Integrating LLM Agents with Heterogeneous Systems via the Web of Things

S. Ensiye Kiyamousavi*‡, Boris Kraychev*, Jan Bosch‡, Helena Holmström Olsson†
*Big Data for Smart Society Institute (GATE), Sofia University, Sofia, Bulgaria
†Malmö University, Malmö, Sweden
‡Eindhoven University of Technology, Eindhoven, The Netherlands
{ensiye.kiyamousavi, boris.kraychev}@gate-ai.eu
helena.holmstrom.olsson@mau.se
j.bosch1@tue.nl

***Abstract*—Current tool-calling protocols such as the Model Context Protocol (MCP) enable AI agents to invoke external services, but their rigid transport assumptions and lack of mandatory security can be limiting in heterogeneous, event-driven IoT environments underpinning edge intelligence. This work analyzes MCP's limitations and introduces a Web-of-Things (WoT)–based integration layer that uses standardized semantic descriptions and multiple protocol bindings to support dynamic tool discovery and subscription-based events, with security requirements declared via Thing Descriptions. We design a reference architecture around WoT and validate it using a prototype system comprising a web-based front end (implemented with Cesium map) and heterogeneous back-end tools, services, and sensor streams exposed as WoT Things. In our testbed, the prototype sustains high-frequency event interactions, restricts access to protected endpoints in automated security scanning, and completes multi-step tasks with low additional integration overhead.**

I. Introduction

Recent advances in artificial intelligence have accelerated the development of agents capable of interacting with external tools and data sources to fulfill complex user requests [21]. To support such interaction, major platforms have introduced structured mechanisms for tool invocation, such as OpenAI's function calling interface, enabling large language models (LLMs) to retrieve real-time information and execute actions through external services. While effective, these approaches require developers to implement custom connectors and authentication mechanisms for each service, resulting in high engineering overhead and limited flexibility in heterogeneous deployments [18].

To address these challenges, Anthropic introduced the *Model Context Protocol* (MCP) in late 2024 as a standardized interface for connecting AI systems with tools, databases, and APIs. MCP provides a modular abstraction that allows agents to discover and invoke tools dynamically, and has been adopted by several major AI platforms. However, early analyses highlight significant limitations, particularly regarding security, protocol rigidity, and scalability [3], [18]. These issues become especially pronounced in edge and Internet of Things (IoT) environments, where agents must interact with large numbers of heterogeneous devices under constrained network and computing conditions [5].

MCP lacks mandatory authentication, leaving security to implementers and exposing deployments to unauthorized tool invocation and data leakage [18]. Its rigid client–server patterns—primarily STDIO and HTTP-based Server-Sent Events—are ill-suited for event-driven, resource-constrained IoT settings and require custom adapters for continuous streaming or browser interfaces [3]. These shortcomings motivate the search for alternative interoperability frameworks. IoT platforms such as oneM2M, FIWARE, and particularly the W3C Web of Things (WoT) address heterogeneity, scalability and security; WoT provides standardized semantic descriptions, protocol abstraction, event-driven interaction and built-in security using web-native technologies, aligning well with the needs of heterogeneous agentic systems [27].

In this paper, we investigate the suitability of WoT as an integration layer for AI agents operating across heterogeneous tools, services, and real-time data sources. We analyze how WoT's core mechanisms—Thing Descriptions, protocol bindings, event affordances, and security models—can address integration challenges that remain unresolved in MCP-based architectures, particularly in edge and IoT settings. Our focus is on edge-facing agentic orchestration—i.e. how an agent discovers and coordinates heterogeneous edge/IoT resources—rather than on-device model execution or running LLM inference on constrained endpoints. The contributions of this paper are threefold:

1) **Requirements Characterization:** We identify and analyze integration requirements for agentic systems operating in heterogeneous, event-driven environments, and examine their alignment with MCP's design assumptions.
2) **WoT-Based Architecture Pattern:** We propose a WoT-based architectural pattern that enables secure, event-driven, and protocol-agnostic interaction between AI agents and distributed IoT and edge resources.
3) **Case Study and Evaluation:** We implement and evaluate the architecture in a Smart City prototype.

The remainder of this paper is organized as follows. SectionII reviews related work. III-IV presents the methodology and system design and development. SectionV-VI reports the results and the findings. SectionVII concludes the paper.

II. Background and Related Work

A. Tool Integration and Agentic LLMs

Extending large language models with external tools has become a key mechanism for improving reasoning and accessing up-to-date information. Prior work has introduced interleaved reasoning and acting, learned tool use, structured API invocation, and retrieval-augmented generation, enabling LLMs to call external services through schema-defined interfaces and retrieve relevant documents during inference [15], [16], [18], [31], [43]. Agent frameworks such as AutoGPT and Plan-and-Act further demonstrate how LLMs can decompose goals, select tools, and execute multi-step actions autonomously [11], [42]. However, most existing approaches assume controlled environments with well-defined APIs and do not directly address heterogeneous, event-driven, and resource-constrained deployments.

