Building AI Agents for the Internet of Things

Jaime Jiménez
jaime.jimenez@ericsson.com, Ericsson Research
April 2026

Agenda

1. **Demo**
2. **The Problem** - why current IoT integrations don't scale
3. **CoAP & Devices** - the protocol and self-describing devices
4. **The Agent** - architecture, tools, and the ReAct pattern
5. **Takeaways** - resilience, grounding, tokens, local models
6. **The Landscape** - Matter, agentic AI standards, open problems, agent-readiness

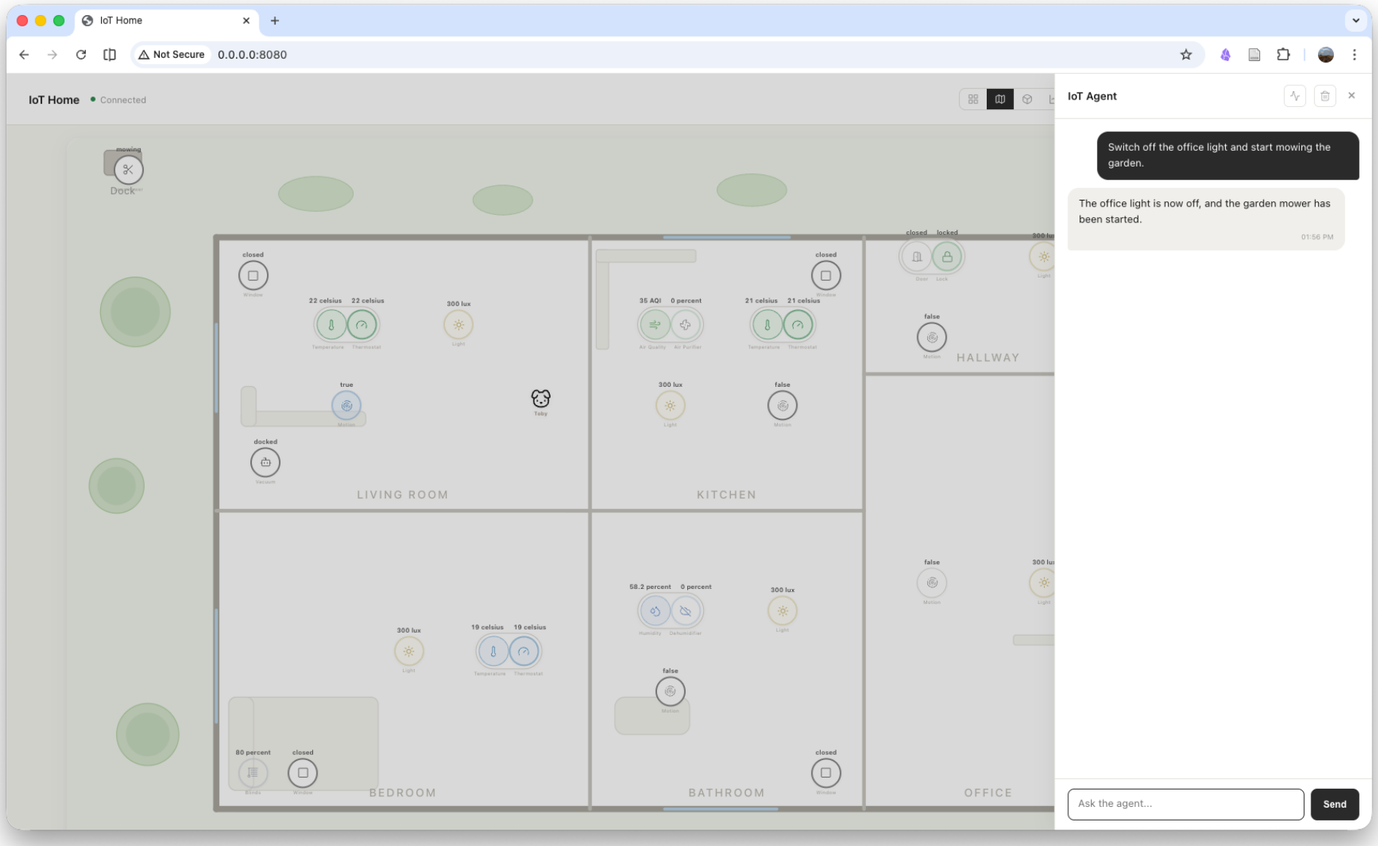
Live Demo

- User expresses **intent** ("it is hot")
- **Planner/Executor** loop discovers IoT resources via CoAP
- Reads `/temperature`, adjusts `/thermostat`
- No hardcoded device list - just a well-defined entry point
- Bridges **Knowledge World** (LLM) and **Physical World** (devices)

