prometo
Embedded Systems • IT • Safeguard

Lightweight and Secure MCP for Embedded AI

A Practical Guide for Engineers and Decision-Makers

PROMETO GmbH | Embedded World 2026
Jürgen Belz | AI Transformation Lead
info@prometo.de | www.prometo.ai

Table of Content

Section 1

MCP on Embedded Hardware An Engineer‘s Guide

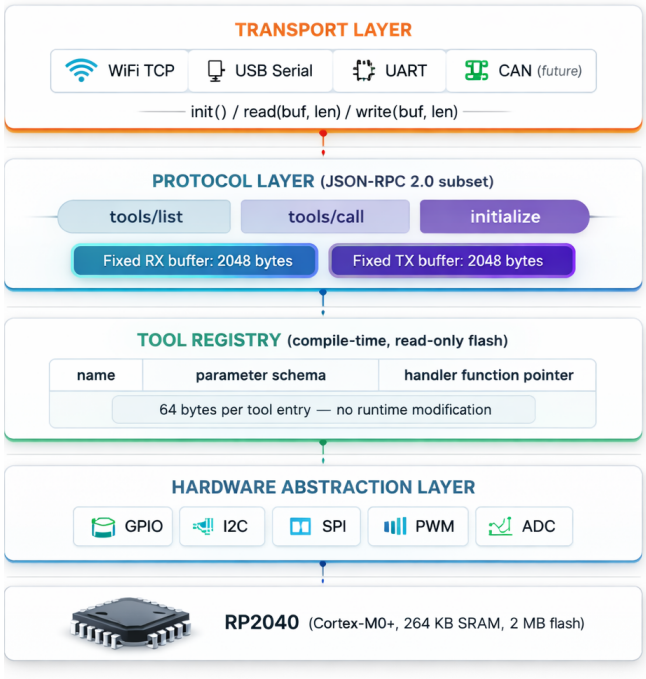
Why Standard MCP Won't Fit - and What Had to Change

The Model Context Protocol runs well on laptops and servers. It assumes a general-purpose OS, a heap allocator, a full HTTP or WebSocket stack, and enough memory to parse arbitrary JSON documents. A Raspberry Pi Pico W has 264 KB of SRAM in total - less than a single HTTP response buffer in a typical cloud MCP server.

Three things had to be eliminated:

Standard MCP assumption	Embedded reality	Design response
Dynamic heap allocation	No MMU, fragmentation is fatal	Static buffers only, allocated at compile time
Full JSON-RPC + HTTP/WS transport	Both too heavy for available RAM	Newline-delimited JSON over bare TCP (or UART)
Full MCP feature set (tools + resources + prompts + sampling)	Code size budget exhausted by WiFi stack alone	Tools only: `tools/list` and `tools/call`

The result is an MCP server that is protocol-compatible - any MCP host connects without modification - but built on entirely different internals.



Source: AI-generated illustration (created using ChatGPT)

Why the layering matters in practice:

The transport boundary means swapping from WiFi to UART requires changing exactly one module. The tool registry boundary means adding a new tool, a new sensor, a relay, a button - is three lines of code: a handler function, a schema string, and a table entry. Nothing else changes.

The Numbers That Matter

Two implementations, one architecture

This guide and the accompanying GitHub demo use two different implementations of the same architecture - intentionally.

	Demo (this repo)	Production target
Language	MicroPython	C (bare-metal or RTOS)
MCP tools exposed	3	6+
Purpose	Run it in an afternoon, understand the protocol	Deployment on constrained hardware
Footprint	MicroPython runtime dominates (~1.4 MB firmware)	See targets below

The MicroPython demo exists to lower the barrier to entry. You can connect a Pico W, run three commands, and have a working MCP server talking to a local LLM within an hour. No toolchain setup, no cross-compilation, no linker scripts. That is the point.

The production targets below describe what a C implementation of the same architecture achieves on the same RP2040 hardware. These are the numbers that matter for a real embedded deployment.

Production target: memory footprint (C, RP2040, WiFi TCP, 6 tools)

Component	Flash	SRAM
MCP protocol layer	8,200 B	4,608 B
JSON parser	3,100 B	128 B
Tool registry (6 tools)	2,400 B	384 B
Transport layer (WiFi TCP)	12,800 B	6,144 B
Hardware abstraction layer	5,600 B	512 B
WiFi stack (CYW43)	41,000 B	38,400 B
Application logic + runtime	13,900 B	3,072 B
Total	**87,000 B**	**53,248 B**

The MCP-specific components (protocol + parser + registry) cost **13.7 KB flash and 5.1 KB SRAM**. The WiFi stack dominates. On UART or CAN transport, the WiFi stack is absent and total footprint drops below 50 KB.