B. Model Context Protocol (MCP)

The Model Context Protocol (MCP) standardizes interaction between LLMs and external tools through a JSON-RPC interface and a host-client-server architecture with dynamically discoverable tools [9], [18]. Although MCP reduces bespoke integration logic, it does not mandate authentication or authorization, leaving security to individual implementations [18], [28]. Its reliance on limited transport patterns and a one-to-one client-server model also makes integration with heterogeneous devices and continuous data streams difficult, especially in edge and IoT settings [3].

C. Web of Things (WoT)

The W3C Web of Things (WoT) addresses interoperability in IoT systems through web-native technologies and semantic Thing Descriptions (TDs), which provide protocol-agnostic descriptions of device and service capabilities across bindings such as HTTP, WebSockets, and MQTT [10]. WoT has been applied in domains including smart homes, industrial IoT, and energy systems, and recent work has explored its use in AI-enabled data processing pipelines [4], [12], [26]. However, whether WoT can serve as a general integration layer for agentic AI systems operating across heterogeneous, edge-centric environments remains underexplored.

III. Methodology

To evaluate whether the Web of Things (WoT) offers an effective integration framework for intelligent agents, we adopt a design science research methodology (DSRM) as described by Peffers et al. [36]. This methodology emphasises an iterative process of problem identification, objective definition, design, development, demonstration, evaluation and communication.

A. Research Objectives and Questions

The study is guided by four research objectives:

RO1: Identify and categorise the limitations of the Model Context Protocol (MCP) in heterogeneous agent-based integration scenarios.

RO2: Select an alternative framework that addresses these limitations.

RO3: Design and implement an architecture that enables AI agents to interact dynamically with heterogeneous external systems, real-time data streams and user-facing interfaces.

RO4: Assess the performance and security of the proposed architecture.

These objectives motivate the following research questions:

RQ1: How well does a WoT-based integration architecture support real-time, secure, and reliable AI-agent interaction with heterogeneous resources?

B. Research Design

1) Problem Identification and Motivation: We began by identifying the need for an integration framework beyond MCP in heterogeneous, agent-based settings. Through a structured literature review and an engineering sprint, we catalogued and validated MCP limitations; these results (Section IV-A) inform the baseline requirements for the framework selection in Section IV-B.

2) Definition of the Objectives of a Solution: From the identified limitation categories (Section IV-A), we formulated baseline requirements for an integration framework suitable for real-world heterogeneous deployments. Recognizing that IoT platforms address similar challenges—heterogeneous protocols, real-time/asynchronous updates and security—we compared representative frameworks and, guided by these requirements, selected WoT; the corresponding mapping of MCP limitations to WoT mechanisms is detailed in Sections IV-B and IV-C.

3) Design and Development: Finally, we translated the requirements and conceptual mapping into a concrete architecture and working prototype (Sections IV-D and III-B4). Using WoT as the reference framework, our design exposes capabilities as *Things* with Thing Descriptions (TDs) and performs interactions via standardized affordances (properties, actions and events) with protocol bindings defined by WoT forms and binding templates.

4) Implementation and Evaluation: This phase specifies how we verify that our proposed WoT-based architecture satisfies the integration requirements distilled earlier in Section III-B1) and thus provides evidence for the research questions.

The assessment uses a working prototype whose *front end* is a live Cesium-powered 3D map with a chat panel, while the *back end* comprises the WoT Gateway and AI Core Host together with Domain/Resource Things representing heterogeneous resources such as map-control tools, geocoding and weather services, and live environmental sensor feeds, as detailed in Section IV-D.

When a user types a request in the chat panel (e.g., *Zoom to the Eiffel Tower* or *What's the weather right here?*), the LLM Thing receives updates from map(UI) about the current location and reasons over the conversation context, inspects the WoT Thing Descriptions advertised by the Gateway, and selects the most appropriate action sequence (such as

`getCoordinates` → `flyTo` or `reverseGeocode` → `getWeather`) or use current sensor status through events. Each action is invoked through WoT primitives, and any resulting events are propagated back to the browser.

We evaluated the prototype in three complementary dimensions:

E1. **Performance.**
We compared SSE—the only unidirectional event-streaming option defined by MCP, requiring a dedicated HTTP connection per subscriber—with MQTT, a lightweight broker-based publish/subscribe protocol that WoT supports through official bindings [34]. Because SSE incurs significant overhead in high-frequency, multi-subscriber settings, we stress-tested both protocols under identical conditions with 50 subscribers and one publisher. We recorded send and receive timestamps for each message to compute throughput and latency metrics.

- end-to-end latency distribution (p_{50}, p_{95}, p_{99});
- loss ratio $(1 - \frac{received}{expected}) * 100\%$;

Test runner iterates over the Cartesian product:

Message size	B	100
Publish rate	msg/s	$100, 500$
Test duration	s	60
Iterations		5

yielding 20 total tests. The comparative results of this experiment are presented in Section V-A.

