LLM-Driven Auto Configuration for Transient IoT Device Collaboration

Hetvi Shastri[1], Walid A. Hanafy[1], Li Wu[1], David Irwin[1], Mani Srivastava[2†], Prashant Shenoy[1]
[1]University of Massachusetts Amherst, USA
[2]University of California Los Angeles, USA

Abstract

Today's Internet of Things (IoT) has evolved from simple sensing and actuation devices to those with embedded processing and intelligent services, enabling rich collaborations between users and their devices. However, enabling such collaboration becomes challenging when transient devices need to interact with host devices in temporarily visited environments. In such cases, fine-grained access control policies are necessary to ensure secure interactions; however, manually implementing them is often impractical for non-expert users. Moreover, at run-time, the system must automatically configure the devices and enforce such fine-grained access control rules. Additionally, the system must address the heterogeneity of devices.

In this paper, we present `CollabIoT`, a system that enables secure and seamless device collaboration in transient IoT environments. `CollabIoT` employs a Large language Model (LLM)-driven approach to convert users' high-level intents to fine-grained access control policies. To support secure and seamless device collaboration, `CollabIoT` adopts capability-based access control for authorization and uses lightweight proxies for policy enforcement, providing hardware-independent abstractions.

We implement a prototype of `CollabIoT`'s policy generation and auto configuration pipelines and evaluate its efficacy on an IoT testbed and in large-scale emulated environments. We show that our LLM-based policy generation pipeline is able to generate functional and correct policies with 100% accuracy. At runtime, our evaluation shows that our system configures new devices in ~150 ms, and our proxy-based data plane incurs network overheads of up to 2 ms and access control overheads up to 0.3 ms.

SEC '25, Arlington, VA, USA

ACM ISBN 979-8-4007-2238-7/2025/12
https://doi.org/10.1145/3769102.3770619

CCS Concepts

• **Security and privacy** → *Access control*; • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**.

Keywords

IoT, LLMs, Access Control, Device Collaboration

ACM Reference Format:
Hetvi Shastri[1], Walid A. Hanafy[1], Li Wu[1], David Irwin[1], Mani Srivastava[2†], Prashant Shenoy[1]. 2025. LLM-Driven Auto Configuration for Transient IoT Device Collaboration. In *The Tenth ACM/IEEE Symposium on Edge Computing (SEC '25), December 3–6, 2025, Arlington, VA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3769102.3770619

1 Introduction

Recent technological advances have led to rapid growth in Internet of Things (IoT) devices in varied domains, such as smart homes, mobile health, and entertainment [5, 22, 36, 52, 62, 64, 70, 72, 86]. Researchers estimate that the number of deployed IoT devices exceeded 18.8 billion in 2024 and is growing in a near-exponential manner [68]. Some have even predicted an Internet of Trillion Things where such devices become ubiquitous in our daily lives [18]. Alongside this growth, IoT devices have evolved from passive sensing and actuating devices to smart collaborating devices that interact with one another. For example, motion sensors may trigger smart lights or security cameras upon detecting motion [27]. To support such collaboration, devices need to interact with one another and share access to their data, connectivity, and resources with other devices and their users [39, 53]. In addition, newer IoT devices are often mobile or nomadic in nature, frequently moving into new environments beyond their home networks.

Device collaboration in such dynamic and ad hoc settings requires device owners to specify access control policies for visitors, while also requiring visiting devices to be configured to access network resources and IoT services in remote environments. For example, food delivery robots are now

[†]Author holds concurrent appointments as a Professor at UCLA, and as an Amazon Scholar. This paper describes work performed at UCLA and is not associated with Amazon.

common in many university campuses [13], and online retailers are increasingly experimenting with autonomous robots or drones to deliver packages [34]. In the future, such robots could collaborate with home IoT devices to securely unlock entry points—such as a garage door [83]—and place packages or food items inside the home. In this scenario, the homeowner needs to configure access control policies that specify which actions delivery robots are allowed or denied. In particular, access to home devices or services needs to be granted in a fine-grained manner such that only an appropriate subset of them becomes accessible to the delivery robots. For security or privacy reasons, such access may need to be limited in different ways—e.g., restricted to the time period of the visit or rate limited to prevent resource hogging or DDoS attacks [80]. When a delivery robot arrives, the IoT system must automatically configure its access rights and facilitate secure and real-time interaction with home devices. While policy specification and device configuration can be done manually, such an approach does not scale when the number and frequency of transient guest devices entering and leaving an environment increases significantly.

Enabling secure and seamless device collaboration in transient IoT environments remains challenging in today's edge environments for several reasons. First, device ecosystems grow increasingly complex and interconnected, *specifying* access policies and *configuring* devices to access resources based on these policies becomes highly challenging–especially in transient settings where fine-grained access control is a necessity [45, 65, 89]. However, manually configuring of policies and devices is both tedious and impractical for non-expert users. One approach is to use AI-driven auto configuration to automate these tasks. For example, we can leverage the code generation capabilities of large language models (LLMs) to convert a natural language description of policy intent into structured policies. Such policies can then drive the auto configuration of transient guest devices to provide the desired levels of resource access. However, the generative nature of LLMs introduces several limitations for our setting. First, their outputs are probabilistic and may contain hallucinated or logically inconsistent information, which poses risks in policy enforcement. Second, LLMs are typically trained on broad, general-purpose datasets and lack exposure to domain-specific corpora, which hinders their ability to generate system-specific access policies. As a result, relying solely on LLMs is insufficient for automatic generation of fine-grain access control policies from high-level specification with the accuracy and reliability required in transient IoT environments.

A second challenge is that enabling seamless device collaboration at runtime presents significant hurdles due to the heterogeneity of devices in edge environments and the scale of possible interactions. IoT devices in edge environments such as buildings or homes are inherently heterogeneous. Devices vary widely in terms of hardware capabilities, operating systems, communication protocols, and runtime environments, and they often expose different vendor-specific APIs. For example, enabling a delivery robot to interact with a smart door lock of a home may require installing specific drivers or SDKs. This setup is not only time-consuming but also not scalable in transient IoT environments where devices arrive and leave frequently. Moreover, as the number of devices increases, the number of possible interactions grows rapidly. Supporting real-time collaboration at this scale requires systems that can dynamically discover devices, determine authorized interactions, and enforce policies efficiently, which remains a significant systems challenge.

To address these challenges, we present `CollabIoT`, a system that enables devices in transient IoT environments to securely collaborate by sharing data, resources, and services in a seamless and secure manner. `CollabIoT` employs a large language model (LLM)-driven approach to generate fine-grained access control policies from high-level natural language descriptions provided by device owners. The LLMs produce structured YAML-based policies, which are then validated through a multi-stage pipeline to ensure syntactic correctness, semantic soundness, and alignment with the user's intent and contextual constraints. `CollabIoT` supports an automated policy-driven configuration pipeline based on capability-based access control tokens, which allows for secure and scalable access control enforcement. Furthermore, `CollabIoT` enables collaboration via lightweight proxies that provide policy enforcement and hardware-independent interfaces. Lastly, `CollabIoT` is designed to be lightweight and can be deployed on edge nodes, such as a home hub or gateway controller, enabling real-time and secure interactions with enhanced privacy.

In designing, implementing, and evaluating `CollabIoT`, our paper makes the following contributions:

(1) **LLM-based Policy Generation.** We present an LLM-based policy generation pipeline that generates validated access control policies from natural language descriptions. Our approach includes multi-stage validation to ensure correctness and security.
(2) **Device Auto Configuration.** Our `CollabIoT` system supports auto configuration of transient IoT devices to enable secure, seamless, and scalable device collaboration. Specifically, `CollabIoT` auto-configures devices with capability-based access tokens for authorization and utilizes lightweight proxies for policy enforcement, providing hardware-independent abstractions.
(3) **Implementation and Evaluation.** We implement `CollabIoT` using Pydantic for structured policy generation, with ~2.6k SLOC. We evaluate `CollabIoT` on a

real testbed and in large-scale emulations. Our results demonstrate the effectiveness of the LLM-based auto-configuration and granular access control in transient IoT environments. Our results demonstrate that our LLM-based policy generation pipeline is capable of generating functional and accurate policies with 100% accuracy. At runtime, our evaluation shows that our system configures new devices in ~150ms, and our proxy-based data plane incurs network overheads of up to 2 ms and access control overheads up to 0.3 ms.