After boot, **~211 KB of SRAM remains free** for application data, sensor buffers, and stack - on a device with only 264 KB total.

Production target: tool call latency (WiFi TCP, local network)

Tool	Mean	Max
Sensor read (I2C, 2 values)	12 ms	23 ms
Actuator write (NeoPixel / PWM)	8 ms	15 ms
Display write (SPI)	14 ms	28 ms
GPIO read (button state)	7 ms	14 ms

All tool calls complete in under 30 ms over WiFi. USB serial latencies run approximately 40% lower. The MicroPython demo is measurably slower due to interpreter overhead - but sufficient to validate the protocol behaviour and tool interaction patterns.

Power (both implementations, hardware is identical)

- Active WiFi + tool call processing: ~120 mA at 3.3V
- WiFi connected, idle: ~80 mA
- Acceptable for mains-powered or battery systems with moderate duty cycles

Three Design Rules - and What Breaks if You Ignore Them

Rule 1: No dynamic allocation after boot

All buffers . receive, transmit, tool table, sensor cache - are statically sized at compile time. No `malloc`, no `new`, no dynamic list growth.

Break it and: On a microcontroller without an MMU, heap fragmentation accumulates silently. A device that runs for 20 minutes in the lab may hang after 6 hours in the field. The failure mode is non-deterministic and extremely hard to reproduce.

Rule 2: Support only `tools/list` and `tools/call`

The full MCP specification defines four capability categories: tools, resources, prompts, and sampling. This implementation supports tools only. Resource serving (file/data streams) and prompt templates are omitted.

Break it and: Adding resource serving means handling streaming responses and potentially unbounded data sizes — neither is compatible with fixed buffers. Adding sampling means the embedded device becomes an LLM client, which inverts the architecture entirely.

Rule 3: Abstract the transport behind three operations

The transport interface is `init()`, `read(buffer, length)`, `write(buffer, length)`. The protocol layer calls only these three functions and has no knowledge of whether it is running over TCP, UART, or USB serial.

Break it and: Transport-specific code bleeds into the protocol layer. Porting to CAN or RS-485 for a production deployment requires surgical changes throughout the codebase rather than a single module swap.

Security Checklist for Edge-MCP Deployments

Edge devices are physically accessible. The threat model differs from cloud: an attacker may have physical access to the hardware, debug interfaces may be exposed, and the MCU cannot run expensive cryptographic operations inline with low-latency tool calls.

Threats specific to embedded MCP

- **Input manipulation**: Malformed JSON-RPC crafted to overflow the receive buffer or trigger undefined behaviour in tool handlers
- **Tool poisoning**: Runtime modification of the tool registry to register unauthorized tools or override handler pointers
- **Credential leakage**: WiFi credentials and any auth tokens stored in unprotected flash, readable via debug interface
- **Side-channel attacks**: Timing or power-analysis attacks on devices with physical access

Mitigations (implement all six for regulated deployments)

- ☐ **Fixed-size receive buffer with hard rejection**: Messages exceeding 2048 bytes are dropped at the transport layer, before any JSON parsing begins. No partial processing.
- ☐ **Static tool registry in read-only flash**: Tool table compiled into flash at firmware build time. No runtime registration, no function pointer overwrite possible.
- ☐ **Parameter validation before dispatch**: All tool parameters are type-checked and range-checked against the schema before the handler is called. String lengths are capped. Numeric values are bounds-checked.
- ☐ **Allowlisting + capability-based access control**: Each tool is annotated with a required capability level. Multi-client or multi-role deployments restrict tool access per authenticated caller.
- ☐ **Rate limiting**: Configurable maximum tool calls per time window. Excess requests are rejected with a protocol error. Prevents both DoS and brute-force scenarios.
- ☐ **Hardware watchdog**: If the MCP server loop becomes unresponsive (malformed message causing a hang), the watchdog resets the MCU to a known safe state within a configurable timeout.