E2. **Security.** As outlined in Section IV-C, one of the most important limitations of MCP is the optional nature of its authentication and access control mechanisms. In addition, if there is a need to build any custom adapter (in our prototype a custom adapter to transmit data to the user interface), we need to rebuild an authentication mechanism (Section III-B1). So, if the developer does not build this mechanism correctly, there will be a chance of unauthorized access to the system. In this experiment, we perform an autonomous security assessment using the OWASP ZAP framework[1] to validate if `securityDefinitions` in WoT are tied to all attributes (properties, actions, events) in a registered "Thing" or not. The goal is to find out if ZAP can find an unauthorized access to any of the defined Things or their attributes. To ensure reproducibility, the test plan was defined as a YAML configuration. The configuration executes a full multi-phase security evaluation, including both passive and active vulnerability discovery. The results of this evaluation are reported in Section V-B.

E3. **Task-Effectiveness.** To evaluate our WoT-based architecture's ability to enable AI agents to complete user tasks, we designed a benchmark of 30 representative interaction scenarios, categorized by complexity:

- **Simple** (17 tasks): Single tool call (e.g., weather lookup, zoom, sensor loading).
- **Medium** (11 tasks): Two sequential tool calls (e.g., geocoding + navigation, parameterized sensor queries).
- **Complex** (2 tasks): Three or more chained calls requiring multi-step reasoning (e.g., navigate to a location and retrieve air pollution data).

We ran each tasks for 10 time with 10 different seeds. For each task, we recorded:

(a) **Success rate** and **tool sequence correctness**;
(b) **Mean tool calls per task**;
(c) **Latency breakdown**: end-to-end time, model inference time, and tool execution time;
(d) **Resource utilization**: CPU and memory consumption.

Results are presented in Section V.

Prototype configuration. We selected `Qwen/Qwen3-32B` [35] as the core model due to its strong multi-round reasoning performance and its open-access compatibility. The model was hosted using the VLLM engine, running on a separate server: an NVIDIA A100 GPU with 80 GB of VRAM. The WoT Things and UI were hosted on a virtual machine (VM) with a 4-core CPU and 32 GB of memory. We ran the experiments also on the Same VM.

Our evaluation focuses on the edge-facing integration/orchestration layer. We do not evaluate on-device LLM inference on constrained endpoints

IV. Design and Development

This section presents the detailed design and development activities and their outcomes, corresponding to Section III-B.

A. Requirements and Problem Analysis

This phase was designed to produce a literature-based and practice-driven understanding of MCP limitations in heterogeneous, agent-based integration settings, and to translate those limitations into concrete requirements for a WoT-based alternative.

Literature review protocol.: Search strategy. We queried five major repositories—ACM Digital Library, IEEE Xplore, Scopus, SpringerLink, and arXiv—using the exact phrase `"Model Context Protocol"` across titles, abstracts, and keywords. Because MCP is a recent protocol introduced in late 2024, the majority of the identified records were preprints hosted on arXiv.

Screening and eligibility. DOI-based de-duplication yielded 46 unique records. We screened titles and abstracts using the following criteria: (i) MCP is a central topic of investigation; (ii) the full text is available in English; and (iii) the study discusses MCP limitations, challenges, or open problems (rather than focusing solely on architecture or applications). This process resulted in 13 studies included for analysis.

Supplementary sources. To incorporate practitioner perspectives, we added Anthropic's public MCP specification (revision 2025-03-26) and its accompanying developer FAQ.

Data extraction and coding. Each included publication was coded for statements describing limitations, failure modes, or open challenges. Across the reviewed literature, two limitation categories appeared consistently:

[1]https://www.zaproxy.org

(L1)Security [6], [7], [17], [18], [22], [23], [28], [33], [39], [41]
(L2)Protocol rigidity [3], [8], [19]

In addition, analysis of the MCP specification contributed two further limitations relevant to our target setting:

(L3)Limited event-driven communication
(L4)Backend-only integration

Practical problem analysis.: To validate whether these limitations surface in practice, we conducted an engineering sprint. The sprint implemented a Cesium-based 3D map front end connected to an MCP service layer using JSON-RPC over SSE. During this implementation, we observed that the constraints identified in the literature translate into concrete engineering bottlenecks in UI-centric, real-time scenarios. In particular:

1) *Absence of browser-side SDKs.* UI updates (chat payloads and map status changes) required a custom two-way adapter to transmit JSON-RPC, reflecting both protocol rigidity (L2) and backend-only integration (L4).
2) *Ad-hoc security plumbing.* Introducing a custom adapter required duplicating authentication and channel-encryption logic, illustrating the security limitations (L1).

B. Framework Selection

Building on the limitations and requirements identified in Section IV-A, we selected an alternative integration framework by applying a requirement-driven comparison of mature interoperability standards. As [9] notes, LLM-based agents require robust and standardized protocols to integrate tools, share contextual data, and coordinate tasks across heterogeneous systems. While prior work has proposed improvements and extensions to MCP [28], [38], these proposals primarily address isolated issues and do not eliminate the fundamental limitations observed in real-world, heterogeneous deployments.

Selection criteria.: From Section IV-A, we derived three core requirements that an integration framework must satisfy in heterogeneous settings:

1) **Security by design:** the framework should provide built-in authentication and authorization mechanisms that can be consistently enforced.
2) **Protocol flexibility:** the framework should support diverse communication protocols to reduce the need for custom adapters in heterogeneous environments.
3) **Event awareness:** the framework should support asynchronous interactions and continuous data streams through native event-oriented patterns.

