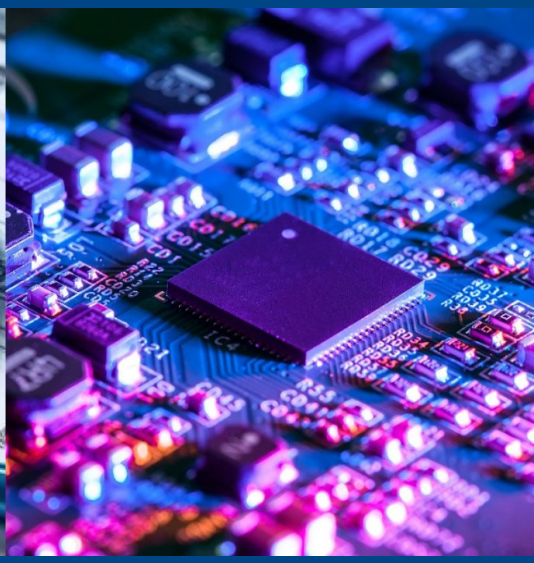
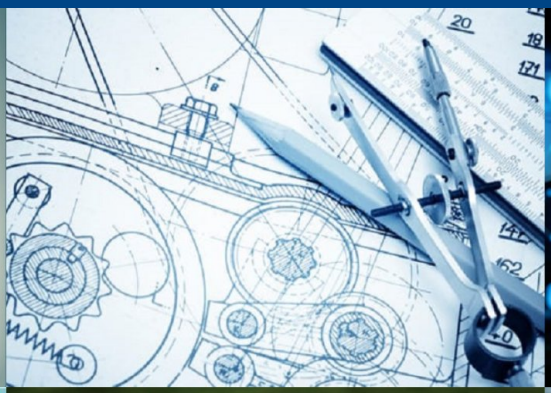
ISSN: 2582-7219

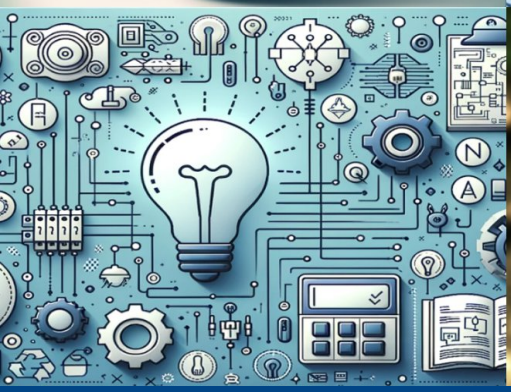




International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.206 **Volume 9, Issue 4, April 2026**

© 2026 IJMRSET | Volume 9, Issue 4, April 2026| DOI:10.15680/IJMRSET.2026.0904435

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Sol: An open modular framework for agentic AI integration with physical home environments via the Model Context Protocol

Mohamed Muaath Rifath L M, Mohamed Hussain M, Mohamed Wafiq H,

Department of Electronics and Communication Engineering, Aalim Muhammed Salegh College of Engineering, Chennai, Tamil Nadu, India

ABSTRACT: Thanks to low cost of embedded hardware, it is now possible to create a fully-fledged smart home. Despite such positive trend in the industry, existing platforms for smart home applications are still fragmented, closed, and unfriendly to AI agent development. In this paper, we introduce Sol, an agent-based smart home framework using modular open-source hardware based on ESP32. The proposed architecture consists of four tiers of systems, which could be used separately without mutual dependencies. At the hardware tier, the components such as ESP32-S3 chip, relay board, speakers and omnidirectional microphone module, communicate through the MQTT protocol and are connected via X.509 mTLS certificate-based mutual authentication. The backend is implemented in Go language and hosted behind Traefik reverse proxy. WebSocket hubs are scalable and synchronized via Redis Pub/Sub mechanism. Telemetry data is stored in PostgreSQL using TimescaleDB hypertable engine. At the frontend tier, we use the Next.js framework for the web application development and React Native for mobile applications development. These components cover areas such as device management, automations management and voice communication with device. ESP32 firmware based on ESP-IDF is compiled using a Go language-based builder tool using text/template package and Sprig library. Compiled binary files and their respective sources are stored in MinIO object storage service. But most crucial part of our work is agent ecosystem, where Sol provides fine-grained access control using permission tools based on MCP protocol allowing external agents like OpenClaw and Claude access to devices of your home..

KEYWORDS: smart home automation, Model Context Protocol, agentic AI, ESP32, modular IoT, MQTT, WebSerial, retrieval-augmented generation

I. INTRODUCTION

Ten years of low-cost microcontrollers and Wi-Fi chips enabled the smart home ecosystem to develop. Using it in daily life, however, is a totally different matter. In a typical smart home equipped with products from five vendors, there are five separate apps, each with its own registration procedure, update cycle, and cloud backend [1, 2]. Convenience turned into a constant need to remember multiple credentials and keep devices running despite their tendency to stop functioning after updating.

Vendor lock-in does not only make user experience poor; it also has a negative impact on the environment. According to the UN Global E-waste Monitor, 62 million tonnes of electronic waste were produced worldwide in 2022, which corresponds to an 82% increase compared to 2010 [3]. Only about 20% of the collected waste was recycled. Smart lights, plugs, and other IoT products significantly contribute to the issue. Such devices are usually sealed and contain integrated radios and microcontrollers; once a service ceases to exist or standards change, their lifetime ends immediately. As a rule, smart bulbs never get repaired; they are always thrown away when malfunctioning. At present, small electronics account for almost a third of global e-waste volume [3].Several open-source projects attempted to tackle the vendor lock-in problem. Home Assistant [4] provides support for more than 2,000 devices and can be run on Raspberry Pi. ESPHome [5] generates firmware for ESP32 boards based on YAML files. While these solutions are useful, they require some technical knowledge from the user. They involve configuring devices in YAML formats, flashing boards via command line, and managing mdns and Linux-based systems. Brush et al. [6] show that users with technical skills also struggle during initial setup; later CHI research conducted by Desolda et al. [12] confirms that setup complexity remains among top reasons for users' refusal to configure smart homes. The necessary tools exist, yet they are hardly accessible for ordinary users.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

In addition to the existing gap, another one is emerging due to developments in agent- based artificial intelligence (AI). Current state-of-the-art large language model (LLM) agents are capable of planning, using tools, and executing multi-step tasks [7]; however, they lack a standard protocol allowing them to interact with a physical environment. There is no standard way of requesting such simple information as a room temperature or turning a light switch off. The Model Context Protocol (MCP), introduced by Anthropic in November 2024 and adopted by both OpenAI and Google DeepMind under the auspices of the Linux Foundation Agentic AI Foundation [8, 40], standardizes agent's interaction with tools. However, no home automation platforms integrate this solution for physical devices so far.

