Poster Abstract: Probabilistic Reasoning on Microcontrollers for Robust Embedded AI Sensing

Jelin Leslin*, Martin Andraud†, and Roberto Morabito‡
*Aalto University, Espoo, Finland †UC Louvain, Louvain-la-Neuve, Belgium ‡EURECOM, Sophia Antipolis, France
Email: jelin.leslin@aalto.fi, martin.andraud@uclouvain.be, roberto.morabito@eurecom.fr

***Abstract*—Embedded AI sensing systems must detect unreliable predictions before acting, yet exact probabilistic reasoning is considered too costly for MCUs. In this work, we introduce μPC, a hardware-aligned inference pipeline enabling exact probabilistic circuit execution via integer arithmetic and sparsity pruning. On an ESP32, μPC achieves up to 44x lower latency than log-domain computation while preserving validation accuracy, demonstrating practical on-device uncertainty checking[1].**

***Index Terms*—Trustworthy Embedded AI, Probabilistic Circuits, Neurosymbolic AI, Edge Computing, Fixed-Point Inference.**

I. Why Embedded AI needs uncertainty guards

Motivation. AI-powered sensing devices increasingly make local decisions under strict latency, energy, and safety constraints (e.g., drones, industrial IoT, health monitors). Hence, a high classification accuracy alone is insufficient: devices must also detect when their prediction is reliable before triggering an action. Yet, embedded AI deployments mostly rely on deep neural networks (DNNs) that are deterministic predictors providing limited mechanisms to assess semantic consistency or uncertainty under shifts. To tackle this, DNNs can be augmented with a lightweight probabilistic reasoning block that validates predictions against a structured world model. For instance, Neural Probabilistic Circuits (NPCs) [1] combine a neural feature extractor with a probabilistic circuit (PC) that reasons over extracted attributes (e.g., shapes or colors) and flags incoherent predictions via posterior inference.

Yet, deploying NPCs on a microcontroller-powered (MCU) device exposes severe hardware limitations. Standard PC implementations rely on log-domain arithmetic and floating-point operations to ensure numerical stability. Popular MCUs (e.g., Cortex-M, Xtensa) lack native 64-bit floating-point units and vector support. As a result, these operations are emulated in software, incurring substantial latency and energy overheads and often violating embedded timing budgets.

Proposed Solution. We introduce μPC, a hardware-aligned inference stack that enables exact probabilistic circuit execution on MCUs. Our key insight is that prediction validation on embedded devices does not require the full dynamic range of log-domain arithmetic typically used in PCs. Instead, we redesign inference around native integer operations and exploit structural sparsity in the circuit. This removes the need for costly floating-point emulation while preserving exact probabilistic reasoning. As a result, low-power devices can execute a complete *"predict-validate-escalate"* pipeline locally, acting not only as sensors but as uncertainty-aware decision entities. We demonstrate μPC on an ESP32, enabling practical NPC inference within embedded timing constraints. Unlike hierarchical inference approaches that rely on confidence thresholds or learned classifiers for offloading decisions [2], µPC performs structured probabilistic validation of semantic consistency directly on-device.

[1]This work was partially funded by the European Union (SUSTAIN project, No. 101071179). The views expressed are those of the authors and do not necessarily reflect those of the EU or EISMEA, which cannot be held responsible.

II. The μPC System Architecture

Fig. 1 illustrates μPC as a multi-stage inference pipeline designed for resource-constrained hardware. The system adapts NPC [1], originally evaluated with large backbones and GPU-class platforms, to operate under microcontroller-level constraints while preserving their uncertainty validation capability.

In contrast to prior implementations that rely on ResNet-34 and floating-point execution, we retain the same probabilistic circuit structure and semantic rules, but redesign both the backbone and the inference arithmetic to fit embedded timing and energy budgets. The goal is to preserve the same uncertainty signals under severe hardware limitations. The system is composed of three tightly coupled stages:

1) **Neural Stage.** A lightweight backbone (ResNet-4 in INT8) processes the input frame and outputs attribute probabilities $\mathbf{A}$ (e.g., color, shape, symbol, text), rather than a direct class prediction.
2) **Sparsity Stage.** Weak attribute signals ($p_i < \tau$) are clamped to zero, enabling subtree masking in the PC and reducing unnecessary computation.
3) **Symbolic Reasoning Stage.** The PC computes the posterior $p(C \mid \tilde{\mathbf{A}})$ on device. If the posterior is confident (e.g., $\max_c p(c|\tilde{\mathbf{A}}) > \theta$), the system outputs SAFE, where θ is calibrated and influenced by the sparsity parameter τ. Otherwise, it raises an `UNCERTAIN` flag.

When uncertainty is detected, the device can optionally escalate the input to a stronger backend (e.g., cloud or edge server), enabling a *hybrid* predict-validate-escalate workflow.

A. PC Arithmetic Variants

We implement and benchmark multiple PC arithmetic variants to select a deployment configuration that satisfies embedded latency–accuracy constraints.

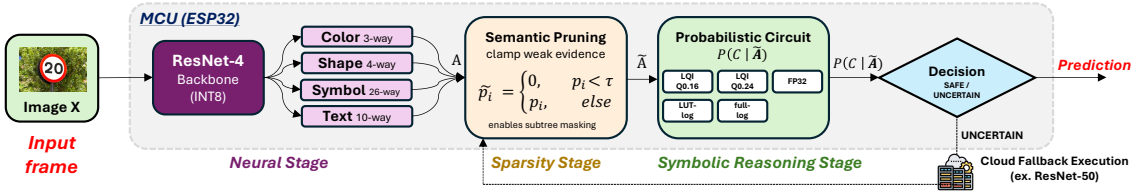

Fig. 1. μPC architecture. A lightweight backbone extracts semantic attributes, which are sparsified and validated via on-device probabilistic reasoning. The system outputs "Safe" or "Uncertain", with optional escalation.