2 Background and Requirements

This section provides background on transient collaborative IoT systems and access control auto-configuration, introduces LLM-based approaches for auto-configuration, and outlines the key system design requirements.

2.1 Transient IoT Devices in Edge Environments

Our work focuses on IoT devices in edge environments such as a home or a building. We assume that each building or a home is under a different administrative domain, and devices in that network are controlled by the home owner or an administrator. IoT systems in such environments have evolved from simple statically-mounted sensing and actuating devices to mobile, intelligent devices that interact and collaborate with one another and their environments. For example, many devices such as delivery robots, wearables (e.g., AR glasses), and smartphones are mobile and can interact with other devices. Further, such devices can enter and leave different edge environments due to their mobile nature, making them *transient devices* when visiting a different network in a new administrative domain. We refer to devices that are present in their home network environment as *native devices*, and while those that visit a different edge environment become *guest devices* in that network. Guest devices need to configured with appropriate credentials (e.g., wifi password or authorization tokens) upon entering a new network to access network and devices resources in that environment. Our hypothesis is that as the scale and frequency of transient device visits grows, such configuration can no longer be done manually (as is done in some networks today).

Some current IoT frameworks already support automated configuration primarily through service discovery, enabling devices to locate and identify each other on a network. Protocols such as UPnP [25], Zero-Conf (Apple Bonjour) [11], and Jini (Apache River) [57] provide plug-and-play capabilities, allowing devices to advertise and discover services without requiring manual configuration. However, these systems generally assume implicit trust among devices on the same network and offer little to no built-in access control or verification mechanisms. While suitable for small, static environments, these protocols are not suitable for transient devices involving multiple trust levels.

2.2 Access Control in IoT system

Since transient environments require multiple trust levels, there is a need for a user-friendly and fine-grained access control system [23, 41, 45, 54], which was not addressed in current IoT systems. For example, smart home applications such as Nest Thermostat [40], Amazon Alexa [8], and Samsung SmartThings [63] only support all-or-nothing access for guest users. Application such as Wyze [84] allows per-device sharing but fail to cover the capability level of sharing. Apple HomeKit [12] can allow additional users with limited capability options such as full control, view-only control, and local or remote control. Although these approaches offer a user-friendly interface, they often fail to offer fine-grained access control, which is crucial in transient settings, since each device needs its own specific access capabilities.

Enabling fine-grained access control, such as allowing a guest with read-only access to stream from a camera without granting it write access to control or reconfigure it, can requires manual configuration of access control policies. In dynamic settings where guest devices frequently enter and leave, manual approaches are cumbersome and do not scale, as they are primarily designed for static and single-user environments. As a result, even though fine-grained access is essential, configuration of these policies at scale is not fully addressed in most user-facing platforms due to its complexity. While access control systems such as AWS IoT [9], Azure IoT [17], and Amazon Greengrass [16] support detailed, fine-grained policies, they are developer-centric and require technical knowledge with a steep learning curve, which is unsuitable for non-expert users.

2.3 LLM-based Policy Generation

Large language models (LLMs) [78] demonstrated a strong performance across various language tasks. Trained on massive corpora of data, models like GPT-4 [3] excel at interpreting high-level descriptions, identifying semantically equivalent expressions, performing multi-step contextual reasoning, and generating coherent natural language [55, 58, 59]. These capabilities make LLMs highly valuable for applications that involve generating structured results, such as in programming tasks [81].

In particular, the use of LLMs for generating policy configurations has attracted growing interest [28, 71, 79, 87]. Their generative power enables the synthesis of configuration policies directly from natural language descriptions, significantly lowering the barrier for non-expert users. However, LLMs have inherent limitations that make them unreliable when used alone, especially in security-sensitive domains. Their outputs are probabilistic and can introduce hallucinated or logically inconsistent content. For example,

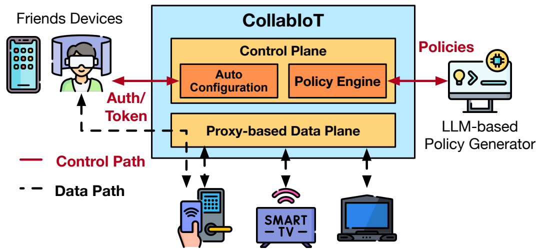


Figure 1: `CollabIoT` design overview.

research work [79] explores the use of LLMs to synthesize AWS Identity and Access Management (IAM) policies using zero-shot prompting. The study found that the synthesized policies were frequently incomparable to the ground-truth policies and tended to be overly permissive. These deviations largely stemmed from imprecise or underspecified prompts, highlighting the difficulty of producing accurate and secure policies without structured policy specification mechanisms and additional validations.

2.4 Design Requirements

To enable seamless collaboration between guests and native devices and services (e.g., embedded intelligence) running in the host environment, our system should address four design requirements:

R1 **Intuitive and Precise Policy Configuration Interface** The system should be able to translate users' high-level intent into machine-readable policies and validate the correctness of the generated policies.

R2 **Granular Access and Rate Control** Given the varying levels of trust in guest users and their devices, it is important to implement granular access and rate control. Different guests may need to interact with different subsets of native devices. Therefore, it is essential to restrict access to certain devices and services, limit the request rates, and define which capabilities are permitted.

R3 **Auto Configuration** The system should be able to address devices' transient requirements and policy changes by automatically granting access to newly arriving devices, without user intervention.

R4 **Hardware independence** Since different IoT devices expose different vendor-specific APIs, it may not be reasonable to expect a guest to know about all vendor-specific interfaces on native devices (e.g., camera for different vendors). Seamless collaboration requires the ability to use vendor- and hardware-independent abstractions.

3 `CollabIoT` Design

This section presents the architecture and design details of `CollabIoT`, our proposed AI-driven auto configuration system for collaborating devices in transient IoT environments. We begin with an overview of our system design, followed by detailed descriptions of the core system components.

3.1 Design Overview

Figure 1 depicts the architecture of our `CollabIoT` system. At the core of `CollabIoT` is a control plane comprising a policy engine and an auto configuration engine that collectively provide automated access to guest devices. `CollabIoT` provides users with an LLM-based policy generation interface, where users specify their intents via natural language, which are compiled into policies. At runtime, when a new device joins the network, the policy engines automatically create an access token based on the device's attributes and system policies. Lastly, `CollabIoT` includes a data plane based on lightweight device proxies that enforce runtime granular access and rate limits and implement hardware-independent interfaces. Next, we provide an overview of `CollabIoT`'s components:

LLM-based Policy Generator. Our LLM-based policy generator enables homeowners or administrators to use text or voice commands, significantly lowering the barrier for non-expert users to manage their IoT environments. Our LLM-based policy generation includes a processing pipeline that compiles the user's prompts into a schema-structured grouping and access control policies.

Policy Engine. The policy engine is responsible for validating the correctness of new or LLM-generated policies and maintaining a database of available devices and policies. Additionally, when a new device joins the system, the policy engine matches the device attributes with the current system policies. It then generates a list of permissible actions for the specific native devices based on their attributes.

Auto Configuration Engine. The auto configuration engine utilizes the list of permitted actions and creates cryptographically signed capability-based tokens (e.g., a JWT access token) and distributes them to the guest devices.

Proxy-based Data Plane. Our proxy-based data plane encapsulates all the device communication via a lightweight proxy that provides a hardware-independent interface and enforces access and rate control on behalf of the device.

In the remainder of this section, we first describe our access control scheme (Section 3.2), then we describe our LLM-based policy generation pipeline, and then explain our access control auto configuration process (Section 3.4) and proxy-based communication (Section 3.5). Finally, we provide an end-to-end workflow example (Section 3.6).

3.2 `CollabIoT` Access Control Scheme

First, we describe our access control scheme, which comprises a separate dynamic grouping scheme and access control policies. This enables the policy engine to construct device groups, map each device to its representative groups, enable resource owners to specify the intended device access policies, and determine devices, capabilities, or rate limits across these groups.