This paper proposes Sol, an open-source platform intended to solve many challenges simultaneously: vendor lock-in, increased e-waste volume, configuration complexity, accessibility for ordinary users, and the aforementioned problem of bridging the gap between AI agents and physical world.

The key contributions of this work include:

1. Modular hardware design where Sol replaces standard switchboards with hot- pluggable ESP32-S3 nodes allowing the installation of eight relay channels and optional GPIO-based sensors on each node. A failed module is easily replaceable with minimal intervention;
2. Zero-install provisioning process based on browser usage of WebSerial API to program the hardware. No integrated development environment (IDE), command line flashing tools, and additional software are required. Future updates will be provided using over-the-air technique via a MinIO-based firmware repository;
3. One single application for controlling all devices through a Next.js web dashboard and React Native mobile app that allows managing automations, controlling the house, performing voice interactions via LiveKit WebRTC, and managing multiple accounts. No extra apps for particular vendors are required;
4. MCP-based standard interface enabling compatible agents to access rooms, sensors, or devices. Agents including Claude, GPT-based solutions, or customized agents can receive permissions to execute tasks and manage devices. Permission levels differ by agent, room, and device;
5. Self-hosted security system based on mutual TLS with per-device X.509 certificates. Agent communication and authentication are performed using MQTT protocol. End-users' authentication uses OpenID Connect, while agents get scoped capability tokens. No proprietary cloud services are required.

The remainder of this paper is organized as follows. First, Section 2 provides an overview of related work. Then, Section 3 briefly describes the platform architecture. Section 4 discusses implementation details. Section 5 illustrates a practical example of a system deployment. Evaluation results are shown in Section 6. Limitations and security measures are considered in Section 7. Possible future developments are described in Section 8. Finally, conclusions are presented in Section 9.

II. RELATED WORK

2.1 Commercial platforms and the Matter standard

Smart home ecosystems in the context of the commercial market currently represent a wide variety of solutions [2]. Alexa, Google Home, HomeKit, SmartThings, Hue, and Lutron all work under their specific stacks, each employing different protocols, apps, and cloud backends. Integrations via Amazon Alexa or Google Home help to avoid fragmentation, but these are still closed systems, where withdrawing support leads to the failure of the respective devices. Moreover, privacy issues arise in relation to the storage and processing of voice data and usage analytics by these platforms.

Matter is a recent initiative by the Connectivity Standards Alliance introduced in 2022 [9]. It represents a standard ecosystem based on Thread and Wi-Fi. As reported by Alrashdi et al., Matter improves interoperability, at least in laboratory conditions [10]. However, it mainly focuses on communication instead of an entire ecosystem. Specific vendor applications are needed to control particular devices; for example, Philips Hue needs the application provided by the vendor for accessing all the capabilities of devices, and it is common practice among other companies as well. Moreover, depending on the update of the firmware, compatibility with the matter protocol might get worse.

According to Noura et al., there are three types of issues regarding interoperability – protocol mismatch, semantic gap, and vendor lock-in [1]. These problems become more evident when working with various devices based on the Zigbee protocol (lights), Wi-Fi (plugs), and Bluetooth technologies (sensors). Though Matter improves interoperability

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

somewhat, issues related to the setup inconsistency, application diversity of vendors, and migration from outdated equipment persist [10, 11].

2.2 Open-source alternatives

Home Assistant [4] is one of the most popular open-source projects. It is written in Python and supports a wide range of devices, offering GUI for creating automations. This solution also benefits from an active community. However, it has a steep learning curve. Coordinators need to be set up, YAML configurations have to be edited, mDNS has to be checked and fixed if necessary, and all this takes place within the environment of Linux. This is referred to as the "tinkerer's ceiling" by Ibrahim [39]: The system is perfect for experienced users who are comfortable working with hardware but not necessarily for everyone else.

ESPHome [5] usually goes hand-in-hand with Home Assistant. It allows creating YAML configurations for flashing ESP32 chips. For the people acquainted with the hardware terminology, it is easier than creating the firmware manually. Nevertheless, configuring, flashing, and maintaining each ESP32-based device separately is tedious. Node-RED [13] has a GUI for building automations using drag-and-drop flows. It does make coding unnecessary, but the understanding of MQTT messaging, topic structure, and asynchronous processing is still required. The interface makes using this tool more user-friendly but does not essentially decrease the complexity.

Sol has several distinguishing features compared to mentioned platforms: it has no YAML configuration, it allows flashing ESP32 chips from the browser, and it introduces an MCP server for controlling IoT devices using AI agents, something which other platforms lack.

2.3 IoT cloud platforms

There are several cloud-based IoT platforms popularly employed for prototype development. Arduino Cloud [14] offers a web-based IDE and dashboard creation services alongside code generators. Blynk [15] provides drag-and-drop interface for mobile application development, which works both for iOS and Android devices. ThingSpeak [17] focuses on time-series processing and integration with MATLAB. Although these platforms provide fast development, there are some drawbacks. First of all, for free users Arduino Cloud is limited to five variables and five devices, and Blynk is constrained by the number of streams. Secondly, both platforms completely rely on cloud-based infrastructure, making them vulnerable to potential interruptions.

In comparison of Blynk and ThingSpeak in the context of smart lighting solutions, Fachri et al. [16] conclude that Blynk outperforms due to the use of WebSockets, which allow avoiding polling, thus increasing the performance. Under the adverse condition of poor network coverage, however, performance of both platforms deteriorates. More generally, the research conducted by Syed et al. [18] concludes that Blynk, Arduino Cloud, and ThingSpeak are applicable to prototyping and education tasks but are too restrictive in other contexts.

Table 1. Feature comparison of home automation approaches.

Platform	Open src	Self-host	No CLI	No YAML	Modular HW	MCP	OTA	One app
Home Asst.	Yes	Yes	No	No	No	No	Via ESP	Yes
ESPHome	Yes	Yes	No	No	Partial	No	Yes	No
Arduino Cld	No	No	Partial	Yes	No	No	Yes	Yes
Blynk	No	No	Yes	Yes	No	No	Yes	Yes
ThingSpeak	No	No	Yes	Yes	No	No	No	Yes
Google Home	No	No	Yes	Yes	No	No	N/A	Yes
Matter	Partial	Yes	Yes	Yes	No	No	N/A	No
Sol (ours)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.4 Hardware sustainability and the right to repair

Under the influence of the EU Ecodesign Directive and the broader right-to-repair movement, sustainability concerns

© 2026 IJMRSET | Volume 9, Issue 4, April 2026| DOI:10.15680/IJMRSET.2026.0904435

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

have led to increased attention to the lifecycle costs of consumer electronic products [3]. In modern devices, such as smart plugs, the relay, microcontroller, and wireless communication module are put on one board inside an enclosure, and the failure of any element typically requires discarding the entire device. However, work on circular economy models suggests that devices should be modular, making it easier to recycle or reuse individual parts [3].