T2TRG slides | Video



Live Demo II


IoT Home
Not Secure
0.0.0.0:8080
IoT Home
Connected
IoT Agent
Switch off the office light and start mowing the garden.
The office light is now off, and the garden mower has been started.
01:56 PM
Ask the agent...
Send
LIVING ROOM
KITCHEN
HALLWAY
BEDROOM
BATHROOM
OFFICE
Dock

I. The Problem

The Universal Hypermedia Client

There is no magical "smart client" that somehow knows that `rel=comments` *means that the link leads to comments about the current resource and can figure out it should* `POST` *there to create a new comment. It would effectively be a single client that could make use of every single (properly built) REST API in existence without requiring documentation. What would that even mean?*

The HATEOAS[1] promise: a client navigates any API by following links, no hardcoded endpoints. The blocker was always semantic understanding.

Then LLMs arrived. They *are* the universal hypermedia client.

---

[1] Hypermedia as the Engine of Application State

Static vs Agentic Integration

Traditional	
Commission device	Manual, per-device
Client support	Pre-implemented
New capability	Firmware update
New device type	Code change
New deployment	Re-integration

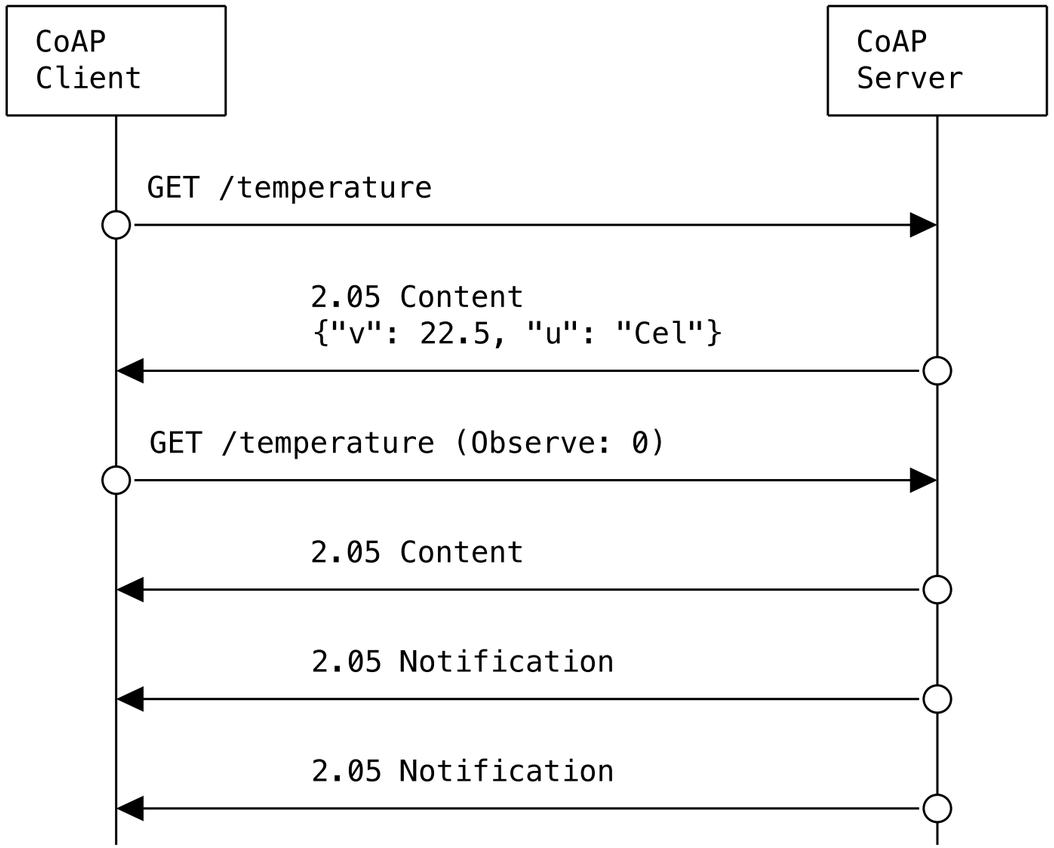
Agentic	
Commission device	Standard onboarding
Client support	**Discovered at runtime**
New capability	**Already accessible**
New device type	**Self-described**
New deployment	**Same client**

II. CoAP & Devices

CoAP Overview

Constrained Application Protocol (RFC 7252)

- Like HTTP, but for tiny devices
- UDP instead of TCP (+ DTLS for security)
- 4-byte header vs HTTP's ~700 bytes
- Same verbs: GET, PUT, POST, DELETE
- Built-in resource discovery
- Built-in Observe pattern.