Table 1: Examples of hardware-independent attributes, capabilities, and APIs.

Device	Attributes	Capabilities	Generic APIs
Phone	`owner, sens-level, ID`	NA	NA
Bulb	`location, type, sens-level, ID, ...`	`switch, brightness,...`	`turn_on(), turn_off(), set_brightness()`
Camera	`location, type, sens-level, ID, ...`	`stream, rotate,...`	`live_stream(), rotate(), retrieve()`
Door lock	`location, type, sens-level, ID, ...`	`switch, configure, logs,...`	`lock(), unlock(), set_conf()`
Laptop	`location, type, sens-level, ID, ...`	`inference,...`	`inference_service()`

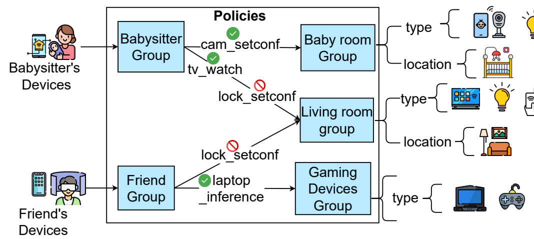


Figure 2: An end-to-end access control policy description, where babysitters are given access to cameras, lights in the baby room, and entertainment devices in the living room, while friends are given access to the gaming console and the living room devices.

3.2.1 ***Dynamic Grouping Scheme*** Our dynamic grouping scheme is an attribute-based method that utilizes device attributes to construct device lists, which simplifies access in dynamic environments. Using attributes to group devices and users is a convenient shorthand over enumerating them, which can be cumbersome. The dynamic grouping policies can use device attributes (e.g., device type or location, such as "all cameras" or "all living room devices"), context attributes (e.g., specify event or times), user attributes (e.g., all my friends), and their combinations to construct groups. We assume each device type i, has a set of attributes A_i and capabilities C_i and that all devices (guest or native) use vendor-independent attributes and capabilities descriptions. Table 1 lists examples of the hardware-independent attributes used by `CollabIoT`.

3.2.2 ***Access Control Policies*** Our access control policy scheme is a capability-based access control scheme that maps guest and local (native) device groups. The access control policy links different groups by defining source (i.e., guest devices) and destination groups (native devices) using group IDs. The access control policies define a per-device-type `include` and `exclude` lists that denote allowed and not allowed device capabilities. Additionally, our access control policies feature rate-limiting configurations that enable users to set device limits or priority scheduling policies, further facilitating secure collaboration between devices. Our utilized capabilities combine the hardware-independent capabilities names (see Table 1) within a device type, as capability names may differ across vendors.

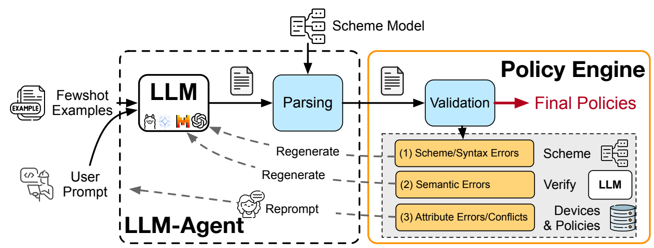


Figure 3: LLM-based policy generation pipeline.

Figure 2 shows an example access control policy with two users, a babysitter, and a gaming friend. We list the policy using our grouping and access control policy scheme in Figure 5 and discuss in Section 4. As shown, the grouping policies use location and device type attributes to generate grouping schemes (e.g., devices in the living room). The policy engine uses these groups to dynamically select devices matching the attributes at runtime, which implies handling dynamics where guests and native devices may arrive or leave. In addition, the figure shows how policies connect baby room and living room groups with the babysitter groups, enabling the assignment of fine-grained access to the babysitter.

3.3 LLM-based Policy Generation Pipeline

Generating access control policies requires a level of experience not commonly available for end-users. For instance, our configuration scheme requires knowledge of YAML and a basic understanding of access control systems. To circumvent such requirements, in `CollabIoT`, we depend on LLMs' abilities to generate structured outputs, such as in programming and configuration tasks [28, 71, 79]. For example, to define a group for the devices in the baby room, the user can prompt: "Create a group with lights and the baby monitor in the baby's room", or " All `door-locks` on the `east-side` of the building." In addition, to create an access control policy that allows his friends to access the gaming devices, the user can prompt: "Allow `my-friends` to use the `gaming-device`". The key challenge, however, in this situation is that LLMs may not produce syntactically correct outputs, may not represent the user's intent accurately, or may not be compatible with the current system configuration.

Figure 3 shows our proposed LLM-based policy generation and validation pipeline. Our LLM-based pipeline consists of two components: an LLM agent that processes user prompts and generates structured and type-safe output. In addition, it contains a validator (part of the policy engine) responsible for validating new configurations. Our pipeline behaves as follows: When a user issues a configuration prompt, `CollabIoT`'s LLM-agent augments the user prompt with a context (often referred to as few-shot examples [90]) that contains the policy definition and a few example policies based on the policy type which enhances the behavior or the LLM model. The LLM agent also instructs the LLM model to produce output that follows a structured and type-safe model scheme. The structured scheme is defined as a class representation of the YAML file that constrains the format and types of the policies as per policy type (see Listing 1). The class representation also includes rules on the allowed attributes and constraints on domain values, as well as other complex validation constraints.

To validate a new policy, the policy engine employs a three-step validation approach. First, the policy engine validates the scheme of the generated `YAML` files against the `CollabIoT` schemes. Although the LLM-agent instructs the LLM model to follow the model scheme, we found that since our scheme was not in the training set, LLM may generate a configuration with a scheme error. In this case, `CollabIoT` uses a feedback loop with LLM while referring to the mistake that the LLM created based on the custom validators we implemented in the model. Second, since LLMs may produce outputs that deviate from users' intentions (i.e., hallucinations), we implement an LLM-based *actor-critic* feedback loop that verifies semantic errors to ensure the original prompt retains the same semantic meaning as the generated prompts. To do so, we convert the generated policy into a text representation using a predefined template and feed it, along with the original prompt, to the LLM, asking it to validate that both hold the same semantics. Similar to the scheme errors, if it fails, we use a feedback loop to address the foreseen issue. Third, the policy engine validates the output against attribute errors (e.g., a user referred to a location not defined in the system) or new policies that bring configuration conflicts. In this case, we send the prompt to the user for correction. Note that in some situations, our agent may not be able to generate valid configurations, for example, when user prompts are unclear. However, in this case, our pipeline can flag their correctness and ask the user for correctness. Lastly, once the LLM-generated output is validated, we convert the generated classes into YAML files and feed them to the policy engine database.

3.4 Device Auto Configuration

When a new device joins the collaboration network, the `CollabIoT` automatically configures its permissions based on its attributes and the current policies. We assume that authenticated devices may hold a set of global attributes as part of their authentication certificates (e.g., a delivery robot may authenticate itself via a certificate that indicates and verifies its attributes). Additionally, the system administrator may tag this new device with a set of attributes that are then used for configuring the device (e.g., tagging the device as 'friends'), which can also be automated using a discovery LLM-based agent.

While auto configuration may include multiple operations, we focus on the process of generating access control capability tokens and assigning them to new devices. To configure a new device with the proper access control permissions, we rely on cryptographically signed capability-based tokens. We generate the tokens as follows. After the device joins the network, the auto configuration engine first interacts with the policy engine to find all policies that match the device's attributes, which determines the device's permissions and rate limits accessible to a guest device based on the specified policies. Each matching policy enumerates all groups mentioned within its policies and determines all matching devices. The auto configuration engine then generates a capability token (also known as an access token) for each matching device, along with the allowed capabilities (i.e., actions that are permitted on the devices), rate limits (e.g., requests per second), and the token's expiration time. The utilized structure of the tokens is presented in Section 4.1. Capability tokens are signed and encrypted by the auto configuration engine's public key. Tokens are then sent to guest devices to enable access to the native devices.

It is important to note that the policy engine generates the capabilities by merging all policies that match this device, which is achieved by combining the policies and the specified devices into a comprehensive list. For instance, if the friends' group has two policies allowing them to access all light bulbs and all devices in the living room, they will retain access to all the light bulbs in the living room and other rooms. In addition, when the policy changes, the policy engine iterates over the affected groups and their associated native devices, issuing an invalid token request. Device proxies retain a list of invalidated tokens until they expire.