Sol adopts this approach in a straightforward manner. The relay module, sensor boards, microphone, speaker, and ESP32-S3 are all distinct components connected using standardized headers. In case of a malfunctioning relay, users can replace only it rather than discarding the entire board with the microcontroller onboard. Replacing any sensor is possible without affecting the entire structure. Overall, this leads to reduced device waste and extended utility of parts in operation.

2.5 LLM agents and physical environments

Some recent research efforts focus on connecting LLMs to IoT infrastructure and smart homes. Sasha [19] considered goal-oriented commands like "make it cozy" and concluded that even though generated LLM plans are sometimes valuable, they require grounding to avoid fragility. A structured plan-based execution was proposed in [20], where a sequence of actions, status check, and clarification loop were outlined, resulting in improved performance compared to baseline LLM agents in a 50-task benchmark. Another effort, IoTGPT [21], considers the possibility of breaking down users' requests into reusable subtasks, thus reducing redundancy and improving personalization.

Another area of interest is prompting and orchestrating the operation of generative IoT systems. It has been shown that reliability relies less on unstructured outputs from the system than on its structured toolset. A limitation that most systems exhibit is having a unique API, which limits further integration to the scope of that project.

Sol takes a completely different path in this regard. Instead of introducing yet another proprietary API, Sol leverages the Model Context Protocol [8], which is natively supported by many large language models, including Claude, GPT-powered agents, and Gemini. Sol extends the capabilities of MCP to include control over devices, i.e., discovery of available tools (e.g., toggle_relay, read_sensor, list_rooms) and usage thereof without any extra integration. To our best knowledge, this kind of feature is unavailable in any other home automation software.

III. SYSTEM ARCHITECTURE

The architecture of the Sol system consists of four layers – hardware, backend, application, and agentic ecosystem. Their interaction is depicted in Figure 1. The boundaries between layers are determined via specific protocols, thus allowing changing, scaling or replacing each layer independently from others.

3.1 Hardware layer

On the hardware side, the Sol architecture relies on ESP32-S3 based development board meant for installation in the wall switchboard enclosure instead of standard mechanical switches. The device comprises of four major modules.

The first of them is the relay module containing eight optically isolated relays up to 10A. There are three GPIO pins designated for internal purposes – status LED pin, reset button pin, and the I2S audio. This leaves only five relay channels available, however, more experienced users are able to change the designation of pins if necessary. INMP441 MEMS microphone is used as audio input and MAX98357A is used as an amplifier for driving small speaker, which allows voice outputs. Remaining GPIO pins are available for connecting sensors and actuators, which users are able to designate through the application software choosing any available GPIO pin.

The device is also designed to be modular in its physical design – each module is situated on its own breakout board and connected through standard 2.54 mm headers. This means that any faulty relay board is easy to disconnect from the system and replace; as well as the fact that when new module is required to connect to the system, say, CO2 detection one, it is possible just to plug it in any available GPIO header.

Table 2. Sol hardware node specifications.

Component	Specification
Microcontroller	ESP32-S3, dual-core Xtensa LX7, 240 MHz, 512 KB SRAM, 8 MB flash
Connectivity	Wi-Fi 802.11 b/g/n, Bluetooth Low Energy 5.0

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Relay module	8-channel, 10A/channel, optocoupler isolated, individual LED indicators
Microphone	INMP441 I2S MEMS, omnidirectional, -26 dBFS sensitivity
Speaker	MAX98357A I2S amplifier, 3W output, 4-ohm driver
GPIO available	Up to 21 pins after system reservations
Security	Per-device X.509 client certificate, TLS 1.3, secure boot
Firmware update	OTA via HTTPS with SHA-256 signed manifests, rollback protection
Power	5V DC, 2A, integrated AC-DC converter for switchboard installation
Enclosure	DIN-rail mountable, fits standard Indian/EU switchboard cavities

Each node is supposed to connect to VerneMQ MQTT broker cluster [23] using MQTTS (MQTT over TLS 1.3). When installing the device, there is generated an individual X.509 client certificate for this node. At connection initiation, both broker and device prove their identity presenting their respective certificates. Thus, mutual authentication takes place, where broker identifies device and vice versa [24]. Commands and telemetry use Model Context Protocol tool invocation format, hence no message transformation is required on the part of the agentic layer.

3.2 Control-plane backend: Traefik, Go services, Redis, and data

The control-plane backend implementation is a Go service located between the frontend application and the hardware layer. The main reason behind selecting Go was its goroutine- based concurrency model, which made it possible to run numerous parallel WebSocket and MQTT connections without using expensive OS-level threads.

3.2.1 API and data persistence

The implemented RESTful API consists of functions for registering devices, managing homes and rooms, inviting new users, assigning relays to homes, configuring sensors, and creating automation rules.

All data is stored in the PostgreSQL database, where the PgBouncer [25] connection pooling tool manages connections to the database. Telemetry and actuation logs are kept in TimescaleDB hypertables [41]. Semantic searches can be performed using the pgvector tool [32], running on the same PostgreSQL cluster. There are about 200 API calls per day from 34 devices using six server instances. PgBouncer makes sure that the total number of connections remains stable despite multiple concurrent connections initiated from the application by different users.

3.2.2 Real-time communication

Every time when a user acts on some device using the mobile app, such a command is sent to the device through the backend using the MQTT protocol and back for the confirmation. For such a confirmation to reach all clients immediately, real-time communication is needed, since otherwise, the UI becomes unresponsive.

For that, Sol uses horizontally scalable WebSocket hub that tracks connected clients independently for each hub instance. Intra-hub synchronization is implemented with Redis Pub/Sub [26]. With a pub/sub channel, each hub is able to publish an event that other hubs duplicate to their clients. To scale, it is only necessary to add more hubs that can be distributed by Traefik [42].

3.2.3 MQTT bridge and firmware store

The backend opens a persistent connection with the VerneMQ cluster, subscribes to telemetry topics, and sends commands to devices using MQTT. All intra-service communications occur using gRPC and mTLS for securing connections and boosting speed since the traffic is encrypted but stays inside one LAN.