Resource Discovery

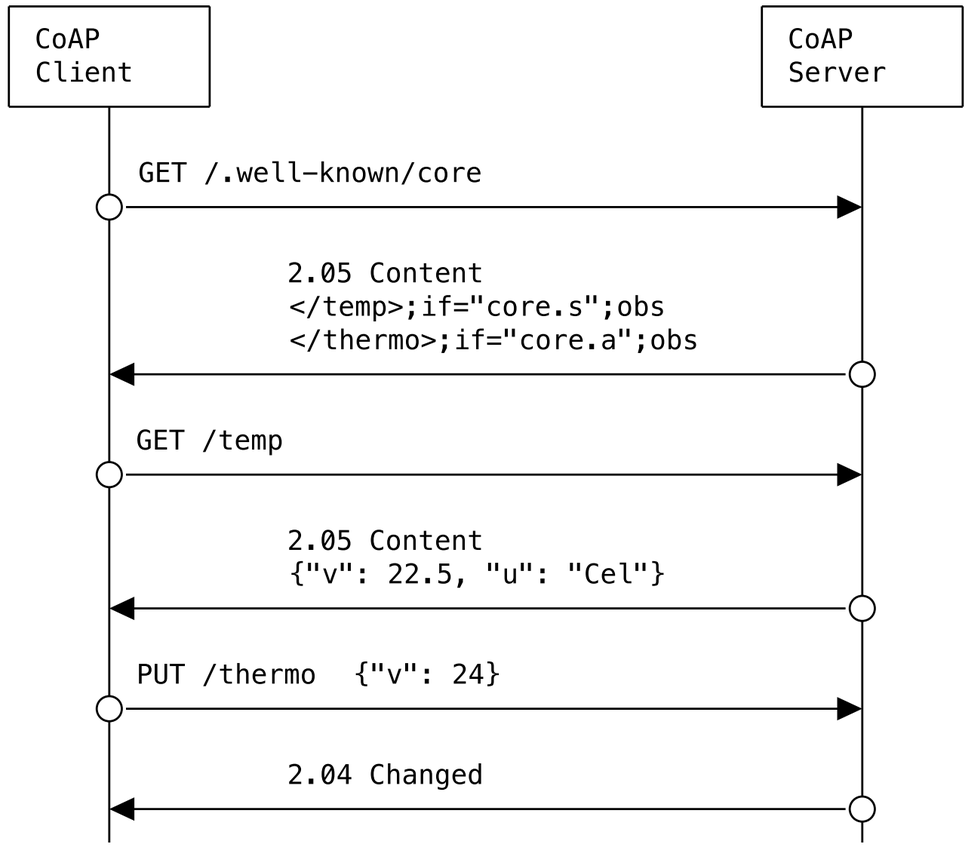
Every CoAP device advertises its resources at a well-known URI.

1. **Discover**: `GET /.well-known/core`
2. **Read**: `GET /temp` ➳ sensor value
3. **Write**: `PUT /thermo` ➳ actuate

`if="core.s"` = sensor, `if="core.a"` = actuator

`obs` = push notifications

Devices join and leave at runtime. The agent re-discovers automatically.



Device Semantics: SDF & W3C TD

Two standards, both agent-ready. We use SDF here; W3C TD works equally well.

	SDF (RFC 9880)	W3C TD 1.1
Origin	IETF ASDF WG	W3C WoT WG
Models	Properties, Actions, Events	Properties, Actions, Events
Protocol	Agnostic (pairs with CoRE links)	Built-in (`forms`)
Hypermedia	No	Yes

Both give the agent what it needs: types, ranges, actions, and semantics. Convertible via OneDM sdf-wot-converter.