3.5 Proxy-based Device Collaboration

In `CollabIoT`, devices communicate via lightweight proxies that enable vendor-independent communication, enforce access control, impose rate limits, and augment the capability of accessed devices. In `CollabIoT` each native device in the system has a proxy[1]; to interact with a device, a guest device sends an RPC request to the device's proxy along with the capability token. The proxy first verifies that the token is not in the invalid lists. Then, the proxy decrypts the request using the public key of the auto configuration engine and verifies the token's validity, capabilities, and rate

[1]Note that guest devices that provide services (e.g., a friend who brings her gaming console) require a proxy to enforce access control and rate limits.

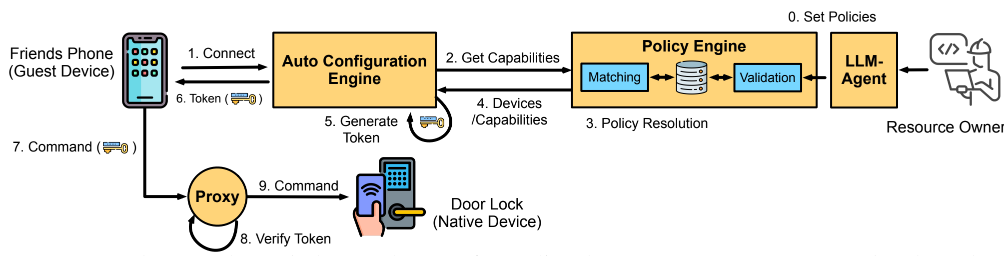


Figure 4: End-to-end workflow, where a friend's phone is given access to the door lock.

limits before allowing the request. A request is denied if the capability does not permit the requested action or if rate limits are exceeded. For efficiency reasons, signed tokens are validated during session establishment time and cached. The proxy maintains a list of verified tokens in its cache to reduce per-request verification overhead in the data path. Details of token validation are presented in Section 4.4.

Each proxy exposes a virtual, hardware-independent interface for its device, enabling device-to-device communication without requiring knowledge of vendor-specific interface details, as these interfaces vary from one vendor to another, even for the same device type. For example, all IoT cameras can be accessed using the same abstract interface regardless of the vendor. For each allowed request, the proxy performs interface translation to convert the request to the underlying vendor-specific interface and forwards the request for execution on the device. The hardware-independent interface in the proxy provides a virtual interface between access requests and specific device APIs. Table 1 shows examples of the generic APIs and capability definitions for different device types.

3.6 Collaboration Workflow

Figure 4 lists an example and shows how the resource owner and a guest device interact with `CollabIoT`. We assume all point-to-point interactions are encrypted to ensure secure interactions and token distributions. First, the resource owner uses the LLM to create a one-time dynamic grouping and access control polices which are verified by the policy engine (Step 0). When a new guest device is connected and tagged with appropriate attributes when it joins the system for the first time (e.g., a friend's phone), the auto configuration engine contacts the policy engine, which computes the accessible devices and capabilities (Steps 2-4). Subsequently, the auto configuration engine creates a token per native device (Step 5). Then, the auto configuration engine sends this token to the guest device (Step 6) along with the device proxy access information (e.g., IP address and device type). Finally, when the guest device issues a command, the native device proxy verifies the token, ensures that the requested command is authorized, and sends it to the device. Note that the interaction between native and guest devices can be *bidirectional*. To enable that, the guest can rely on a `CollabIoT`-style control plane (e.g., hosted in the cloud), which can issue capability tokens to other connected devices.

4 Implementation

This section outlines an implementation prototype of `CollabIoT` described in Section 3. Our `CollabIoT`'s prototype is implemented in Python3 using ~2.6k SLOC. `CollabIoT` components' communicate with external devices (e.g., Guest Devices) via GRPC, and unless otherwise stated, all communications are encrypted using SSL/TLS with AES-256. In the remainder of this section, we outline the implementation of our access control scheme, detail our LLM-based policy generation and validation pipeline, explain our auto-configuration implementation, and finally describe our proxy-based data plane.

4.1 Access Control Scheme

Our schemes comprise a separate dynamic grouping scheme and access control policies. We based our policy specification on NIST Next Generation Access Control (NGAC) architecture [35] and implemented an access control spec using `yaml`. Our access control policy implementation allows users to define device groups G based on the logical relationships between devices and their attributes, and access control policies P that connect these groups using the allowed capabilities and rate limits. Our dynamic grouping scheme is an attribute-based access control scheme that uses hardware-independent attributes to construct groups of accessible devices (see Table 1). The grouping scheme defines `include` and `exclude` lists per attribute type. Our specification assumes that items in the same include/exclude lists are *ORed*, while items across lists (i.e., for different attributes) are *ANDed*. Our policy engine saves the policies in the SQL database that are queried whenever a new guest device arrives.

Figure 5a lists the living room devices (see Figure 2) where the resource owner creates a group that selects "locks, TVs, and bulbs in the living room". Figure 5a lists the living room devices (see Figure 2) where the resource owner creates a group that selects "locks, TVs, and bulbs in the living room". Figure 5b and Figure 5c define the groups for gaming devices and the friends group. Figure 5d lists two access control policies from the friends' scenario in Figure 2. `policy1`

```yaml
groups:
- name: LR
  spec:
    attributes:
      location:
        includes:
        - name: livingroom
      type:
        includes:
        - name: lock
        - name: TV
        - name: bulb
```

(a) Living room group

```yaml
groups:
- name: GD
  spec:
    attributes:
      type:
        includes:
        - name: laptop
        - name: console
```

(b) Gaming devices group

```yaml
groups:
- name: FD
  spec:
    attributes:
      relation:
        includes:
        - name: friend
```

(c) Friend's devices group

```yaml
policies:
- name: policy1
  source: LR
  destination: FD
  capability:
    excludes:
    - name: lock_setconf
- name: policy2
  source: GD
  destination: FD
  capability:
    includes:
    - name: laptop_getinference
      limit: 10 req/sec
```

(d) Policies

Figure 5: Grouping scheme and access control policy to describe the policies illustrated in Figure 2.

Listing 1: Pydantic schema for group schema model.

```python
class NameEntry(BaseModel):
    name: str
AttributeTypeLiteral = Literal[
    "devLocation","devType","devStatus",
    "devManufacturer","ownerName","devBattery"
]
class AttributeSpec(BaseModel):
    includes: Optional[List[NameEntry]] = None
    excludes: Optional[List[NameEntry]] = None
class Spec(BaseModel):
    attributes: Dict[AttributeTypeLiteral,
        AttributeSpec]
class GroupItem(BaseModel):
    groupName: str
    spec: Spec
class GroupGrammar(BaseModel):
    groups: List[GroupItem]
```

permits friends to access the living room devices and all capabilities except for setting the lock configuration. In contrast, `policy2` permits the friends to access the inference service with a maximum rate of 10 req/sec.

4.2 `CollabIoT` LLM-based Pipeline

Our policy generation pipeline uses a Large Language Model (LLM) to compile user intents into a grouping scheme or access control policies. Our pipeline comprises an LLM agent responsible for generating a structure and type-safe configuration. We build our agent on top of the Pydantic LLM agent framework [76] and use models supplied by Ollama[60]. Pydantic AI utilizes Pydantic validation models, which provide an intuitive way of self-validating schemes. We formalize both group and access control policies using Pydantic validation models. Listing 1 shows an example of a schema that validates a grouping scheme and lists optional properties and domain values for attributes.

At runtime, our LLM agent takes the user prompt, the validation model, and a few-shot prompting examples (denoted as `system_prompt`) and produces results that follow the examples and the validation model. These validation models are passed to the LLM to constrain the output and ensure type-safe and structured generation. The LLM output is automatically parsed and validated against this model, enabling us to identify and correct type-missing fields or malformed outputs early. Moreover, we implement a feedback loop to reprompt the LLM in case of syntax or scheme errors as explained in Section 3.3. Additionally, we implement a semantic validation agent that ensures the generated configuration aligns with the user's intent. The validation process works by converting the generated configuration into a text representation and comparing it with the original prompt, asking the agent to confirm that they hold the same semantics. Lastly, our attribute validation is implemented using Python, which queries the policy and devices database to validate attributes and resolve conflicts.

