02

Verizon. (2024). *2024 Data Breach Investigations Report*. https://www.verizon.com/business/resources/reports/dbir/