SDF Switch:

```
json
{"sdfObject": {"Switch": {
  "sdfProperty": {"value": {"type": "boolean"}},
  "sdfAction": {"toggle": {}}
}}}
```

W3C TD Switch:

```
json
{"@context": "https://www.w3.org/2022/wot/td/
v1.1",
 "title": "Switch",
 "properties": {"value": {
   "type": "boolean",
   "forms": [{"href": "/switch"}]}},
 "actions": {"toggle": {
   "forms": [{"href": "/switch/toggle"}]}}}
```

III. The Agent

Architecture

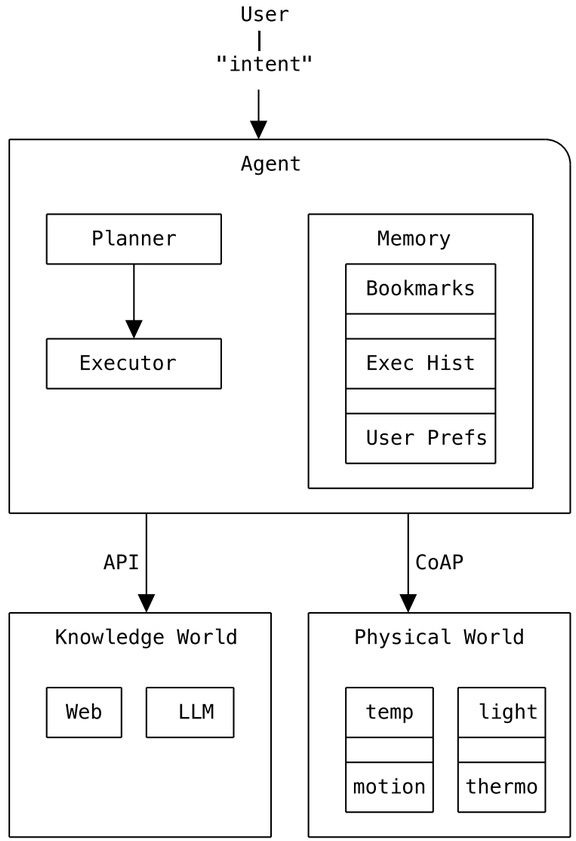
User: interacts via natural language "intents"

Agent: LLM-powered system that breaks intents into actionable steps

- **Planner**: decomposes intent into a plan
- **Executor**: runs code against tools
- **Memory**: bookmarks, execution history, user preferences

Knowledge World: web search, LLM reasoning

Physical World: IoT devices via CoAP



Agent-Device Interaction

The MCP/LangChain way

20 devices = 20 tools

- `get_kitchen_temperature()`
- `set_bedroom_thermostat()`
- `toggle_hallway_light()`
- … one per device, per room

Add a device ➳ write a new tool.

The CoAP way

20 devices = **4 tools**

- `list_resources()`
- `get_resource(name)`
- `set_resource(name, value)`
- `observe_resource(name)`

Add a device ➳ it just works.

MCP vs CoAP as Agent Protocols

Capability	MCP (2025-06-18)	CoAP/IoT
Actions	`tools/call` with JSON Schema	GET, PUT, POST, DELETE on typed resources
Context data	`resources/read`, resource templates	`/.well-known/core` + CoRE Link Format
Discovery	`*/list` methods, Server Cards	Resource Directory (RFC 9176)
Real-time updates	`notifications/*`, subscriptions	CoAP Observe (RFC 7641)
Semantic descriptions	Tool/resource descriptions (free text)	SDF (RFC 9880) / W3C TD, structured
LLM integration	Sampling, Elicitation	No equivalent (agent-side concern)
Interaction templates	Prompts (few-shot examples, system prompts)	No equivalent

ReAct Pattern

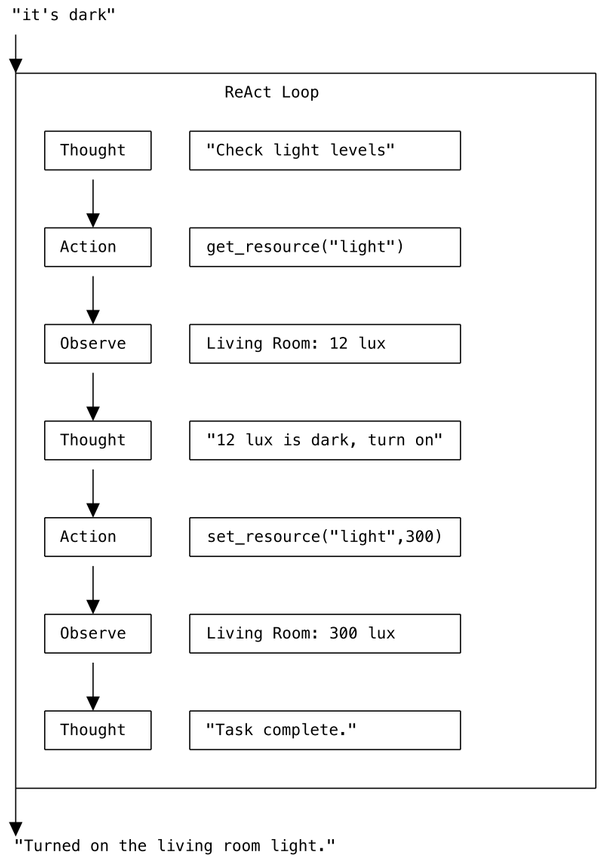
Reasoning + **Act**ing (Yao et al., ICLR 2023)