The firmware binaries are uploaded to MinIO [27] as an S3-compatible object storage. Go service prepares firmware images based on the templates using text/template engine [43] powered by sprig library [44]. Then, images are optionally configured for the target home and room, signed, and put into the object storage. Upon availability of new firmware versions, nodes are informed and the update takes place using OTA updates.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

3.2.4 Authentication

To enable authentication in Sol, we used OpenID Connect [28] protocol, with Keycloak [29] as the identity provider. Both web and mobile applications authenticate themselves with OAuth2 Authorization Code flow with PKCE extension, while provisioning of ESP32 nodes is realized by OAuth2 Device Code flow, where users accept the device through the other device, e.g., mobile application. This way, nodes obtain access tokens.

Access tokens contain user id, user's homes, and user's roles in homes. Since Keycloak is maintained by Red Hat, Bosch, and other companies and is CNCF incubating project, it fits perfectly Sol's scenario with separate domain for each home.

3.3 Application layer

Both kinds of clients employ the same backend, which consists of Next.js web dashboard and React Native mobile application, offering all above-mentioned functionality.

3.3.1 Home, room, and device management

Provisioning of smart home implies room provisioning with the ESP32 nodes inside, alongside with configuring channels of relays. Namely, when channel 3 and its description such as "Bedroom ceiling fan" are configured, everything becomes ready. Sensor configuration can be equally simple as well. Namely, users need to choose GPIO pin 17 and the type of sensor "DHT22 temperature/humidity sensor". Nevertheless, in case the chosen pin is used by any other device, it is no longer available. Any changes to YAML, JSON or any other configuration files is not required. The procedure of configuring up to four devices and one sensor in one room takes roughly three minutes.

3.3.2 Collaborative access

The users might invite others to get access rights in terms of the four mentioned roles which includes owner (granting access, deleting smart home), administrator (managing devices, automations, but no deleting of the smart home), member (managing devices, seeing sensors), guest (managing devices in the certain room). The procedure of invitation implies generating a special access code that will be included into the unique hyperlink. For instance, the landlord might invite the tenant to become a member in order to let the former invite yet another user to manage living room lighting.

3.3.3 Voice assistant

Voice communication is conducted by the means of LiveKit [30], which stands for open- source library utilizing WebRTC technology. Namely, audio streaming from the phones' microphones and INMP441 microphones attached to ESP32 nodes is sent to the AI services by the means of LiveKit media pipeline. Subsequently, the AI turns received audio stream data into text and sends them to the augmented LLM. Finally, the system provides a response in voice format implying text-to-speech transformation. The mean latency measured as the duration between saying something and receiving LLM response amounts to 1.2 seconds, which takes about 800 ms due to LLM processing time. That is, the pipeline of voice communication needs nothing more than LiveKit.

3.3.4 Importance of a single application

That is important nowadays in case if for managing several devices users have to employ several applications, that is, in case the smart home uses Philips Hue lights, Nest thermostat, Ring doorbell, and TP-Link smart plugs. In other words, in such cases users would have to work with four separate apps, four accounts, four notifications, four views of rooms organization. Moreover, four different applications may become four possible attack surfaces for hackers. Hence, Sol seeks to introduce a single application only.

3.4 Agentic ecosystem layer

Here, Sol deviates significantly from traditional home automation systems. In Sol, the house itself works as a dedicated MCP server.

3.4.1 MCP server

The AI service is implemented with the help of FastAPI [31] and provides the tools available in Sol via MCP API. The tools include functions such as toggle_relay, read_sensor, list_rooms, list_devices, get_device_state, create_automation, get_history and other operations. Each tool has its JSON schema that defines the accepted input and output parameters. Once the agent connects and is able to recognize the provided tool, it can start using the corresponding functionality.
It's done in a similar way to how the agents communicate with various services when using GitHub or Slack. The only difference is that Sol enables this kind of interaction with the physical hardware instead. For example, a user can allow

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

the agent such as Claude access to the smart home and perform operations such as reading the bedroom temperature, turning lights off and creating some automated behavior, e.g. turn on the exhaust fan if the humidity is above a certain level.

3.4.2 Inference pipeline

As in the previous layers, the commands written or spoken in natural language go through several stages. First, the guardrails are applied, including policy check, Azure Content Safety check to detect malicious commands.

Next, the embedded command is used to query indexed with pgvector tables in PostgreSQL [32]. The latter holds all structured data related to the smart home infrastructure, i.e., the room name, device labels, sensor locations, automation history and recently gathered context. As a result, the generated actions are based on reality rather than imagination of the AI, which makes the system avoid actions for non-existing devices.

The enriched prompt is sent to Azure OpenAI API that responds with one or more calls to the available MCP tools. The commands returned by the AI are executed on the backend side, and the results are sent back to the model that returns natural-language answer. In practice, the whole inference pipeline takes less than a second.

3.4.3 Permission model

Having an AI system managing physical devices carries significant safety risks; hence, access must be restricted. In Sol, the capability-based permission system [33, 34] is employed, where each agent has its token defining exactly what it can do with the home infrastructure.

There are three different types of permissions, including global (on per agent basis), room-level and device-level ones. Thus, for example, calendar-related agents can only read data from occupancy sensors, while bedtime automation agents could be allowed to interact only with devices present in a particular room.

Permissions management is done by users through the web app interface with easy-to- use options like check-boxes and dropdown menus. Moreover, there are no policies to manage – instead, if an agent attempts to perform any unauthorized actions, it will get a structured response explaining which permission it needs. Afterward, an agent can request new permissions that can be denied or granted by the user.

3.4.4 Third-party composition

MCP provides the means for both serving and consuming services. Sol can offer tools to third-party agents, but at the same time, it is able to call the tools made available by various other MCP-enabled services. This allows constructing workflows that involve multiple systems during a single execution.

For example, to execute something like "create reminder to switch off the AC once I leave for work", Sol can work with tools of services such as Google Calendar or Google Tasks. The agent creates a reminder and connects it to the automation created in Sol that checks the occupancy of the space.

IV. IMPLEMENTATION AND DEPLOYMENT DETAILS

4.1 WebSerial provisioning

The user connects his ESP32-S3-based node to a computer using USB cable and opens the Sol browser-based app in Chrome. From there he proceeds to "Add new device" button. Wizard leads further actions. WebSerial API [35] provides backend support for communicating via browser with the node's serial port. Sol utilizes web browser version of esptool [36] - the default utility for flashing Espressif SoCs.

It consists of the following steps: first comes the browser request to access the serial port, followed by bootloader handshake for identifying the chip type and size. After that, the firmware flashed into device is pre-configured with Wi-Fi SSID, password, and MQTT broker IP address. Lastly, when the device reboots, it connects to Wi-Fi and starts OIDC Device Code flow which authorizes the device explicitly in the mobile app and registers it in the platform. It takes less than a minute and a half.