Candidate frameworks.: Given these criteria, we drew from the Internet of Things (IoT) literature, where interoperability, heterogeneous protocol support, and secure integration have been studied extensively. We considered representative and widely deployed frameworks: *W3C Web of Things (WoT)*, *oneM2M*, *FIWARE (NGSI-LD)*, and *OCF/IoTivity*. These frameworks are frequently compared in empirical studies and surveys, which evaluate how each addresses interoperability, event handling, and security in heterogeneous deployments [13], [20], [44].

Comparison summary.: All four frameworks pursue interoperability but emphasise different aspects. The W3C Web of Things (WoT) uses Web technologies and semantic descriptions to abstract protocol diversity and enable secure, event-driven interactions; oneM2M provides a unified service layer with built-in notifications and fine-grained access control; FIWARE focuses on context data management and real-time publish/subscribe; and OCF/IoTivity delivers device-level interoperability with embedded security [2], [14], [24], [25]. Comparative studies highlight complementary strengths—such as WoT's flexible, Web-linked interaction and FIWARE's robust context management—and note that no single framework dominates, as bridging mechanisms are often required [20], [44]. They also observe that asynchronous interaction is supported through different mechanisms (e.g., broker-based publish/subscribe vs. observe-style) and that WoT and FIWARE leverage Web-based security, whereas oneM2M and OCF embed security within their stacks [13].

Selection rationale.: WoT was chosen because it explicitly abstracts protocol differences using web-native technologies, eliminating the need for custom adapters; it treats events as first-class, supporting subscription-based real-time updates suitable for interactive tool-driven workflows; and its Thing Descriptions allow authentication and encryption requirements to be declared and consistently enforced across properties, actions, and events.

C. Conceptual Mapping of MCP Limitations to WoT Capabilities

After selecting WoT as the reference integration framework (Section IV-B), we constructed a *conceptual map* that links the MCP limitation categories identified in the problem analysis (Section IV-A) to concrete WoT capabilities. The purpose of this mapping is twofold: (i) to make the design rationale explicit by showing how WoT addresses each integration challenge, and (ii) to guide architectural decisions in the subsequent design and implementation of the proposed system (Section IV-D).

1) Protocol rigidity → Protocol abstraction via Thing Descriptions and Binding Templates: MCP enables AI agents to interact with external tools through a single JSON-RPC message format. The current specification defines two standard transports: `stdio` (for local subprocesses) and streamable HTTP (`POST` + optional SSE). In this model, an agent issues a JSON-RPC request and the MCP server returns a corresponding JSON-RPC response with the result [9]. While this uniform interface simplifies tool invocation, it also constrains interoperability: if a target system does not support the defined transports (e.g., a REST microservice, a WebSocket stream, or an OPC UA controller), a custom adapter is required to translate the native protocol into MCP's JSON-RPC framing [3].

WoT is explicitly designed to address interoperability across heterogeneous protocols and devices. A WoT Thing Description (TD) provides a protocol-independent, semantic de-

scription of a device or service's affordances—properties, actions, and events—while WoT Binding Templates specify how each interaction is realized over concrete protocols such as HTTP, CoAP, MQTT, Bluetooth, or OPC UA [32], [40]. For example, a service may expose a single logical action (e.g., `getWeather`) but provide multiple protocol bindings, enabling clients to choose a suitable protocol without requiring protocol-specific wrappers. This decoupling supports dynamic discovery and interaction based on TD semantics rather than transport-specific integration code.

2) Limited event-driven communication → First-class event affordances and subscriptions: In MCP's HTTP transport, clients typically issue a `POST` and block until a single response is returned. MCP's main streaming mechanism is Server-Sent Events (SSE), which allows a server to push a sequence of messages over a single connection [18]. However, SSE is one-way (server-to-client) and does not natively support subscription management (e.g., subscribe/unsubscribe), event filtering, or shared observation by multiple independent consumers without duplicating requests [30].

WoT treats events as a first-class interaction type. Through an *event affordance*, a consumer can subscribe to asynchronous notifications, receive event payloads, and later unsubscribe. Each event payload is formally described in the Thing Description and can be validated at runtime [29]. Because TDs are protocol-agnostic, a single event definition can be mapped to different protocol bindings (via `forms` and binding templates), enabling flexible deployment across diverse communication infrastructures [40].

3) Backend-only integration → Web-native consumption patterns and UI-facing interoperability: MCP is primarily positioned as a back-end protocol for AI agents to invoke tools via structured function calls in hosts such as IDE assistants or chat systems [1]. In practice, MCP integration in browser-based interfaces is constrained: no official browser-side SDK exists, and the reference TypeScript client depends on Node-specific modules. As a consequence, front-end applications must typically rely on server-side proxies that speak MCP's JSON-RPC framing, manage OAuth tokens, and enforce policy before returning results to the browser. In UI-centric systems, this architecture pushes developers toward ad hoc adapters to support bidirectional interaction and streaming updates.

In contrast, WoT is designed around web-native consumption and exposure of Things. A browser-based client can fetch Thing Descriptions and interact with Things through standard web patterns, while binding templates allow consistent interaction across heterogeneous components [37]. This enables user interfaces to directly invoke actions and subscribe to events using the same interaction model as other consumers, reducing the need for custom UI bridges and promoting uniform end-to-end integration.