Three steps in a loop:

1. **Thought**: reason about state, plan next action
2. **Action**: execute a tool call
3. **Observation**: read the return value

Loop repeats until the agent can answer.

Code generation, not function calling: the agent writes Python with loops, conditionals, and variables.



Tool Interface

The **docstrings are the LLM's instruction manual**. The agent reads them to decide which tool to call and how.

```
@tool
def get_resource(resource_name: str) -> dict:
    """Gets current value of a resource by name.
    Returns all matches with room labels."""

@tool
def set_resource(resource_name: str, value: str) -> str:
    """Sets a value. Only works on writable resources."""

@tool
def observe_resource(resource_name: str,
                     ideal_value: float = None) -> str:
    """Confirms a change reached the target.
    Automatically finds the related sensor."""
```

Frameworks

Agent Frameworks

Framework	Approach
smolagents	Code generation
LangGraph	Graph workflows
CrewAI	Multi-agent roles
AutoGen	Conversation agents
Google ADK	Google ecosystem

We chose smolagents: ~1,000 lines of core code, deliberately minimal.

IoT Libraries

Library	Language
aiocoap	Python, async
libcoap	C, embedded
CoAPthon3	Python, sync
californium	Java
go-coap	Go

We chose aiocoap: native async/await, Observe support, same event loop as the agent.

IV. Takeaways

Agent Resilience

Three layers of error recovery, none pre-programmed:

Layer	What goes wrong	What the agent does
Protocol	Device returns 4.08 "value out of range"	Parses problem-detail (RFC 9290), adjusts, retries
Code	Generated Python doesn't compile	Reads traceback, rewrites its own code
Intent	Writes 22°C, reads back 19°C	Detects mismatch, tries alternative approach

Zero lines of error-handling code. The agent resolves errors it was never programmed to handle.

Self-Improving Infrastructure

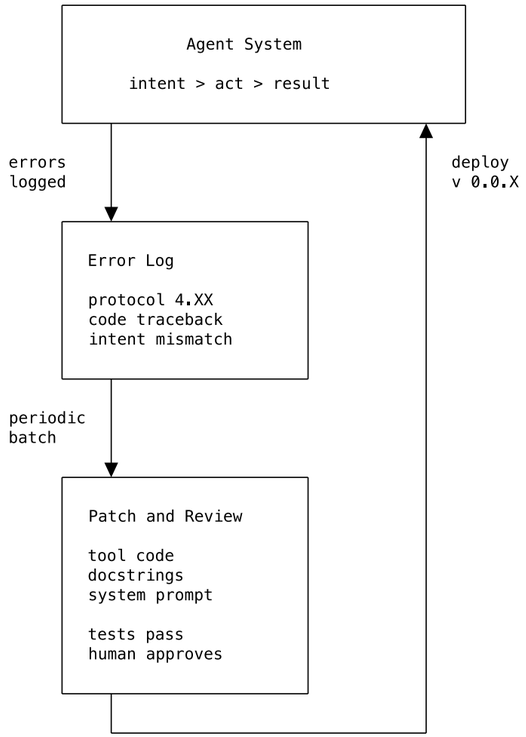
The agent handles errors at runtime (previous slide).

But some errors **recur across sessions**. The root cause isn't the LLM but the code: ambiguous docstring or prompts, a missing field in a tool's return value, an implicit convention, a missing function...

The **outer loop** fixes the environment, not the agent:

- Accumulate structured errors during operation
- Batch-analyze patterns, generate patches
- Human reviews, test suite gates, version bumps

Each release shrinks the error surface.