- **Linear fixed-point (LQI).** All probabilities are represented in unsigned fixed-point with F fractional bits ($F \in \{16, 24\}$). Since all values lie in $[0, 1]$, integer bits are not required. Multiplications use 64-bit intermediates to prevent overflow:

$$\texttt{mul}_F(a, b) = \big((uint64\_t)a \cdot b\big) \gg F.$$

  Sum nodes use 64-bit accumulation followed by a right shift by F. The posterior $p(C \mid \tilde{\mathbf{A}})$ is obtained via a fixed-point ratio; in our baseline implementation this includes an integer division, which is the major cost driver.
- **Linear FP32 (FPU).** Linear probability-domain evaluation using 32-bit float HW core.
- **Approximate log-domain (LUT).** Internal PC inference is performed in fixed-point $\log_2$ space:

$$\ell(x) = \lfloor \log_2(x) \cdot 2^s \rfloor, \quad s \overset{\text{empirically}}{=} 12$$

  Products become integer additions. Sum nodes use a LUT approximation of log-sum-exp:

$$\ell(a + b) \approx \ell(a) + 2^s \log_2\big(1 + 2^{(\ell(b)-\ell(a))2^{-s}}\big),$$

  assuming $\ell(a) \geq \ell(b)$. The LUT is indexed by the log-difference and avoids transcendental operations in the hot loop; conversion back to linear space is done once per class.
- **Full log-domain (double).** Reference baseline using double-precision `log`/`exp` function calls.

B. Sparsity-Aware Pruning

To reduce compute, low-probability attribute outputs are clamped to zero, enabling **masked PC evaluation**. The threshold $\tau \approx 0.2$ is calibrated offline, defining a clamped input distribution $\tilde{\mathbf{p}}$. Under $\tilde{\mathbf{p}}$, product terms involving zero-valued leaves vanish, allowing the engine to skip entire subtrees. This yields large data-dependent speedups while preserving exact inference with respect to $\tilde{\mathbf{p}}$, and removes weak evidence without inflating confidence, leading to conservative decisions when signals are inconsistent.

III. Evaluation

We evaluated our pipeline on an ESP32 (240 MHz, 520 KB SRAM) using a custom ResNet-4 attribute extractor.

Performance Analysis. As shown in Table I, standard log-domain inference on the expressive LearnSPN model [3] is computationally intractable ($>$ 17s per inference on the test set). However, by switching to our LQI backend, we reduce this latency to 19 ms, fitting uncertainty modeling within the budget. While the baseline ManualSPN (knowledge-injected) is exceptionally fast (4.2 ms), it effectively collapses to a fixed logical decision graph and lacks the expressive mixture structure of LearnSPN. In contrast, LearnSPN learns mixture-rich structures with latent sum nodes, enabling expressive uncertainty modeling at the cost of higher complexity. Our results show that optimisations are essential to make these learned PCs deployable. Without these optimizations, learned PCs violate embedded timing budgets. Table II shows the impact of integrating µPC into an INT-8 German Traffic Sign Recognition Benchmark (GTSRB), a 43-class real-world traffic sign classification dataset. While ResNet-4 achieves 95% accuracy, adding the PC increases accuracy up to 99.9% with as little as $\approx$ 19 ms additional latency (best-case LQI).

TABLE I
PC INFERENCE LATENCY (MILLISECONDS). RED ENTRIES EXCEED THE PC BUDGET ($\approx 0.75\times$THE NN LATENCY). NZ = NON-ZERO CANDIDATES

Structure	Mode	Format	1 NZ	2 NZ	All NZ	Test_set
LearnSPN	Fixed-point (LQI)	Q0.16	1	2	400	19
	Fixed-point (LQI)	Q0.24	2	19	3271	120
	Linear	FP32	4	54	10550	257
	Approx Log	Log	13	204	40094	840
	Full Log	Log	195	3116	$> 120k$	17553
ManualSPN	Full Log	Log	0.6	1.3	1477	4.2

TABLE II
GTSRB PIPELINE (INT-8), $\tau = 0.2$. BASELINE: RESNET-4. HYBRID: RESNET-4 (4 HEADS) + μPC VALIDATION (SEE TABLE I)

Pipeline	Accuracy	Latency (ms)	Memory
ResNet-4 (43-class)	95%	478	32 KB
ResNet-4 (4 heads) + PC ($\theta = 0.5$)	99.1%	483 + PC	34 KB + PC
ResNet-4 (4 heads) + PC ($\theta = 0.8$)	99.9%	483 + PC	34 KB + PC

IV. Key Takeaway

μPC probabilistic validation feasibility on MCUs, enabling uncertainty-aware embedded sensing under strict timing budgets. Ongoing work explores scalability across tasks and modalities.

References

[1] W. Chen *et al.*, "Neural probabilistic circuits: Enabling compositional and interpretable predictions through logical reasoning," *arXiv preprint arXiv:2501.07021*, 2025.

[2] A. P. Behera *et al.*, "Improved decision module selection for hierarchical inference in resource-constrained edge devices," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–3.

[3] R. Gens and D. Pedro, "Learning the structure of sum-product networks," in *International conference on machine learning*. PMLR, 2013, pp. 873–880.