4) Security limitations → Declared and enforceable security via Thing Descriptions: Security concerns are prominent in MCP deployments. The specification allows servers to expose endpoints without mandatory authentication, and tool

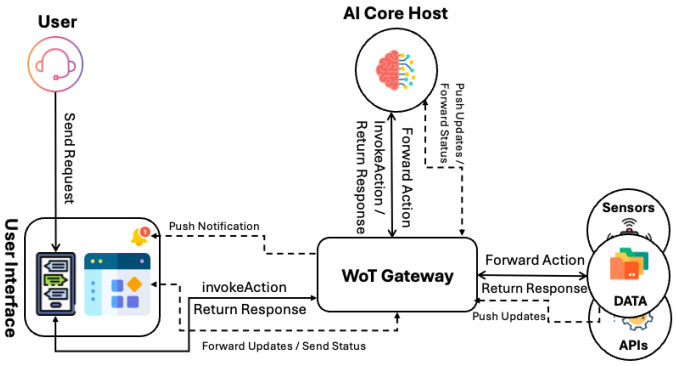

Fig. 1. Proposed three-layer integration architecture.

definitions can be distributed as plain JSON. This creates attack surfaces such as spoofed services and malicious tool swapping, which can lead to data exfiltration or arbitrary command execution [6], [23], [28]. Moreover, MCP recommends an OAuth 2.1 profile for its HTTP binding but explicitly discourages applying OAuth to local `stdio` transport [1]. In heterogeneous deployments that require additional protocol bridges, developers often implement bespoke security plumbing (authentication, token refresh, mutual TLS, policy checks), increasing the likelihood of misconfiguration and privilege-escalation paths [6], [18].

WoT addresses security through explicit declarations in the Thing Description. In the Thing Description 1.1 specification[2], a TD must include at least one `securityDefinitions` block (e.g., bearer token, pre-shared key). Once declared, the selected scheme becomes the default security requirement for all properties, actions, and events of that Thing. This reduces the risk of inadvertently exposing unprotected endpoints and provides clients with a clear contract specifying which credentials are required to access a Thing's resources.

D. Architectural Design

Based on the conceptual mapping in Section IV-C, we designed an integration architecture that follows the W3C WoT reference model. In this model, every functional block that offers capabilities is exposed as a WoT *Thing* described by a Thing Description (TD), while a WoT *servient* can act as both a client (consumer) and a server (exposer) to orchestrate interactions. Figure 1 summarizes the main components and their interactions.

Component roles.: The proposed system comprises four WoT-aligned components:

1) **WoT Gateway (Orchestrator Servient).** The Gateway is the coordinating middleware that turns a set of isolated Things into a coherent system. It exposes a WoT interface to upstream consumers while internally consuming the TDs of downstream Things (e.g., the LLM Thing, data Things, and external tool Things). In its dual client/server role, the Gateway: (i) forwards

[2]https://www.w3.org/TR/wot-thing-description11/

`invokeAction`, `readProperty`, and event subscription requests across Things (protocol and payload mediation); (ii) centralizes security and policy enforcement (e.g., BasicAuth, input validation); (iii) orchestrates multi-step tool interactions by chaining Thing affordances to fulfill a user intent; (iv) provides discovery and caching of Thing Descriptions; and (v) aggregates observability signals (e.g., traces and metrics) for runtime monitoring.

2) **AI Core Host (LLM Thing).** The AI Core Host provides the system's reasoning and decision-making capability as a WoT-compliant Thing. Rather than directly controlling devices, the LLM Thing operates as an *agentic reasoning component* that interprets context, plans actions, and delegates execution to edge-exposed Things.
   The LLM Thing exposes a primary WoT action that receives a user message together with contextual information mediated by the Gateway, including relevant Thing Descriptions and recent observations. Based on this bounded context, the LLM performs the following steps: (i) reasons over the current state and the set of available affordances described in the TDs; (ii) selects one or more actions to be executed on downstream Things; (iii) returns a structured action plan to the Gateway for execution; and (iv) produces a natural-language response reflecting the outcome.
3) **User Interface (UI Thing).** The UI is a Cesium-based 3D map and chat front end that acts as a WoT *consumer*. It fetches Thing Descriptions, invokes actions, and subscribes to events. The UI can publish its current map state as context to the LLM Thing (via Gateway mediation) and reacts to responses and emitted events by updating the visible map state and chat messages in real time.
4) **Domain/Resource Things (tools, services, devices, sensors).** Existing capabilities—including map controls, APIs, and sensor streams—are wrapped as WoT Things. Each exposes its own TD defining relevant properties, actions, and events. This abstraction makes heterogeneous resources uniformly accessible, allowing the LLM Thing and the Gateway to orchestrate them through standard WoT interaction patterns rather than tool-specific integration code.

V. Results

Following *evaluation* settings mentioned in III-B4, this section reports the results of our experiments.