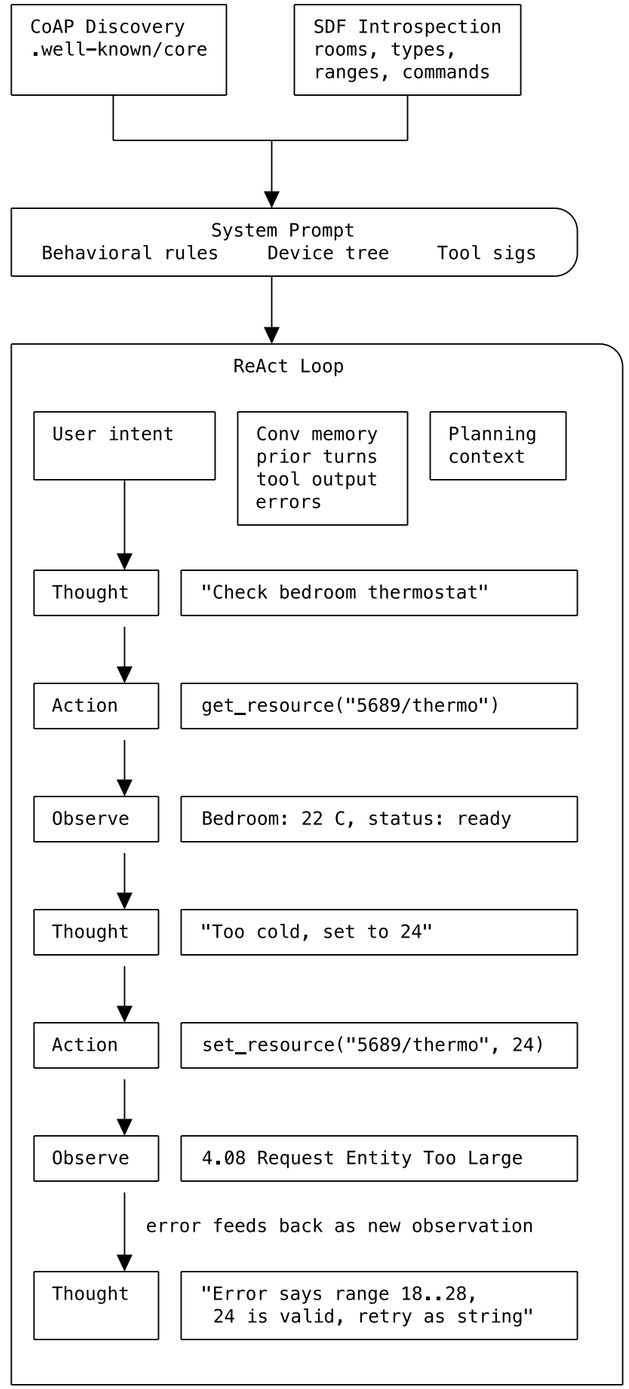


Grounding

At every step of the ReAct loop, the LLM has:

- **Static context**: device tree from CoAP discovery + SDF/W3C TD introspection (rooms, types, ranges, valid commands)
- **Dynamic context**: conversation memory, prior tool outputs, error tracebacks, planning state
- **Error feedback**: CoAP problem-detail (RFC 9290) feeds back as observations for self-correction

Reduced invalid tool calls from ~1 in 3 to fewer than 1 in 20.



Message and Token Optimization

Waste	Fix
Calling `list_resources()` every turn	Device tree baked into system prompt; planning prompt says "do NOT call list_resources"
Reading all n devices	CoAP Multicast
Replanning every step	Planning interval: every 3 steps (max 2 planning rounds per task)
Re-discovering between turns	`reset=False` preserves memory; background health monitor auto-rediscovers on device change
Repeating similar requests	State-aware cached execution (next slide)

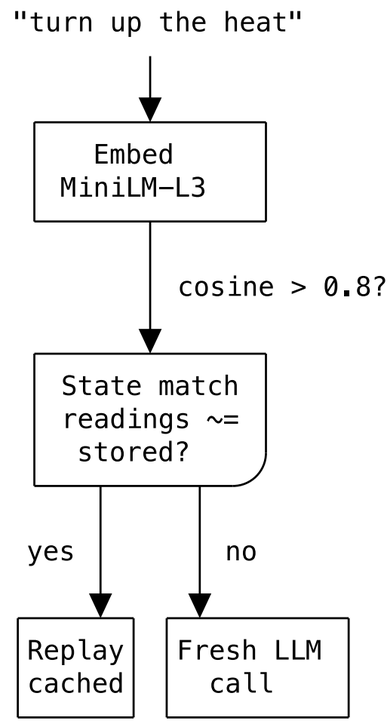
State-Aware Cached Execution

Store `{intent, code, system_state}` after every successful run.