4.3 `CollabIoT` Automatic Configuration

`CollabIoT` operates as a transparent gateway (i.e., home hub) and is able to detect new connections, issuing our auto-configuration protocol that spans the policy and auto configuration engine and provides the new device with an access token and a list of accessible devices. When a new device is authenticated, the auto configuration engine then interacts with the policy engine and computes the accessible devices, the allowed capabilities, and the rate limits for each device, as well as generates an access token for each device. Our system encodes these capabilities as JSON Web Tokens (JWT) [50]. Our tokens use official attributes (also called registered claims) and custom (private) claims such as `inc_cap`, which lists the allowed capabilities and resource limits as described in [38].

The following example explains different claims and shows how we provide `VR_headset` with capabilities to offload processing requests but not to update the gaming console configurations.

```
{
 "iss": 'CollabIoT', //Issuer (CollabIoT)
 "sub": 'VR_headset', //Subject(Guest Device)
 "aud": 'Gaming_laptop', //Audience(Native Device)
 "iat": 1731425174, //Issued At Time
 "nbf": 1731425174, //Not Before Time
 "exp": 1731459572, //Expiration Time
 "inc_cap": // Included Capabilities and Limits
    [{"offload":50}],
}
```

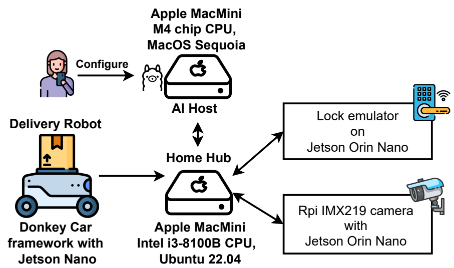


Figure 6: Experimental testbed.

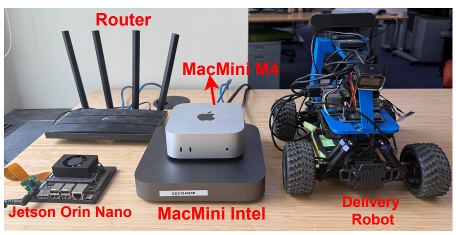


Figure 7: `CollabIoT` real testbed.

After creating the tokens, the auto configuration engine encrypts them using ED25519 256 and signs them using EdDSA digital signatures. These tokens are validated before executing the requests or starting a communication session. Our implementation assumes that the client is authenticated by validating its certificate using a certificate authority, bearer tokens, or group signatures [24], which is practical in scenarios such as the delivery robot example, where companies own a fleet of devices.

4.4 Proxy-based Device Collaboration

The device proxy implements a hardware-independent GRPC interface for its device type (see Table 1). It encapsulates the hardware-independent interfaces by translating the guest device's request to the hardware-dependent interface or utilizing a device-specific SDK (e.g., Wyze SDK [74]). The proxy uses asynchronous communication to serve concurrent access requests. We implement the rate limit with a token bucket rate limiter [73]. For ease of deployment, we implement device proxies as Docker containers, assigning one CPU core per proxy instance.

5 Evaluation

In this section, we evaluate the efficacy of `CollabIoT` using an IoT testbed and various case studies. We start by describing our experimental setup and then present our case studies that shows the end-to-end behavior of `CollabIoT`. Then we evaluate various components of `CollabIoT`. Lastly, we demonstrate the performance of the `CollabIoT` control plane and data plane with multiple micro-benchmarks and large-scale emulations.

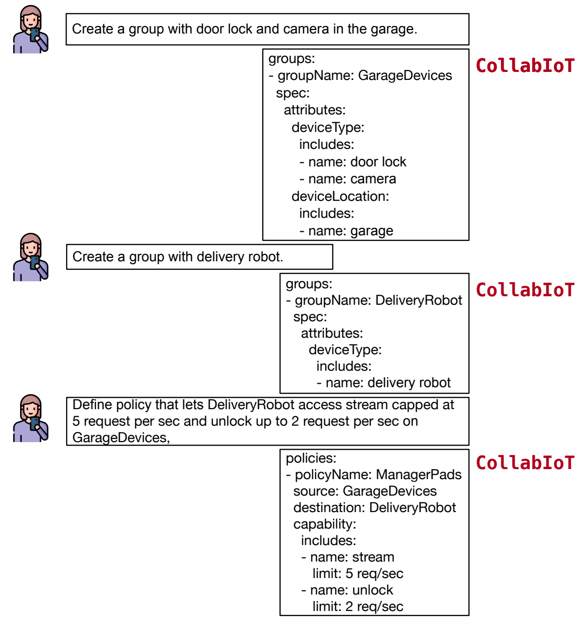


Figure 8: Delivery robot policy configuration.

5.1 Experimental Setup

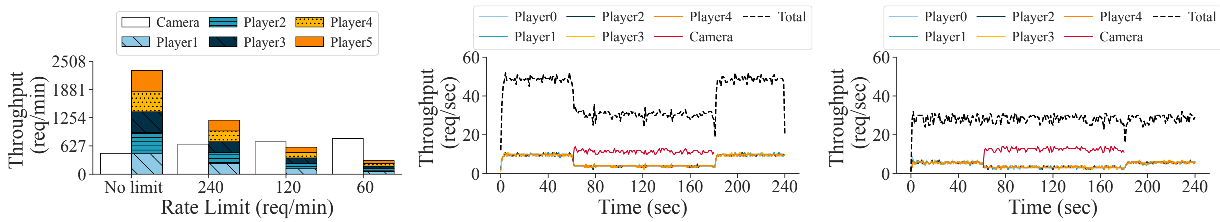
In this section, we detail our experimental setup, LLM models, and IoT devices.

Hardware Testbed. Our testbed, shown in Figure 6 and Figure 7, is based on a local IoT testbed. The testbed uses an Apple Mac mini as a home manager hub. The Mac mini has an Intel i3-8100B CPU running Ubuntu 22.04. The home manager hub hosts `CollabIoT`'s control plane and the data plane. In addition, we utilize a Mac mini with an M4 CPU to host local LLM models from ollama [60] and our LLM agent and emulate IoT devices in our large-scale experiments.

LLM Agent. Our LLM agent can use multiple local and cloud LLM models (See Table 3). We evaluate the behavior of different models under two scenarios, each with 20 grouping and 20 access control policies. Our first scenario, which we denote as `structured prompts`, assumes that the user follows a very elaborate format, which details the grouping or access control policy. Our second set of prompts, which we denote as `unstructured prompts`, represents situations where users provide unclear or concise prompts, which we acquire by manually rephrasing the 40 policies.

IoT Devices. Our testbed, shown in Figure 6 and Figure 7, includes a set of IoT devices: Raspberry Pi IMX219 camera, a delivery robot, and a Jetson Orin Nano. The Delivery Robot is based on the DonkeyCar framework [15] and comprises a Jeton Nano developer kit. The Jetson Orion Nano is used to host the AI service (developed using TensorRT)[2] and emulate different IoT devices (e.g., a door lock). Lastly, we also developed sensor and device simulation clients that we deployed on the M4 Mac mini for system scalability evaluations.

[2]https://github.com/dusty-nv/jetson-inference



(a) Fixed Rate (b) Context-aware rate limits. (c) FCFS with priority

Figure 9: Applying rate limits on an AI processing service in different settings.

Table 2: Results of Case Study 1 (Delivery Robot).

Event	Time (ms)	Verification Time (ms)
Robot Entering	151.51	NA
Lock Access	43.22	1.07
Camera Access	85.36	1.13

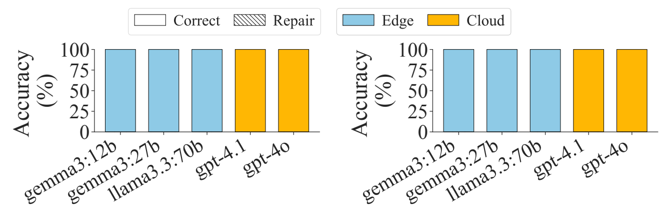
5.2 End-to-End Case Studies

We implemented two case studies to demonstrate the effectiveness and performance of `CollabIoT`.