TABLE I
Comparison of MQTT and SSE at different message rates

Metric	Rate 100		Rate 500	
	MQTT	SSE	MQTT	SSE
Expected	300000	300000	1500000	1500000
Received (%)	99.78	98.49	98.42	87.98
Loss Ratio (%)	0.13	1.51	1.58	12.02
p50 (ms)	1.0	1.0	1.0	1.0
p95 (ms)	2.8	2.6	2.0	2.2
p99 (ms)	3.6	3.6	3.8	4.2

A. Performance Evaluation

Table I compares the performance of MQTT and SSE at message rates of 100 and 500 messages per second. Both protocols perform reliably at 100 msg/s, with MQTT achieving a slightly higher delivery rate (99.78%) than SSE (98.49%). At this rate, the latency metrics (p50, p95, and p99) are comparable across both protocols, remaining under 4 ms. However, as the message rate increases to 500 msg/s, the differences become more pronounced. MQTT maintains a high receive rate of 98.42%, while SSE drops to 87.98%, reflecting a much higher loss ratio of 12.02% compared to MQTT's 1.58%. Despite this, latency remains low for both protocols, indicating that the bottleneck under higher load for SSE is more about message loss than delay.

B. Security Evaluation

To evaluate the robustness of the authentication mechanisms in our WoT-based prototype, we conducted an autonomous security audit using the OWASP ZAP framework. ZAP was configured to simulate an unauthenticated adversary attempting to discover and exploit the system through automated URL crawling, passive analysis, and active vulnerability scans. The automation executed a multistage pipeline: It began with deep URL discovery using traditional and AJAX-based spiders, proceeded with passive scanning for misconfigurations and insecure headers, and ended with an active scan leveraging the *Default* policy.

1) URL Discovery: The spider identified 23 endpoints on the targeted route. AJAX crawling did not produce new dynamic paths.
2) Vulnerability scan: The activescan did not produce high-severity vulnerabilities. A medium-level warning was raised for the use of Basic Auth, which is a known weak scheme if not used over HTTPS. However, in our case, this was intentional and was meant for internal evaluation.
3) Authentication: ZAP was denied access to all endpoints without defined credentials.

C. Task-effectiveness evaluation

As shown in TableII, across all tasks the agent achieved 100% success and always invoked the correct tool sequence. On average it required about three reasoning turns and 1.64 tool calls per task. The mean completion time was 7.49s, and 97% of this latency came from the LLM itself rather than tool overhead. Resource utilization (Table III) was minimal across the board: CPU usage averaged 0.7% (with a maximum of 6.1%), and memory consumption remained stable at about 44.3MB. Latency increased with complexity—simple tasks averaged 6.14s, medium tasks 8.75s and complex tasks 12.08s—while simple tasks exhibited larger variance due to network delays; medium and complex tasks were more consistent.

VI. Discussion

This study assessed whether the Web of Things (WoT) can serve as an effective integration layer for large-language-model

TABLE II
EVALUATION SUMMARY BY TASK COMPLEXITY (S=SECONDS). EACH METRIC IS AVERAGED OVER TEN RUNS PER TASK.

Metric	Aggregate	Simple	Medium	Complex
Number of tasks	30	17	11	2
Success rate	100 %	100 %	100 %	100 %
Tool sequence correct	100 %	100 %	100 %	100 %
Mean reasoning turns	≈ 3	≈ 3	3	4
Mean tool calls	1.64	≈ 1.6	2	3
Mean time (s)	7.49	6.14	8.75	12.08
Std. deviation (s)	2.48	2.26	0.70	1.06
95%CI (s)	7.21–7.78	5.80–6.48	8.62–8.89	11.56–12.60
Mean model share (%)	97.3	98.6	96.9	97.0

TABLE III
RESOURCE UTILISATION ACROSS ALL RUNS (30 TASKS $\times$ 10 RUNS).

Metric	Mean	Std. dev.	Min–Max
CPU utilisation (%)	0.7	0.7	0.3–6.1
Memory usage (MB)	44.3	0.1	44.1–44.5

(LLM) agents interacting with heterogeneous resources. We proposed and implemented a WoT-based architecture (Section IV-D) and evaluated it along three dimensions: (i) performance under high-frequency event streaming, (ii) security and access control, and (iii) task effectiveness in representative user scenarios.

Performance implications: The performance results show that WoT's protocol abstraction enables communication mechanisms better suited to event-driven, multi-subscriber environments than those natively supported by MCP. While SSE performs adequately at moderate message rates, its unidirectional, connection-per-subscriber design leads to substantial message loss under higher load. In contrast, MQTT—used through WoT binding templates—maintains high delivery rates even at 500 messages per second. This supports the claim that separating interaction semantics from transport protocols improves scalability in real-time and IoT-like deployments without changing the agent's reasoning logic.

Security considerations: The security evaluation indicates that WoT's explicit security declarations in Thing Descriptions support more consistent and enforceable access control. By associating authentication schemes with a Thing and applying them across properties, actions, and events, the risk of unintentionally exposed endpoints is reduced. The OWASP ZAP assessment confirmed that unauthenticated access was effectively prevented, even with browser-based clients and custom orchestration logic. Compared with MCP-based deployments, where security is optional and often re-implemented in adapters, this highlights the benefit of a security-by-design approach.