On a new request:
1. Embed the intent with MiniLM (semantic similarity)
2. Rank matches above 0.8 threshold
3. Gate on system state: cached code only replays if current device readings match the original context

Different starting state ➵ cache miss ➵ fresh LLM call.

On match: show predicted result, ask for approval, execute directly. Zero LLM tokens.



Local Models

- Not every deployment can call GPT-4o
- Privacy, cost, latency
- Overall knowledge about systems and appliances is universal
- IETF specs (CoAP, CoRE Link Format) are in the training data of every major LLM
  - Bottleneck is quality code generation, not protocol knowledge

Tested against 3 real tasks: list devices, read a sensor, compare across rooms.

Model	Size	List	Temp	Compare
GPT-4o (Azure)	cloud	+	+	+
glm-4.7-flash	29.9B	+	+	+
qwen3	8B	+	+	+
devstral	23.6B	±	+	–
mistral-small	24B	+	+	–
phi4	14.7B	–	–	–

Best local: glm-4.7-flash (self-corrects from errors mid-loop)
Best tradeoff: qwen3 8B (passes all three, fits on a laptop)

V. The Landscape

Home IoT Today: Matter

The smart home ecosystem has consolidated around **Matter**.

- Apple, Google, Amazon, Samsung ...
- IPv6 networking (Wi-Fi, Thread, Ethernet)
- Interaction Model: Read, Write, Subscribe, Invoke
- Data model: fixed clusters with typed attributes
- Discovery: DNS-SD/mDNS
- Open source (connectedhomeip)

The gap for agents

Matter's Descriptor Cluster lists device type IDs, cluster IDs, and semantic tags per endpoint, and wildcard reads can enumerate attribute values. But it does not expose attribute schemas (types, ranges, units) or command parameter definitions at runtime. You need compiled-in knowledge of what cluster 0x0006 means and what attributes it has.

If Matter exposed thing self-descriptions (SDF or W3C TDs), the same agent pattern would apply.

What Matter Reinvented

Capability	CoAP (IETF)	Matter (CSA)
Reliability	CON/NON per message	MRP retransmit + backoff
Security	DTLS (reusable, interoperable)	PASE/CASE + AES-CCM (Matter-only)
Subscribe	Observe (RFC 7641), lightweight	Subscribe Interaction, heavier but coherent multi-attribute snapshots
Discovery	`.well-known/core` + Link Format	DNS-SD + Descriptor Cluster (IDs and tags, not schemas)
Encoding	CBOR/JSON, content negotiation	TLV only, no negotiation
Caching	Max-Age, ETag, If-Match	None
Large payloads	Block1/Block2 (RFC 7959)	BDX (separate protocol)

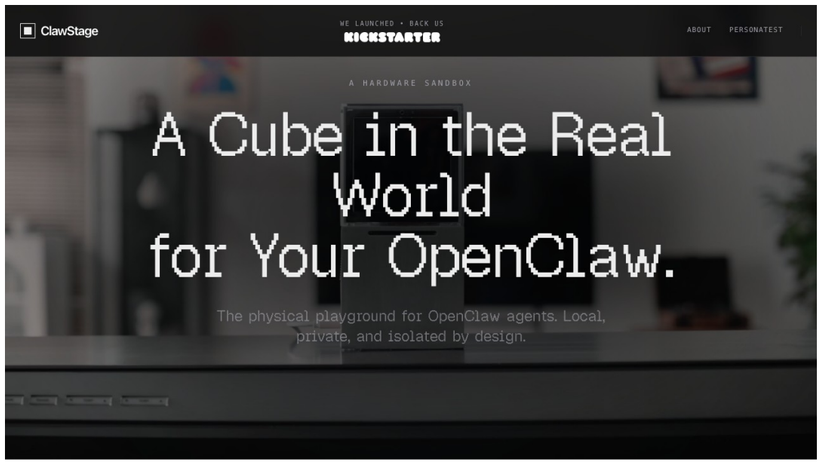
CoAP extensions are independent RFCs that compose with LwM2M, OCF, OSCORE. Matter's equivalents only work within Matter.