5.2.1 ***Case Study 1 - Delivery Robot*** Our first case study demonstrates our auto configuration and device collaboration abilities by enabling a delivery robot to access the garage door lock and camera. In this experiment, the delivery robot (i.e., our donkey car) arrives at a house with a package to be delivered to a designated corner. The robot then requests access to the garage door lock and camera, which aids it in its path-planning decisions.[3] Figure 8 demonstrates how a user can prompt and generate the access control policies for the delivery robot. In this case, the user issues a prompt to create two groups, one for garage devices and the other for delivery robots. Then, prompt the system to create an access control policy that links both groups, allowing the robot to stream the garage camera at a rate of 5 frames per second and access the garage door twice to unlock and lock the door.

Table 2 shows the steps and the time taken by each step in this scenario, which does not require any human intervention — except for setting the policies, which happens beforehand. As shown, the results demonstrate the responsiveness of `CollabIoT`, where it only takes 151.51 ms to connect to the control plane, evaluate the access control policies, list the devices, generate capabilities, and encrypt the access control tokens (Steps 1 to 6 in Figure 4). In addition, the results depict the end-to-end responsiveness of the data plane, where it takes 43.22 ms to execute the unlock door command and 85.36 ms to access the camera stream. In contrast, in both cases, it takes ~1 ms to verify the access token. Lastly, we note that our policy generation time is model and deployment dependent, Table 3 presents the average policy generation time across all prompts, which we detail later in this section.

[3]We do not implement a path-planning approach but demonstrate the feasibility of accessing the camera live stream.



(a) Grouping Policy (b) Access Control Policy

Figure 10: LLM agent performance against structured prompts in (a) grouping and (b) access control policies.

5.2.2 ***Case Study 2 - AI Processing*** In our second case study, we focus on `CollabIoT`'s proxy-based device data plane and demonstrate its ability to augment device capabilities by implementing device-agnostic scheduling and rate control capabilities in the proxy. In this scenario, we consider a GPU-based AI inference service deployed on a Jetson Orin and shared between multiple users in the environment. The AI server provides a service utilized by five players (e.g., AR/VR devices engaged in a video game) and a live feed from a security camera, where all services send requests in a closed-loop manner. For simplicity, we assume that all requests are processed using an inference service that runs the ResNet152 DNN model and is deployed with the TensorRT inference framework. Figure 9 shows how `CollabIoT` can have implemented device-agnostic rate limiting and scheduling policies:

Static Rate. Figure 9a illustrates the scenario where the proxy enforces a static rate-limiting policy on the players, but imposes no limits on the camera feed. As shown, removing the rate limit allows all players and the camera to push the same number of requests — total throughput is 2,769, while each player/camera sends ~461 requests/min. However, as we apply more constrained limits, the camera feed becomes more favored, increasing its throughput by up to 790 requests per minute, when all players have a rate limit of 60 requests per minute. Note that since the camera cannot substitute for the requests of all the players, the total request rate shrinks as devices have more aggressive rate limits.

Dynamic Rate. Our second scheduling policy demonstrates dynamic rate limiting, which can be used to implement situational awareness by adjusting the rate limits (e.g., in response to camera movement detection). In this case, when the AI

Table 3: The models and sizes incorporated in our LLM agent and their average policy generation time.

Model	Parameters	Deployment	Time (s)
Gemma 3 [75]	12B	Local	7.07
Gemma 3 [75]	27B	Local	14.83
LLaMA 3.3 [42]	70B	Local	22.41
GPT-4.1 [3]		Cloud	1.22
GPT-4o [48]		Cloud	1.94

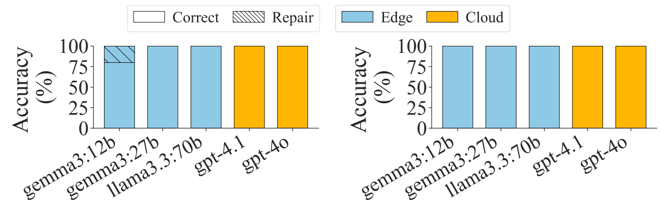
model detects any objects, the proxy applies a stricter rate limit on other players, allowing the camera to maintain its rate limits. Figure 9b shows our evaluation for four minutes, where we inject the movement detection at 60 seconds for 120 seconds. As shown, before the movement detection, each player can send ∼10 requests per second, totaling 50 requests per second. However, when the movement detection is injected, the player's rate is dropped to ∼4 requests per second, and the camera rate is maintained at ∼12 requests per second, totaling ∼30 requests per second. Note that, similar to the previous example, the cameras are unable to send enough requests to maintain the same throughput when players' devices are limited. Finally, when the system state is restored, the proxy can restore the earlier request rates.

Priority Scheduling. Our third policy shows `CollabIoT`'s ability to implement priority scheduling, where the camera maintains a higher priority than the players. In this case, the proxy features a priority queue that uses the First Come, First Served (FCFS) algorithm for tie-breaking. Figure 9c illustrates our evaluation scenario, where, similar to the previous example, the camera issues requests after 60 seconds for 120 seconds, during which the players' requests undergo multiple changes. As shown, while the total throughput remains constant, primarily limited by the FCFS behavior. The camera can be prioritized among other players' devices once it starts issuing requests and can maintain the highest request rate (of ∼12 requests per second).

Key Takeaways. *Our case studies show that* `CollabIoT` *enables users to employ granular access control policies without requiring any technical knowledge, showcasing* `CollabIoT`*'s ability to facilitate seamless communication between IoT devices, as well as our proxy-based data plane capabilities to augment devices with new features.*

5.3 LLM-based Policy Generation

In this section, we address the following question? *Are LLMs able to create valid fine-grained access control policies?* To do so, we evaluate our LLM-based policy generation and validation pipeline. As mentioned in Section 5.1, our LLM-agent is based on ollama, which enables us to evaluate multiple models using structured and unstructured prompts, each with 20 grouping and 20 access control policies. Since no standardized dataset exists for policy generation tasks, we created a custom dataset containing both structured and unstructured prompts. The unstructured prompts are specifically designed to capture extreme conditions. For example, they omit attribute types (e.g., type, location) and do not follow a consistent, structured format. Table 3 presents five LLM models that we utilized in our evaluation, ranging from 12B to 70B open-source and local models, as well as paid cloud models[4]. Note that we use each model across the entire pipeline (i.e., the same model is used both in the generation and validation parts, but with different contexts). As our goal was to produce structured output, the table highlights how the model size and deployment can affect the processing time. For instance, a small model, such as Gemma 3 (12B), takes around 7.07 seconds to produce a response, while large models deployed locally can take up to 22.41 seconds. However, the cloud models, although they have a larger size, respond faster because they utilize fast accelerators.



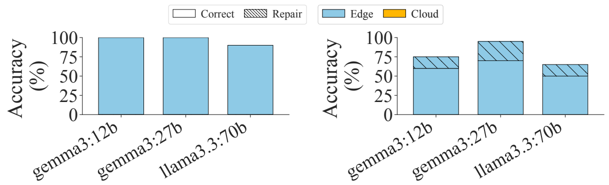
(a) Grouping Policy **(b) Access Control Policy**

Figure 11: LLM agent performance against unstructured prompts in (a) grouping and (b) access control policies.

Figure 10 illustrates the behavior of the LLM policy generation pipeline, sorted by model size, when the user provides well-structured prompts (e.g., `Create a group including devices where type is thermostat, location is bedroom and living room.`). As shown, all models are consistently able to generate functional configurations correctly for both grouping and access control policies.

Figure 11 evaluates the behavior of our LLM pipeline with scenarios where the user uses a less structured prompt (e.g., `Create a group with oven, microwave, and fridge but leave out the ones in the garage.`). Similar to Figure 10, the results show that all models generate correct configurations for access control policy. However, for grouping policy the results highlight that Gemma 3 (12B) fails to generate correct result for 20% times but due to the validation and feedback loop it repairs and gains 100% accuracy. Intuitively, in comparison to access control policy, generating output from an unstructured prompt for grouping policy is difficult, as LLM needs to classify values into different attribute types. For example, when considering a prompt such as: `Create a group with plugs in the backyard but leave out the broken ones`, the LLM needs to classify

[4] GPT-4* is a mixture of experts model, whose size is not publicly disclosed.