Task effectiveness and agent behavior: The results suggest that *WoT Thing Descriptions* provide sufficient semantic information for an LLM-based agent to map user intent to appropriate affordances and execute the required tool sequences. Achieving 100% success and 100% tool-sequence correctness across 30 tasks indicates strong compatibility between the WoT interaction model and agentic planning. Tasks typically completed in about three reasoning turns with 1.64 tool calls, while 97% of end-to-end latency came from the LLM rather than the integration layer. Resource usage remained minimal, indicating negligible orchestration overhead.

Implications for MCP-based ecosystems: Taken together, the findings suggest that WoT addresses several structural limitations of MCP in heterogeneous and edge-centric settings. Rather than replacing MCP, WoT can serve as a complementary integration layer that shifts protocol diversity, event handling, and security enforcement into a standardized interoperability framework. This allows MCP-style tool invocation to remain useful in controlled back-end environments, while WoT supports broader deployment across devices, services, and user interfaces.

Limitations and future directions: Despite its advantages, WoT introduces additional complexity, particularly in authoring and maintaining high-quality Thing Descriptions and managing large numbers of distributed Things. Future work should investigate automated or semi-automated TD generation, hybrid architectures combining WoT with context brokers or agent coordination frameworks, and longitudinal studies of operational overhead, maintainability, and developer experience.

VII. CONCLUSION

This paper shows that the W3C Web of Things (WoT) can serve as an effective integration layer for AI agents operating across heterogeneous and edge-centric environments. Our prototype-based evaluation indicates that the architecture can sustain high-frequency interactions with low latency, prevent unauthorized endpoint access in automated security testing, and reliably support multi-step tool orchestration with minimal integration overhead. In general, WoT complements MCP by shifting interoperability, event handling, and security enforcement into a standardized framework, making agentic systems more scalable and reliable in real-world deployments.

ACKNOWLEDGMENTS

This research is supported by the GATE project, which is funded by the Horizon 2020 WIDESPREAD-2018-2020 TEAMING Phase 2 programme under grant agreement no.857155, and the programme "Research, Innovation and Digitalisation for Smart Transformation" 2021-2027 (PRIDST) under grant agreement no. BG16RFPR002-1.014-0010-C01 and DS4SSCC-DEP project, funded by the European Union Digital Europe Work Programme 2021-2022 under Grant Agreement no. 101123342.

REFERENCES

[1] Model context protocol specification, version 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18, June 2025. Accessed 2025-07-16.

[2] Web of things (wot) architecture 1.1. W3C Recommendation, December 5, 2023. https://www.w3.org/TR/wot-architecture/.

[3] Arash Ahmadi, Sarah Sharif, and Yaser M Banad. Mcp bridge: A lightweight, llm-agnostic restful proxy for model context protocol servers. *arXiv preprint arXiv:2504.08999*, 2025.

[4] Luca Bedogni and Federico Chiariotti. A web of things approach for learning on the edge-cloud continuum. *Future Generation Computer Systems*, page 107736, 2025.

[5] Akshat Bhat, Aishee Mondal, and Aniket Tripathy. Llm agents for internet of things (iot) applications. 2025.

[6] Manish Bhatt, Vineeth Sai Narajala, and Idan Habler. Etdi: Mitigating tool squatting and rug pull attacks in model context protocol (mcp) by using oauth-enhanced tool definitions and policy-based access control. *arXiv preprint arXiv:2506.01333*, 2025.

[7] Ivo Brett. Simplified and secure mcp gateways for enterprise ai integration. *arXiv preprint arXiv:2504.19997*, 2025.

[8] Enfang Cui, Yujun Cheng, Rui She, Dan Liu, Zhiyuan Liang, Minxin Guo, Tianzheng Li, Qian Wei, Wenjuan Xing, and Zhijie Zhong. Agentdns: A root domain naming system for llm agents. *arXiv preprint arXiv:2505.22368*, 2025.

[9] Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: Model context protocol (mcp), agent communication protocol (acp), agent-to-agent protocol (a2a), and agent network protocol (anp). *arXiv preprint arXiv:2505.02279*, 2025.

[10] Nazmi Ekren, Mehmet Sensoy, and Tahir Cetin Akinci. Smart buildings using web of things with. net core: A framework for inter-device connectivity and secure data transfer. *Information*, 16(2):123, 2025.

[11] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

[12] Leonhard Esterbauer, Gernot Steindl, and Wolfgang Kastner. Improving energy community interoperability by utilizing web of things. *e & i Elektrotechnik und Informationstechnik*, 140(5):425–431, 2023.

[13] Ali Farhat, Abdelrahman Eldosouky, Jason Jaskolka, Mohamed Ibnkahla, and Ashraf Matrawy. Open source horizontal iot platforms: A comparative study on functional requirements. *arXiv preprint arXiv:2209.06384*, 2022.

[14] FIWARE Developers. Fiware ngsi-ld tutorials – subscriptions and notifications. https://fiware-tutorials.readthedocs.io/ngsi-ld-subscriptions/index.html, 2021.

[15] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

[16] Shailja Gupta, Rajesh Ranjan, and Surya Narayan Singh. A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions. *arXiv preprint arXiv:2410.12837*, 2024.