In comparison, ESPHome requires Python and pip installation, YAML config file preparation and flashing; likewise,

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Arduino Cloud needs IDE setup, activation of the target board and running the Create Agent procedure. These are all reduced by Sol to minimum in- browser configuration.

Once the firmware is flashed to the device, further updates are applied automatically. Every hour, the device requests the new firmware release from the update server. It is pulled from the MinIO object storage via HTTPS; the new release is signed with the previously flashed certificate and validated against SHA-256 checksum. If there is any error or the device fails to execute self-checking within 30 seconds after reset, then the rollback to the previous release is performed. Possible risks associated with OTA updates, such as replay attacks, partial updating and weak key management are reduced by signing the manifest and rollback functionality.

4.2 MQTT and device security

All devices use VerneMQ cluster [23] for their communications. The chosen MQTT broker was the one having clustering ability, predictable resource usage under high load conditions and flexible authentication using plugins interacting with Sol certificate authority.

Each device has a unique X.509 certificate issued to it during flashing. They provide mutual TLS 1.3 authentication and identification between VerneMQ and the device, which has device UUID embedded in the certificate CN field. Stealing client certificate results in only a single device being banned from the VerneMQ cluster.

VerneMQ cluster access is limited to specific device namespaces. Each device gets unique path sol/home-id/room-id/device-id/.

Each device is able to publish to and receive from only its namespace. Sending messages to or from another device from the same home will be rejected by VerneMQ due to strict topic access policies.

Tests of VerneMQ cluster consisting of 3 brokers were able to handle 10,000 simultaneous connections while maintaining 99th percentile latency at 12 ms. This is far beyond what is required by the average smart home containing a dozen devices.

4.3 LiveKit voice pipeline

For audio transport, the agent relies on the LiveKit's Agents toolkit [30], which works on WebRTC. Audio processing pipeline involves three main stages: speech-to-text conversion (audio to text), LLM inference (text to MCP tools calls and text answer), and text-to-speech conversion (answer text to audio). LiveKit is responsible for all low-level details such as codecs negotiation, dynamic adaptation to current network bandwidth, echo cancellation, and network address translation using STUN/TURN services.

Typical average time to process a voice query and start replying after the user stopped speaking is about 1.2 seconds. 800 milliseconds out of those are taken by LLM inference stage, 400 milliseconds – speech-to-speech and text-to-speech conversions together. This is slower than the fully local version of the voice assistant, but is fast enough to sound natural.

4.4 MCP server implementation

The MCP server is implemented as part of the FastAPI-based AI service. It implements the server-side MCP protocol as specified [8] and makes all capabilities available in Sol as a list of callables.

Table 3. MCP tools exposed by Sol.

Tool	Parameters	Returns	Scope required
toggle_relay	room_id, device_id, channel, state	Device state object	device:write
read_sensor	room_id, device_id, sensor_type	Reading + timestamp	device:read
list_rooms	home_id	Array of rooms	home:read
list_devices	room_id	Array of devices	room:read

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

get_device_state	device_id	Full state snapshot	device:read
create_automation	trigger, conditions, actions	Automation ID	auto:write
get_history	device_id, range	Time-series data	device:read
delete_automation	automation_id	Confirmation	auto:write

Each tool execution request undergoes three validations before being processed. Firstly, it is checked if the provided OAuth 2.0 bearer token is valid. Secondly, the system checks if the desired action is permitted in the current scope of either home, room, or device. Thirdly, rate limiting is done in order to limit number of allowed requests in a specific time interval. If any of these conditions fails, an error message is returned in a standard format.

4.5 RAG grounding with PostgreSQL pgvector

The AI service stores the data related to user's home context within a PostgreSQL database via the pgvector library [32]. It stores room names, device names, sensor locations, automation information, and even the latest interaction history. After receiving the command, it creates the embedding of the command and requests similar entries from the database, adding them to the prompt that is sent to the model.

It should be noted how this procedure operates depending on the specifics of the command entered by the user. As regards the phrase "turn off the light," we may assume that there is enough context in order for the model to recognize the user's intention. The client identifies the user's location, the device catalogue is extracted from the index, and the name of the device is known. In other words, the model knows which particular device should be used and there is no need for additional information.

In this way, an additional security measure has been provided as any action is performed with only devices found in the database and, thus, in reality.

In addition, guardrails checks and Azure Content Safety procedures apply in both directions in regard to input and output. There are multiple reasons why such measures should be taken: in the first place, natural language input is transformed into a real action. Thus, bypassing previous instructions and turning on all devices at once becomes impossible.

V. USE CASE WALKTHROUGH

This next story goes along with what happens during a user's interaction from unboxing the device to communicating with voice assistants.

5.1 Setup: from box to working node

Priya buys a Sol node for her bedroom. She mounts the relay module on the plate and connects AC wiring to relay channels (or uses an electrician's help). She connects the ESP32- S3 board to the computer using the USB-C cable, which comes with the board.

She opens sol.local in Chrome on her laptop. The website detects the device and starts the setup wizard. Priya enters her Wi-Fi credentials and presses "Flash". The website uploads the firmware with the help of WebSerial protocol. After one and a half minutes the node restarts and connects to the WiFi network. A notification pops up on Priya's smartphone requesting her to approve the device and add it to her house. She clicks "Approve" button, selects "Bedroom" option from the list of available rooms and completes the process.

The interface shows eight relays. Priya maps channel 1 to "Ceiling light", channel 2 to "Bedside lamp", and channel 3 to "Fan". Then, she connects a DHT22 temperature sensor supplied by the manufacturer to the node GPIO connector. Then, she opens the dialog window, selects GPIO pin 4, selects "DHT22" in the list of available sensors, and names the sensor "Room temperature". It takes her four minutes to complete the process. Priya had to enter no text other than labels.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

5.2 Daily use: app and voice

Priya controls the state of the bedroom node through the Sol app. There are three toggle switches corresponding to three relay channels; current room temperature is shown on the dashboard. Priya taps on a microphone icon and says: "Turn off the fan". The audio is sent to AI via LiveKit, processed and recognized at the AI server side, the corresponding action is detected and executed in real-time with the help of RAG pipeline. Meanwhile, the state of the room is updated in the dashboard through the WebSockets protocol.

Priya sets up the automation: "If the temperature becomes more than 28C, then turn on the fan". In order to create an automation, she opens a simple creation form, fills it with data: "Trigger – sensor reading", "Condition – temperature higher than 28C", "Action – Toggle relay 3 on". There are no MQTT topics and YAML involved in this process.