(a) Structured Prompts (b) Unstructured Prompts

Figure 12: LLM agent performance without fewshot examples for grouping policies.

'bulb', 'backyard', and 'broken' into different attribute types on its own.

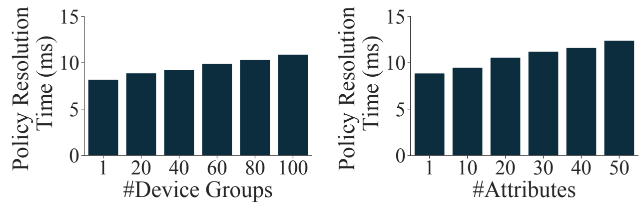
> ***Key Takeaways.*** *Our results show our policy generation pipeline boosts the performance of edge-based LLMs, making them comparable to cloud-based LLMs. In either case, our results demonstrate the feasibility of generating accurate and functional grouping and access control policies. Our results highlight the value of the validation feedback pipelines in detecting and correcting issues in the generated configurations, especially for unstructured prompts.*

5.4 Micro-benchmarks

In addition to the end-to-end evaluations, we implement micro-benchmarks to evaluate the performance of our LLM-policy generation pipeline and scalability of our `CollabIoT`' control and data planes. To ensure consistent results, we repeat each experiment 20 times and plot the average.

5.4.1 ***LLM Agent without fewshot examples*** Since policy generation requires a structured output and is not a common task for LLMs, our approach features few-shot examples to set the context for the LLM. In this experiment, we evaluate the LLM Agent's performance in the absence of such examples to assess their impact on system accuracy and highlight the role of our validation pipeline. In Figure 12, we show the performance of different LLMs when generating a grouping policy across structured and unstructured prompts. As shown, policy generation from a structured prompt is typically straightforward; we did not see much impact on our examples (see Figure 10a) . In contrast, for the unstructured prompts, the accuracy of gemma3:27b decreases from 100% (see Figure 11a) to 70%. More importantly, our pipeline was able to detect invalid outputs and correct them, raising accuracy to 95%.

5.4.2 ***Policy Engine*** Figure 13 depicts the policy resolution time (Steps 3.1 to 3.4 in Figure 4) across different numbers of device groups and attributes. Figure 13a shows the resolution time across different numbers of groups. In this case, we create a list of groups, each with ten attribute combinations and a device with 7 attributes , and record the time to compute the groups this device is allowed to access. As shown, increasing the number of groups has minor effects on the performance, where going from 1 to 20 groups only adds



(a) Varying Device Groups (b) Varying Attributes

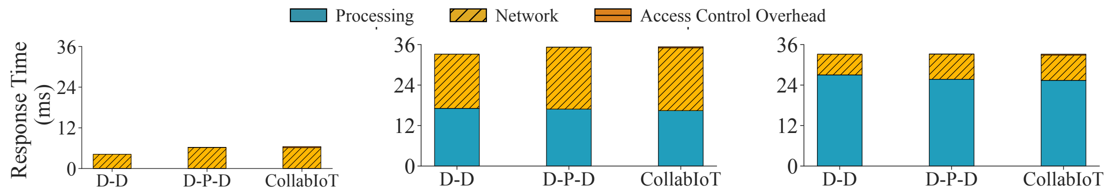
Figure 13: Evaluating policy resolution time when varying the number of groups and attributes.

0.69 ms, while going from 20 to 100 device groups only adds 2 ms. Figure 13b shows how the number of attributes inside a group affects the policy resolution time, where we create a list of 50 groups with combinations of different numbers of attributes from 1 to 50. Similarly, changing the number of attributes has minor effects, where the difference between a device with 1 and 50 attributes is only 3.52 ms.

5.4.3 ***Access Tokens*** Figure 15 evaluates the performance of our capability-based authorization scheme across a different number of capabilities. The figure focuses on the capability generation time (ms) and the final token sizes (bytes). As shown, the capability generation time is constant across different numbers of capabilities (∼1ms). The token size, however, increases sublinearly with the number of assigned capabilities, where the token size increases by 851B when the token contains 20 capabilities.

5.4.4 ***Proxy Overhead*** In this section, we evaluate the overheads of our lightweight proxy-based architecture compared to direct device-to-device interactions. Figure 14 evaluates the performance of proxy-based architecture in three service types: `get_status, load_image`, and `process_image` services. Figure 14a) depicts the performance of `get_status` request, Figure 14b) implements an image load service (e.g., loading a camera feed), where the network overheads become more pronounced, and Figure 14c shows the time for the AI processing (e.g., image captured and classified on Jetson Orin Nano using ResNet-152.), where most of the time is spent at the devices. We note that in all cases, the proxy adds up to 2 ms to the end-to-end response time, and access control overhead (including the token verification and enforcing rate limits) adds a maximum of 0.3 ms.

5.4.5 ***Run-time Scalability*** Next, we evaluate the scalability of our run-time environment. Figure 16 shows the end-to-end response time of the get status functionality, which includes the time for token verification (i.e., access control overhead), network time, and processing time. In this figure, we change the number of guests from 5 to 50 concurrent guest devices accessing a single device proxy. As shown in Figure 16, the results demonstrate that the proxy can support multiple clients with small overheads. For instance, the results show that while a single client requires 6.46 ms per request (see Figure 14a), five clients have an average response



(a) Get Status (b) Load Image (c) AI Processing

Figure 14: Evaluating the performance of proxy-based architecture across three service types.

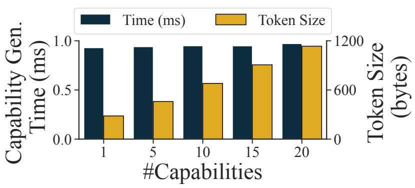


Figure 15: Token generation time and size across capabilities #.

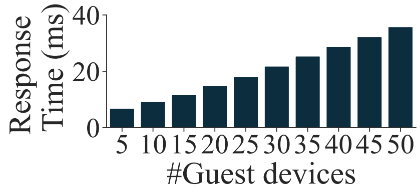


Figure 16: Response time of get status service for concurrent requests.

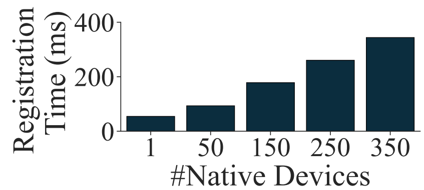


Figure 17: Registration time across multiple native devices.

time of 6.83 ms, representing a 5.7% overhead. Moreover, the results show that our system can withstand 50 clients with minimal overhead, resulting in a latency of 35.8 ms.

5.4.6 ***Control Plane Scalability*** In this section, we evaluate the scalability of the control plane in environments with multiple native devices requiring access management. As shown in Figure 17, we measure the registration time, which includes generating access tokens and distributing them to guest devices. As the number of native devices increases from 1 to 350, the registration time incurs an overhead of only 5.31%. This demonstrates that our capability-based access control can limit the registration overhead, which remains low even as the number of native devices scales. Intuitively, when a roaming device enters the environment, the capability-based approach only requires generating and distributing capabilities to the roaming device, rather than reconfiguring access for multiple native devices.

> ***Key Takeaways.*** *Our large-scale evaluation demonstrates the scalability of* `CollabIoT` *across different levels of policy complexities, including different attributes and capabilities. Additionally, our evaluation reveals that our proxy-based data plane incurs network overheads of up to 2 ms and access control overhead up to 0.3 ms.*

6 Discussion

6.1 Control Plane Deployment

`CollabIoT`'s control plane deployment relies on an edge architecture in which policy generation and auto-configuration are handled on a Mac Mini. Edge-based execution offers complete autonomy and privacy, but at the cost of higher local computation overhead for compute-intensive tasks such as LLM-based policy generation. However, as we show in our cloud-based policy generation, such compute-intensive tasks can be offloaded to cloud environments, which can reduce local computation latency, as seen in Table 3, but introduce dependencies and potential exposure risks. Nonetheless, `CollabIoT` can mitigate such risks during policy generation by transmitting only natural-language commands, while keeping validation local, ensuring that no sensitive device or environmental data leaves the local setup.