For regulated industries (IEC 62443, ISO 21434):

- Integrate an external secure element (e.g. ATECC608A) for WiFi credential and key storage
- Write all tool invocations and results to a circular audit log in flash
- Map the above mitigations to your applicable security level or TARA findings

What Is Not Solved Yet

Honest assessment as of Embedded World 2026:

JSON overhead on low-rate links. A `read_temperature` request and response total ~280 bytes in JSON-RPC. On a LoRa link or slow UART, this is significant. Replacing JSON-RPC with MessagePack or CBOR would reduce message size by 50–70% while preserving MCP semantics. This trade-off - standard compatibility vs. efficiency - is an open engineering problem.

Local LLM reasoning quality. A 7B–14B quantized model running on a laptop handles simple sensor-actuator loops well. Complex multi-step reasoning or nuanced fault diagnosis may require larger models, which reintroduces cloud connectivity. The quality floor for local-only deployments is a function of model capability, not MCP.

No RTOS integration. The current implementation runs cooperatively. There are no hard real-time guarantees. For safety-critical applications requiring bounded response times, integration with FreeRTOS or Zephyr and priority-based scheduling is necessary - not yet done.

Single-client only. One MCP host per device. Multi-agent scenarios (two LLMs coordinating via the same device) are not supported by the current architecture.

Try It Yourself

Everything needed to reproduce the demo is on GitHub.

Hardware (total cost ~€45):

- Raspberry Pi Pico W
- Pimoroni Pico Omnibus + Pico Display Pack PIM543
- Seeed Studio TH02 Grove temperature/humidity sensor
- AZ-Delivery CJMCU-2812 8-LED WS2812B ring
- 1N4148 diode

Three commands to get running:

```bash
# 1. Flash Pimoroni MicroPython to the Pico W (see README)
# 2. Install Ollama and pull a model
brew install ollama && ollama pull llama3.2 && ollama serve

# 3. Connect and start the interactive agent loop
python3 mcp_client.py <pico_ip>
```

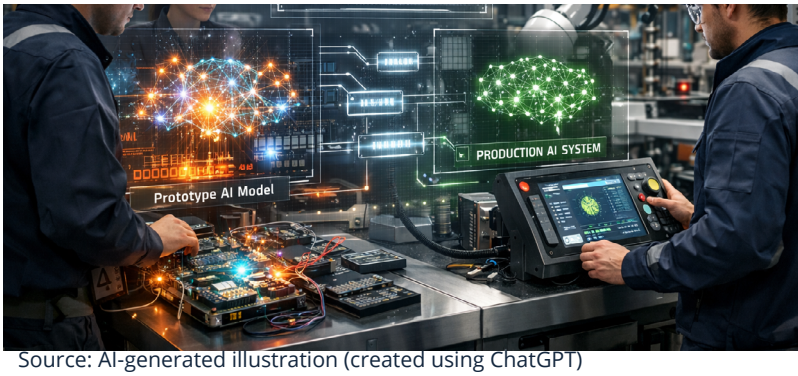
Full wiring, pin assignments, and step-by-step setup:
https://github.com/PROMETO-GmbH/EW26-Embedded-MCP/tree/main

Section 2

From Pilot to Production - How PROMETO Helps

Who This Is For

You are building AI into embedded systems, industrial development, or regulated production environments. You have enough technical depth to evaluate what you read in Section 1. The question is not whether AI is relevant - it is how to move it from a proof-of-concept into something that actually runs in your organization without creating new compliance liabilities or engineering debt.



Source: AI-generated illustration (created using ChatGPT)

About Jürgen Belz

AI Transformation Lead, PROMETO GmbH


Source: PROMETO GmbH

I work as an operational partner for firms in regulated industries - energy, automotive, aerospace, smart home, industrial automation - helping them embed AI into governance structures, development workflows, production processes, and compliance frameworks.

My focus is the gap between „we've done a pilot" and „this is now part of how we work." That gap is not primarily a technical problem. It is an organizational and architectural one: AI initiatives that stay in pilot mode are usually missing either a structural home in the engineering process or a clear path through the compliance requirements of the sector.

The MCP work at Embedded World is one concrete example of what that looks like at the hardware level. The broader practice applies the same logic across the full engineering and operations stack.

Why This Matters Now

AI is not an add-on feature for development and production organizations operating in competitive global markets. It is the structural spine of how fast you can move, how much institutional knowledge you can preserve, and how robustly you can make decisions under pressure.

The organizations that will hold their position - and the ones that will lose it - are being separated right now by one variable: whether AI is integrated as a first-class citizen into tooling, testing pipelines, and decision loops, or whether it remains a collection of ad-hoc experiments that never compound.

The shift from ad-hoc pilots to resilient AI infrastructure delivers three things that are directly measurable.