[17] John Halloran. Mcp safety training: Learning to refuse falsely benign mcp exploits using improved preference alignment. *arXiv preprint arXiv:2505.23634*, 2025.

[18] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*, 2025.

[19] Cheonsu Jeong. A study on the mcp x a2a framework for enhancing interoperability of llm-based autonomous agents, 2025.

[20] Jahoon Koo and Young-Gab Kim. Resource identifier interoperability among heterogeneous iot platforms. *Journal of King Saud University-Computer and Information Sciences*, 34(7):4191–4208, 2022.

[21] Naveen Krishnan. Ai agents: Evolution, architecture, and real-world applications. *arXiv preprint arXiv:2503.12687*, 2025.

[22] Vineeth Sai Narajala and Idan Habler. Enterprise-grade security for the model context protocol (mcp): Frameworks and mitigation strategies. *arXiv preprint arXiv:2504.08623*, 2025.

[23] Vineeth Sai Narajala, Ken Huang, and Idan Habler. Securing genai multi-agent systems against tool squatting: A zero trust registry-based approach. *arXiv preprint arXiv:2504.19951*, 2025.

[24] oneM2M. Published specifications. https://www.onem2m.org/technical/published-specifications, 2025.

[25] Open Connectivity Foundation. Ocf security specification v2.1.2. https://openconnectivity.org/specs/OCF_Security_Specification_v2.1.2.pdf, April 2020.

[26] Guadalupe Ortiz, Juan Boubeta-Puig, Javier Criado, David Corral-Plaza, Alfonso Garcia-de Prado, Inmaculada Medina-Bulo, and Luis Iribarne. A microservice architecture for real-time iot data processing: A reusable web of things approach for smart ports. *Computer Standards & Interfaces*, 81:103604, 2022.

[27] Firoj Parwej, Nikhat Akhtar, and Yusuf Perwej. An empirical analysis of web of things (wot). *International Journal of Advanced Research in Computer Science*, 10(3), 2019.

[28] Brandon Radosevich and John Halloran. Mcp safety audit: Llms with the model context protocol allow major security exploits. *arXiv preprint arXiv:2504.03767*, 2025.

[29] Fady Salama, Ege Korkan, Sebastian Käbisch, and Sebastian Steinhorst. Stateful-wot: Capturing the behavior of highly dynamic cyber-physical systems. In *2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–8. IEEE, 2024.

[30] Ridwan Sanjaya, Andre Kurniawan Pamudji, and Evelyn Vania. Comparative analysis and practical implementation of lot communication using http/2 and mqtt. In *2024 7th International Conference on Green Technology and Sustainable Development (GTSD)*, pages 221–226. IEEE, 2024.

[31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[32] Luca Sciullo, Lorenzo Gigli, Federico Montori, Angelo Trotta, and Marco Di Felice. A survey on the web of things. *IEEE Access*, 10:47570–47596, 2022.

[33] Hao Song, Yiming Shen, Wenxuan Luo, Leixin Guo, Ting Chen, Jiashui Wang, Beibei Li, Xiaosong Zhang, and Jiachi Chen. Beyond the protocol: Unveiling attack vectors in the model context protocol ecosystem. *arXiv preprint arXiv:2506.02040*, 2025.

[34] OASIS Standard. Mqtt version 3.1. 1. *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, 1:29, 2014.

[35] Qwen Team. Qwen3 technical report, 2025.

[36] Jan Vom Brocke, Alan Hevner, and Alexander Maedche. Introduction to design science research. *Design science research. Cases*, pages 1–13, 2020.

[37] W3C Web of Things Community Group. Communications Technologies for the Web of Things. https://www.w3.org/community/wot/wiki/Communications_technologies_for_the_Web_of_Things, 2024. Accessed: 2025-07-16.

[38] Libo Wang. Towards humanoid robot autonomy: A dynamic architecture integrating continuous thought machines (ctm) and model context protocol (mcp). *arXiv preprint arXiv:2505.19339*, 2025.

[39] Zihan Wang, Hongwei Li, Rui Zhang, Yu Liu, Wenbo Jiang, Wenshu Fan, Qingchuan Zhao, and Guowen Xu. Mpma: Preference manipulation attack against model context protocol. *arXiv preprint arXiv:2505.11154*, 2025.

[40] World Wide Web Consortium. Web of Things (WoT) Binding Templates. https://www.w3.org/TR/wot-binding-templates/, 2024. W3C Group Note, May 28, 2024.

[41] Junjie Xiong, Changjia Zhu, Shuhang Lin, Chong Zhang, Yongfeng Zhang, Yao Liu, and Lingyao Li. Invisible prompts, visible threats: Malicious font injection in external resources for large language models. *arXiv preprint arXiv:2505.16957*, 2025.

[42] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

[43] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. *URL https://arxiv. org/abs/2210.03629*, 2023.

[44] Ivan Zyrianoff, Alexandre Heideker, Luca Sciullo, Carlos Kamienski, and Marco Di Felice. Interoperability in open iot platforms: Wot-fiware comparison and integration. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 169–174. IEEE, 2021.