5.3 Agentic interaction

Priya uses Claude as her regular AI assistant. During MCP handshake, she authorizes him to have read access to all sensors in her home and write access to bedroom devices. Claude finds the Sol ecosystem while connecting to the MCP server.

During a conversation about her morning routine, she mentions: "I keep forgetting to turn off the bedroom light when going to work. Could you please do something about it?". Claude asks her to list the devices in her bedroom. The list arrives: he sees there is a ceiling light on channel 1. Now, he calls create_automation method with trigger defined as the occupancy sensor reading after 5 minutes. "Sure thing. Done. Your bedroom light will be turned off five minutes after you leave the room". If Claude tried to turn off the kitchen light as well, Priya's authorization would not allow that.

VI. EVALUATION

In total, Sol was distributed across 6 rooms within one household, using 6 ESP32-S3 chips controlling 34 smart devices (light fixtures, fans, switches), and 4 more chips used as sensors (2 × DHT22, 1 × BH1750 light sensor, 1 × PIR motion sensor). This setup was running continuously for 12 weeks.

6.1 Latency

Table 4. End-to-end latency measurements across 10,000 operations.

Operation	Median (ms)	P95 (ms)	P99 (ms)
Direct relay toggle (MQTT)	28	38	45
Sensor read (MQTT round-trip)	31	42	51
NL command, 1 user	820	1100	1350
NL command, 50 concurrent	2100	3200	4100
Voice pipeline end-to-end	1200	1800	2300
OTA firmware push (1 MB)	4200	5100	6800
WebSocket state broadcast	8	15	22
MCP tool call (pre-authed)	35	52	68

Direct MQTT interactions have very low latency — 28 ms median delay for switching off relay. This is the case for any user command issued manually through the application, and for automation that doesn't use language model at all. Inference with the language model takes an additional 800 ms. With high number of concurrent chat sessions (50), maximum 95th percentile latency for processing commands reaches 3.2 seconds, due to waiting on the inference API response alone. WebSocket broadcast to all connected clients after state change has a median delay of just 8 ms. Fast enough for real-time UI.

6.2 Reliability

During 12 weeks of operation, the application has sent 47,832 commands for switching on/off relays to MQTT broker. No lost messages were observed in our log comparison against acknowledgment received from the broker. VerneMQ cluster consisting of 3 Erlang nodes experienced 1 planned downtime caused by OS update, recovering itself within less than 4 seconds. After the cluster failure, all ESP32 nodes automatically re-connected back in average of 1.8 seconds.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

The OTA process was done 3 times during the experiment, with total time of up to 15 minutes required for updating all 6 devices with the latest firmware.

6.3 Provisioning time

We tracked the duration starting from plugging in a node via USB to full provisioning (ability to execute commands sent from MQTT and report sensor readings) of 6 nodes.fully operational node (relay responding to commands, sensor reporting data) for 6 nodes.

Table 5. Provisioning time per node.

Node	Flash (s)	Network join (s)	Registration (s)	Total (s)
Bedroom	62	8	12	82
Living room	58	7	11	76
Kitchen	65	9	14	88
Bathroom	60	8	13	81
Study	61	7	12	80
Hallway	63	8	11	82

Provisioning took 82 seconds on average. Major contributor here is flashing time, which takes about 60 seconds and consists of uploading 1.2MB firmware via USB connection. The remaining 20 seconds account for connecting the node to local network and OIDC authorization code registration in Sol backend. Compare it with ESPHome – YAML file creation can take several minutes even for inexperienced users, flashing and compiling add another 30-60 seconds.

6.4 MCP agent interaction

To measure MCP agent interaction we used Claude and connected it to Sol API. We generated 200 natural-language commands, containing device controls, data retrieval requests, and automation creation. The agent succeeded in execution of 187 commands (93.5%). In 13 cases of failed command executions, 8 of them had ambiguous formulation, prompting the user for clarification instead of guessing; 3 of them timed out due to fast requests triggering rate limits; 2 of them had incorrect permissions to operate the room outside its scope.

VII. DISCUSSION

7.1 Limitations

WebSerial is currently only available on Chromium-based web browsers, i.e., Chrome and Edge [35]. Since both browsers hold the market share exceeding 65% among desktop users, most people should not face difficulties in using the application; however, those who use Firefox and Safari have an option as well. In particular, a command-line utility for flashing is offered, although it decreases the zero-install feature slightly.

The next limitation concerns the use of Azure OpenAI services for performing natural language processing. While all other parts can be run locally, an additional request has to be made to the external service for each text or voice input. Overall, the system is flexible enough to allow configuring communication with any compatible OpenAI endpoint, including local installation, which has not been tested for performance yet.

Finally, the experiment has been conducted in a domestic scenario with six nodes. The apartment building, co-living community, or office scenario implies other constraints, which include networking separation between users, centralized device management, coordination of common policies on home automation, and regulatory compliance, particularly with regard to fire safety and accessibility requirements. None of these cases has been investigated so far.

As for the hardware part, it is possible to note that the selection of the ESP32-S3 microcontroller enables the usage of relatively powerful 240 MHz dual-core CPU with 512 KB of built-in SRAM. Although it is sufficient for operating relays, measuring sensors, and communicating over the MQTT protocol, not many opportunities remain for performing other operations. Even some basic functionality, such as wake-word detection, requires significant computational

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

resources. The next generation of ESP chips, namely ESP32-P4, can prove more suitable in terms of performance, though it has not been taken into account yet.

7.2 Security analysis

Sol features a multi-layered security architecture. Specifically, the hardware level provides mutual TLS authentication with the help of device-specific X.509 certificates. It guarantees that an unauthorized device cannot join the network and minimizes the possibility of MITM attacks. The user level uses OIDC with PKCE to protect the authentication phase. Finally, the capability tokens on the agent level cannot be misused because they are restricted by default, and it is possible to revoke any token instantly via ZITADEL.

After implementing the MCP server component, one more kind of threat emerges: abuse of capabilities assigned to agents. Namely, a compromised agent may try to gather information about its environment within its permission scope, e.g., by collecting data from different sensors in an effort to determine the presence of occupants. To address this issue, the logging of all MCP tool invocations with corresponding timestamps, agent identity, parameters, and output values is implemented. Additionally, a certain degree of rate limiting and anomaly detection, which involves identifying fast consecutive calls to multiple sensors, are applied to prevent misuse. However, this approach does not completely eliminate the potential risk. Proving formally that permissions cannot be abused in any scenario requires further investigationAccording to the report [8], there exist three kinds of specific threats related to the MCP. They include prompt injection, permission escalation, and tool impersonation. The first problem has been tackled with the help of content filtering, whereas the second one has been addressed by blocking schema changes in the manifest of the server. This solution makes it impossible to implement fake tools. As for permission escalation, it can never occur without explicit user consent in the app itself.