Faster time-to-market
AI inside the development process, not bolted onto the side of it.

Higher quality
AI-assisted review, test generation, and anomaly detection at the points where errors are cheapest to catch.

Stronger regulatory compliance
AI governance built into the process, not retrofitted during audits.

What I Bring to the Table

- Hands-on implementation experience across regulated sectors - not advisory from a distance
- A framework for aligning AI initiatives with corporate governance and compliance requirements (IEC 62443, ISO 26262, ISO 21434, and others)
- Tools and test-beds that your engineers can use immediately to accelerate experimentation - including the open-source embedded MCP stack from this talk
- Pattern libraries from deployments that have already navigated the organizational and regulatory friction you are about to encounter

Where Are You Right Now?

Three situations account for most of the organizations that reach out after conversations like this one. Read the one that fits, skip the others.

„We've heard of MCP - or AI at the edge, but we don't know where it fits in our architecture."

You have engineers who are interested. You may have run a small experiment. But you do not yet have a clear picture of where MCP or edge AI belongs in your system architecture, how it interacts with your existing communication stack, or whether the security and compliance implications are manageable.

AI Transformation Lead, PROMETO GmbH

A structured discovery session - typically half a day - to map your current architecture, identify the integration points where AI-agent protocols would add the most value, and surface the compliance and security questions you will need to answer before going further.

The output is a concrete roadmap: where to start, what the first working prototype should demonstrate, and which organizational or compliance questions need to be resolved in parallel.

What you walk away with:

- A written integration map showing where MCP or edge AI fits in your specific architecture
- A prioritized list of the two or three integration points with the highest near-term value
- A scoped proposal for the next phase if you want to continue

This is the right starting point if your question is still „should we, and where?"

„We‘re prototyping, but we can‘t get it to production-ready quality.“

You have a working demo - or close to one. The fundamentals are there: you can read a sensor, invoke a tool, get a response from an LLM. But the prototype has rough edges: memory behavior under load is untested, the security model is informal, the failure modes are unclear, and nobody has thought through what happens when the WiFi drops mid-tool-call.

You know it needs hardening before it goes anywhere near a production environment. You are not sure exactly what „hardened“ means in your context, or how long it will take.

What engagement looks like:

A focused sprint - typically two to four weeks - alongside your team. We work through the architecture systematically: memory budget review, failure mode analysis, security threat modelling against your specific deployment context, and compliance mapping to whichever standard governs your sector.

The output is not a report. It is a hardened version of your implementation with documented design decisions, a security checklist that maps to your applicable standard, and a test protocol that validates the critical edge cases.

What you walk away with:

- A production-ready architecture with documented design decisions
- A security and compliance package your team can take into your internal review process
- Engineers who understand why each design decision was made and can maintain and extend it

This is the right starting point if your question is „we have something working - how do we make it production-grade?“

„We need compliant AI integrated into our development or production workflows.“

This is the broadest version of the challenge. You are not primarily asking about embedded MCP. You are asking how to make AI a structural part of how your engineering organization works, across development, testing, documentation, and compliance, in a sector where „we tried a new tool and it failed an audit“ is a real and career-limiting outcome.

The embedded MCP work is one piece. The broader challenge is architectural and organizational: building the AI infrastructure that makes your engineers faster and your compliance posture stronger at the same time, rather than trading one off against the other.

What engagement looks like:

An ongoing operational partnership, typically structured in three-month cycles. We work inside your organization across the full stack: tooling selection and integration, AI governance framework design, compliance alignment, engineer enablement, and the process changes needed to make AI stick as a working practice rather than a pilot that fades out.

The scope is defined by your highest-priority challenges and reviewed at each cycle.

What you walk away with:

- AI integrated into the engineering process at the points where it delivers the most leverage
- A governance framework that meets your compliance requirements and does not block your engineers
- A team that can continue building on the infrastructure without depending on external support

This is the right starting point if your question is „we need AI to become part of how we work - and we operate in a regulated environment.“

Next Steps

If one of the three situations above fits where you are, the right next step is a 30-minute conversation — no preparation required on your side, no commitment implied.

I will ask you three questions about your architecture, your compliance context, and where the biggest friction is today. You will leave with a clear picture of which engagement model makes sense and what the first concrete step looks like.

Email directly: info@prometo.de
Reference code for this guide: EW26-MCP

For questions about the technical content, open an issue on GitHub.
For questions about the engagement models, write to info@prometo.de