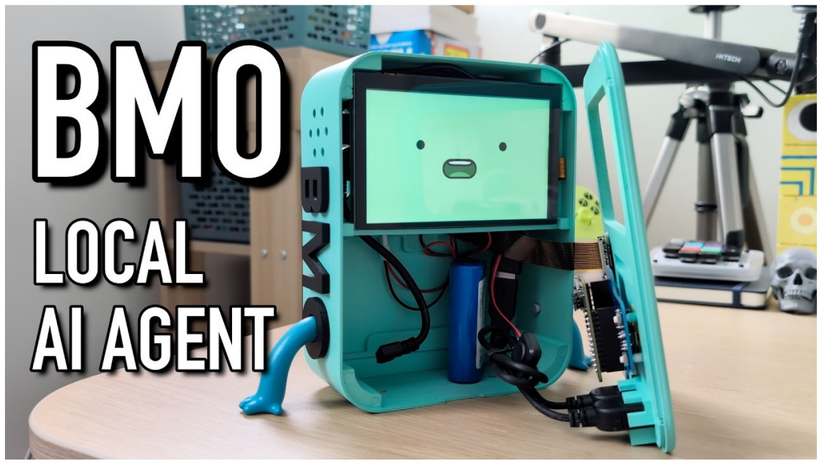
Agentic AI Today

Two waves, same pattern as home IoT:

Wave 1 - Innovation (now)
Open source and big tech ship fast. MCP, A2A, LangChain, smolagents, OpenClaw.
Competing protocols, competing frameworks, no interop.

Wave 2 - Consolidation (next)
Best practices harden into standards. Identity, authorization, discovery, trust. The IETF has the building blocks; the question is whether it moves fast enough.





Identity and Authorization

The most active area. An agent needs to prove **who it is**, **who sent it**, and **what it is allowed to do**.

WIMSE (Workload Identity in Multi-System Environments):
- `wimse:` URI scheme for agent identity
- Workload credentials (JWT + X.509)
- HTTP Signatures for workload-to-workload auth
- Closest to shipping: WGLC target IETF 126

OAuth extensions for agents:
- **Identity chaining**: access tokens across trust boundaries via Token Exchange
- **Transaction tokens**: propagate identity through multi-agent call chains
- **Rich Authorization Requests**: per-tool authorization scoping
- **Agent auth framework** (draft-klrc-aiagent-auth): composes WIMSE + OAuth + SPIFFE, identifies gaps

Discovery and Trust

How do agents find each other?

- DNS-AID: _agent subdomains + `.well-known` fallback
- API Catalog (RFC 9727): machine-readable API discovery
- Content preferences (draft-ietf-aipref-vocab): sites express AI crawling rules
- The `.well-known` namespace is getting squatted: `agents.json`, `ai.txt`, no real specs behind them

How do you verify an agent is what it claims?

- RATS CoRIM: reference integrity manifests for agent environments
- Entity Attestation Results: standardized verifier output
- Verifiable agent conversations: CDDL log format signed with COSE_Sign1
- SD-CWT: privacy-preserving credentials for constrained agents

Web APIs can learn from IoT

The same pattern applies to IoT (CoAP) and web APIs (HTTP). Each level is independent:

Level	IoT (CoAP)	Web API (HTTP)	What agents gain
Permissions	ACE scope advertisements	`robots.txt` + Content Signals	Know what's allowed
Discovery	`/.well-known/core` (RFC 6690)	`/.well-known/api-catalog` (RFC 9727)	Find available resources
Auth	ACE-OAuth (RFC 9200)	OIDC + OAuth discovery (RFC 9728)	Authenticate without docs
Description	SDF / W3C TD	OpenAPI + SKILL.md	Understand capabilities
Navigation	CoRE Link Format + Observe	`_links` in responses (HATEOAS)	Follow links, not hardcode paths
Error recovery	CoAP Problem Details	RFC 9457 Problem Details	Self-correct on failure

References

Name	URL
RFC 7252 - CoAP	rfc-editor.org/rfc/rfc7252
RFC 6690 - CoRE Link Format	rfc-editor.org/rfc/rfc6690
RFC 7641 - CoAP Observe	rfc-editor.org/rfc/rfc7641
RFC 9176 - Resource Directory	rfc-editor.org/rfc/rfc9176
RFC 9880 - SDF	rfc-editor.org/rfc/rfc9880
W3C Thing Description 1.1	w3.org/TR/wot-thing-description11
draft-jimenez-t2trg-iot-agent	datatracker.ietf.org/doc/draft-jimenez-t2trg-iot-agent
Agentic AI drafts (curated list)	ietf.jaime.win/agentic-ai-drafts
smolagents	github.com/huggingface/smolagents
aiocoap	github.com/chrysn/aiocoap
Matter	github.com/project-chip/connectedhomeip
Ollama	ollama.com
IETF 123 presentation	datatracker.ietf.org/meeting/123/materials/slides-123-t2trg-agentic-ai-operation-of-iot-systems-00

Q&A