7.3 Scalability

The stateless nature of the Go backend makes it trivially scalable just by launching additional instances behind the load balancer. WebSocket hubs rely on the Redis Pub/Sub pattern; therefore, the way to scale is the same. The clustering of VerneMQ occurs out of the box. One can use either an upgraded node or a read replica for scaling PostgreSQL. The Pgvector relies on the sharding by home ID and time range; TimescaleDB applies continuous aggregations for efficient telemetry data querying.

However, the problem with scalability starts with LLM inference. During the stress test, when up to 50 chat sessions were held concurrently, the 95th percentile of the request latency became around 3.2 seconds, and at that time, most requests were queued up in front of the Azure OpenAI API endpoint. Other tasks without LLM invocation showed totally different performance. The queues size of relay toggling requests, getting sensor values, and performing automated rules were up to 500 concurrently with P99 latency lower than 100 milliseconds. In practice, it is highly unlikely to have multiple voice or text chats at once since most requests to the API are generated by the automation scripts and sensor readings by the service, and they do not need the LLM.

7.4 Environmental impact

Classic smart bulbs belong to the category of monoliths – all the components starting from MCU, wireless transceiver, LED driver to LEDs reside within the plastic casing. Once the protocol support is no longer relevant, the product should be thrown away and replaced with new ones. In Sol's case, the control component and the appliance can be separated; hence, the lamp is a lamp, fan is a fan, and heater is a heater regardless of being controlled by relays. Consequently, the whole system is upgradeable since it is only required to update the ESP32 firmware once the communications change. Additionally, in case of the relay failure, the replacement of such parts can occur without changing anything else.

As shown in the tested configuration, the usage of six Sol nodes allows us to cut off up to 34 smart plugs and switches. As a result, there will be lesser amounts of electronics waste produced since the radios, MCUs, and plastic casings do not need to be manufactured anymore. While a complete life cycle assessment is outside the scope of the current work, it is clear that the concept of recycling reusable components works well.According to the latest report, the global electronic waste production is expected to reach 82 million tonnes by 2030, UN Global E-Waste Monitor (2024). At least some part of it will be contributed by IoT-based devices because of the shorter lifetimes and difficulties in repairing them. Nevertheless, the proposed concept introduces a new perspective on such environmental issues.

© 2026 IJMRSET | Volume 9, Issue 4, April 2026| DOI:10.15680/IJMRSET.2026.0904435

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VIII. FUTURE WORK

Local LLM inference. Inferring with a quantized version of LLaMA or Mistral on the host computer would remove the reliance on Azure OpenAI services. A comparison of local inference performance with models from Ollama versus current cloud inference performance would be performed in terms of latency, accuracy, and costs.

Matter bridge. The addition of a Matter bridge would enable controlling of Matter- enabled appliances together with Sol appliances, extending hardware compatibility with no custom firmware required.

Permission verification. To ensure the safety of the capability-based token model, formal verification techniques such as SPIN or Alloy are needed. It needs to be ensured that any sequence of operations executed by the MCP cannot lead to privilege escalation.

Usability study. A study involving 30 subjects of varied expertise levels, including non- technical users, hobbyists, and professionals, will be conducted. Sol will be evaluated compared with other software alternatives like ESPHome and Home Assistant with respect to setup time, error rates, and user satisfaction.

Per-appliance energy monitoring. The inclusion of current sensor modules in all relay channels (e.g., ACS712 and SCT-013) will allow for tracking individual appliance power consumption. With LLM analysis capabilities, the following feedback can be generated: "Your water heater consumed 4.2 kWh last night, which was 30% more than usual. Should I change the scheduling?"

In-device inference. Small neural networks can be run on the ESP32-S3 thanks to support of vector instructions. By offloading wake-word recognition and voice activity detection tasks to the device, one can reduce latency and communication overhead during interactions.
Multi-agent coordination. Conflicts can happen in multi-agent control of the same home environment when several agents try to control the same appliance at the same time. Current solution is very basic and involves locking individual devices from being controlled from other nodes. Multi-agent coordination strategies based on auction-style negotiations can be introduced.

Offline mode. Current system operates under the assumption of constant availability of network connection to backend services. Offline fallback mode that enables cached rule execution on the local ESP32 node in the absence of a connection to the backend can be added.

IX. CONCLUSION

In this paper, we presented Sol as an open-source smart home platform based on the modularity of ESP32-S3 hardware, a web-based configuration, single app UI, and Model Context Protocol server hosting AI agents. Our research aims to address the challenges that currently plague many home automation solutions such as fragmentation, waste, poor accessibility to end-users who may not have technical skills, complicated configurations, and the absence of a standard communication mechanism for the interaction between AI applications and the real world.

Our platform supports self-hosting and uses security measures like mutual TLS, OIDC, and capability- based tokens. Throughout the 12-week experiment, our platform operated reliably with no message loss during 47,832 relay events. The average round-trip latencies were found to be 28 ms for control operations and about 820 ms for natural language processing commands. Average provisioning time per device amounted to 82 seconds.

One of the most important aspects of our solution is the concept of the home as an environment rather than a set of devices associated with a particular application. Unlike traditional applications that work with environments through their APIs, our system sees them the same way as software components and lets the user interact with the actual hardware through the same interface as the software applications use. According to the results of our tests, this approach works, evidenced by the success rate of 93.5% from 200 commands issued by AI agents. The ultimate success will depend on further development of the MCP ecosystem.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

REFERENCES

[1] M. Noura, M. Atiquzzaman, M. Gaedke, Interoperability in Internet of Things: Taxonomies and open challenges. Mobile Netw. Appl. 24, 796-809 (2019). https://doi.org/10.1007/s11036-018-1089-9
[2] R. Minerva, A. Biru, D. Rotondi, Towards a definition of the Internet of Things (IoT). IEEE Internet Initiative 1, 1-86 (2015)
[3] C.P. Balde, R. Kuehr, K. Yamamoto et al., The Global E-waste Monitor 2024 (United Nations Institute for Training and Research, Geneva, 2024). https://ewastemonitor.info/the-global-e-waste- monitor-2024/
[4] Home Assistant, Home Assistant: Open source home automation. https://www.home-assistant.io/ (accessed April 2026)
[5] ESPHome, ESPHome: Smart Home Made Simple. https://esphome.io/ (accessed April 2026)
[6] A.J. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, C. Dixon, Home automation in the wild: Challenges and opportunities, in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11) (ACM, New York, 2011), pp. 2115-2124. https://doi.org/10.1145/1978942.1979249
[7] Y. Wang, H. Zhao, T. Wang et al., A survey on large language model based autonomous agents. Front. Comput. Sci. 18, 186345 (2024). https://doi.org/10.1007/s11704-024-40231-1
[8] Anthropic, Model Context Protocol specification (2024). https://modelcontextprotocol.io/specification/2025-11-25
[9] Connectivity Standards Alliance, Matter: The foundation for connected things. https://csa- iot.org/all-solutions/matter/ (accessed April 2026)
[10] E. Alrashdi, A. Alqazzaz, R. Alharthi, Matter: IoT interoperability for smart homes (2024). https://arxiv.org/abs/2405.01618
[11] A. Mota, C. Serodio, A. Valente, Matter Protocol Integration Using Espressif's Solutions to Achieve Smart Home Interoperability. Electronics 13(11), 2217 (2024). https://doi.org/10.3390/electronics13112217
[12] G. Desolda, C. Ardito, M. Matera, Connecting home: Human-centric setup automation in the augmented smart home, in Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (ACM, New York, 2024). https://doi.org/10.1145/3613904.3642862
[13] Node-RED, Low-code programming for event-driven applications. https://nodered.org/ (accessed April 2026)
[14] Arduino, Arduino Cloud. https://cloud.arduino.cc/ (accessed April 2026)
[15] Blynk, Blynk IoT platform. https://blynk.io/ (accessed April 2026)
[16] M. Fachri, A. Nugroho, N. Akhmadi, Comparative study of Internet of Things (IoT) platform for smart home lighting control using NodeMCU with ThingSpeak and Blynk web applications. Fidelity J. 5(1), 42-50 (2023)
[17] MathWorks, ThingSpeak: IoT analytics platform. https://thingspeak.com/ (accessed April 2026)
[18] I.C. Panagou, S. Katsoulis, E. Nannos, F. Zantalis, G. Koulouras, A Comprehensive Evaluation of IoT Cloud Platforms: A Feature-Driven Review with a Decision-Making Tool. Sensors 25(16), 5124 (2025). https://doi.org/10.3390/s25165124
[19] E. King, H. Yu, S. Lee, C. Julien, Sasha: Creative Goal-Oriented Reasoning in Smart Homes with Large Language Models. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 8(1), 1-35 (2024). https://doi.org/10.1145/3643505
[20] D. Rivkin, F. Hogan, A. Feriani et al., SAGE: Smart home Agent with Grounded Execution (2024). https://arxiv.org/abs/2311.00772
[21] G. Theodorou, A. Kanavos, P. Pintelas, Leveraging retrieval-augmented generation for automated smart home orchestration. Future Internet 17(5), 198 (2025). https://doi.org/10.3390/fi17050198
[22] B. Xiao, B. Kantarci, J. Kang, D. Niyato, M. Guizani, Efficient Prompting for LLM-Based Generative Internet of Things. IEEE Internet Things J. (2024). https://doi.org/10.1109/JIOT.2024.3470210
[23] VerneMQ, A high-performance, distributed MQTT message broker. https://vernemq.com/ (accessed April 2026)
[24] HiveMQ, X509 client certificate authentication: MQTT security fundamentals. https://www.hivemq.com/blog/mqtt-security-fundamentals-x509-client-certificate-authentication/ (accessed April 2026)
[25] PgBouncer, Lightweight connection pooler for PostgreSQL. https://www.pgbouncer.org/ (accessed April 2026)
[26] Redis, Redis Pub/Sub. https://redis.io/docs/manual/pubsub/ (accessed April 2026)
[27] MinIO, MinIO high-performance object storage. https://min.io/ (accessed April 2026)
[28] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, C. Mortimore, OpenID Connect Core 1.0 (OpenID Foundation, 2014). https://openid.net/specs/openid-connect-core-1_0.html
[29] ZITADEL, Open source identity management. https://zitadel.com/ (accessed April 2026)
[30] LiveKit, Open source WebRTC infrastructure. https://livekit.io/ (accessed April 2026)

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

[31] S. Ramirez, FastAPI: Modern, fast web framework for building APIs with Python. https://fastapi.tiangolo.com/ (accessed April 2026)
[32] pgvector, Open-source vector similarity search for Postgres. https://github.com/pgvector/pgvector (accessed April 2026)
[33] S. Gusmeroli, S. Piccione, D. Rotondi, A capability-based security approach to manage access control in the Internet of Things. Math. Comput. Model. 58(5-6), 1189-1205 (2013). https://doi.org/10.1016/j.mcm.2013.02.006
[34] B. Xu, L.D. Xu, H. Cai, C. Xie, J. Hu, F. Bu, Ubiquitous data accessing method in IoT-based information system for emergency medical services. IEEE Trans. Ind. Inform. 10(2), 1578-1586 (2014). https://doi.org/10.1109/TII.2014.2306382
[35] W3C, Web Serial API. https://wicg.github.io/serial/ (accessed April 2026)
[36] Espressif Systems, esptool.py: ESP8266 and ESP32 serial bootloader utility. https://github.com/espressif/esptool (accessed April 2026)
[37] A. Alrashidi, A. Alqazzaz, Secure firmware over-the-air updates for IoT: Survey, challenges, and discussions. Internet Things 18, 100508 (2022). https://doi.org/10.1016/j.iot.2022.100508
[38] N. Dragoni, S. Giallorenzo, A.L. Lafuente et al., Microservices: Yesterday, today, and tomorrow, in Present and Ulterior Software Engineering (Springer, Cham, 2017), pp. 195-216. https://doi.org/10.1007/978-3-319-67425-4_12
[39] A.A.Z. Ibrahim, Taxonomy of home automation: Expert perspectives on the future of smarter homes. Inf. Syst. Front. 26, 2133-2159 (2024). https://doi.org/10.1007/s10796-024-10496-9
[40] Anthropic, Donating the Model Context Protocol and establishing the Agentic AI Foundation (2025). https://www.anthropic.com/news/donating-the-model-context-protocol-and-establishing-of- the-agentic-ai-foundation
[41] Timescale, TimescaleDB documentation. https://docs.timescale.com/ (accessed April 2026)
[42] Traefik Labs, Traefik Proxy documentation. https://doc.traefik.io/traefik/ (accessed April 2026)
[43] Go Team, text/template package documentation. https://pkg.go.dev/text/template (accessed April 2026)
[44] Masterminds, sprig template functions for Go templates. https://github.com/Masterminds/sprig (accessed April 2026)

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com