6.2 Data Plane Design

Our design employs a proxy that contains device-specific logic to enable collaboration across heterogeneous IoT devices. While our current implementation requires an additional effort to support new device types, our approach can be integrated with emerging interoperability standards such as Matter, which define unified data models and communication protocols that can simplify device integration.

6.3 Access Control Enforcement

`CollabIoT`'s access control enforcement focuses on application-level enforcement, where policies are generated and enforced within the application logic to manage device collaboration. However, our approach can be complemented with system-level security mechanisms such as Mandatory Access Control (MAC), Discretionary Access Control (DAC), sandboxing, or Trusted Execution Environments (TEEs).

6.4 Policy Generation

For LLM-based policy generation, we relied on user-specified instructions through prompts. Nonetheless, our approach can be complemented with learning-based mechanisms that auto-configure the access control policies. For instance, `CollabIoT` can utilize configuration mining (e.g., learning rate limits) from historical logs or directly infer appropriate access control components (e.g., rate limits) from the context of a given use case. Together, these directions can enable adaptive, data-driven access control in CollabIoT.

7 Related Work

Automatic Policy Generation. Policy generation frameworks have evolved from structured rule-based interfaces to machine learning–assisted generation, and more recently, to large language models (LLMs)-based generation. Earlier research enabled users to specify access policies using templates or constrained logic [26, 32, 47, 51, 77]. While effective for simple rules, these approaches struggled with scalability and user comprehension as policy complexity increased. To overcome these limitations, subsequent work employed machine learning (ML) and natural language processing (NLP) to automate policy generation [6, 33, 46, 49, 85], though such methods often required domain-specific retraining. More recent efforts leverage LLMs' ability to understand and generate structured outputs to directly generate policies from natural language [28, 69, 71, 79, 87], thereby improving generalization and adaptability. However, these approaches lack mechanisms to validate the correctness of generated policies. In contrast, CollabIoT combines LLM-based synthesis with multi-stage validation to produce accurate, fine-grained, and enforceable access control policies aligned with user intent.

Access Control in Transient Environments. Although central authorization approaches (e.g., OAuth 2.0 [44]) is widely used for delegating access via bearer tokens that get validated in a centralized authorization server. In decentralized settings, however, validating resources via a central authority may not be adequate, as it introduces unnecessary overhead. To address this issue, Capability-Based Access Control (CapBAC) offers an alternative for such settings by encoding access rights directly in signed tokens, allowing users to generate fine-grained and self-validating access tokens [19, 37, 38, 43]. While CapBAC approaches address fine-grained and centralization issues, they often lack mechanisms to configure access to devices automatically. Thus, Attribute-Based Access Control (ABAC) has been proposed to handle fine-grained, context-aware access in smart homes [10, 21, 41, 66, 67], supporting multi-user and multi-device scenarios. CollabIoT draws parallels from hybrid models that combine ABAC and CapBAC [4, 20, 82] and issue capability tokens based on attribute evaluation. Moreover, CollabIoT utilizes an LLM-based approach to facilitate policy generation.

Seamless Collaboration in heterogeneous environments. Many earlier efforts have addressed communication issues between heterogeneous devices. Jini (Apache River) [57], introduced mobile Java proxies to enable vendor-independent communication. However, its reliance on the Java Virtual Machine made it impractical for today's resource-constrained IoT devices. CollabIoT adopts the proxy concept but improves it by localizing translation logic at the device edge, enabling lightweight RPCs from any client. Cloud-based platforms like IFTTT [1], TTEO [88], and Alexa Routines [7] support user-defined automation via trigger-action rules but treat devices as passive and rely on centralized clouds, introducing latency and limiting real-time collaboration. CollabIoT, in contrast, enables direct, local communication and fine-grained access control. Decentralized systems like MOSDEN [61], MATT [56], Coulson's contact-action framework [30], and SIoT [14] explore opportunistic collaboration but lack runtime policy enforcement and vendor abstraction, limiting their use in dynamic settings. Standardization efforts like AllJoyn [29], IoTivity [2], and Matter [31] aim for cross-vendor interoperability but require complex onboarding or firmware compliance. CollabIoT complements these efforts by supporting transient, heterogeneous devices through dynamic proxies that enforce access policies locally and abstract away vendor-specific APIs.

8 Conclusion

This paper presented CollabIoT, a system that enables secure and seamless device collaboration in transient IoT environments. CollabIoT address the security concerns by allowing users to define fine-grained access control policies and employs an LLM-based policy generation pipeline to generate validated access control policies from natural language. To support real-time collaboration, CollabIoT automates authorization by distributing capability tokens to transient devices and employs lightweight proxies for policy enforcement. These proxies expose hardware-independent APIs, enabling interactions across heterogeneous devices. We demonstrate that our LLM-based policy generation achieves 100% accuracy in generating functional and correct policies. At runtime, our evaluation shows that our system configures new devices in ~150 ms, and our proxy-based data plane incurs network overheads of up to 2 ms and access control overheads up to 0.3 ms. In future work, we plan to integrate CollabIoT with vendor-agnostic IoT frameworks such as Matter and extend the role of LLMs in device discovery and tagging.

Acknowledgment

We thank the anonymous reviewers for their valuable insight and feedback. This research is supported by National Science Foundation (NSF) grants 2211301, 2211302, 2325956, 2213636, 2211888 and US Army grant W911NF-17-2-0196.

References

[1] 2025. IFTTT. https://ifttt.com/. Accessed: 2025-06-26.
[2] 2025. IoTivity. https://iotivity.org/. Accessed: 2025-06-25.
[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[4] Ashraf Alkhresheh, Khalid Elgazzar, and Hossam S. Hassanein. 2020. DACIoT: Dynamic Access Control Framework for IoT Deployments. *IEEE Internet of Things Journal* 7, 12 (2020), 11401–11419. doi:10.1109/JIOT.2020.3002709

[5] Sultan H Almotiri, Murtaza A Khan, and Mohammed A Alghamdi. 2016. Mobile health (m-health) system in the context of IoT. In *2016 IEEE 4th international conference on future internet of things and cloud workshops (FiCloudW)*. IEEE, 39–42.

[6] Manar Alohaly, Hassan Takabi, and Eduardo Blanco. 2019. Automated extraction of attributes from natural language attribute-based access control (ABAC) policies. *Cybersecurity* 2, 1 (2019), 2.

[7] Amazon. 2025. AlexaRoutines. https://www.amazon.com/alexa-routines/b?ie=UTF8&node=21442922011/. Accessed: 2025-06-25.

[8] Amazon. 2025. Amazon Alexa. https://www.amazon.com/dp/B0DCCNHWV5?ref=aucc_web_red_xaa_evgn_tx_0002. Accessed: 2025-06-25.

[9] Amazon. 2025. AWS IOT. https://aws.amazon.com/iot/. Accessed: 2025-06-25.

[10] Safwa Ameer, James Benson, and Ravi Sandhu. 2022. An attribute-based approach toward a secured smart-home IoT access control and a comparison with a role-based approach. *Information* 13, 2 (2022), 60.

[11] Apple. 2025. Bonjour. https://developer.apple.com/bonjour/. Accessed: 2025-06-19.

[12] Apple Inc. 2024. Apple Home App. https://www.apple.com/home-app/ Accessed: 2024-11-12.

[13] Associated Students UCLA (ASUCLA). [n. d.]. UCLA Food Delivery with Starship Robots. https://www.asucla.ucla.edu/starship Accessed: 2024-11-05.

[14] Luigi Atzori, Antonio Iera, Giacomo Morabito, and Michele Nitti. 2012. The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization. *Computer networks* 56, 16 (2012), 3594–3608.

[15] Autorope. 2023. Donkeycar: a python self driving library. https://github.com/autorope/donkeycar. Accessed: 2023-11-30.

[16] AWS. 2023. AWS IoT Greengrass Documentation. https://docs.aws.amazon.com/greengrass/. Accessed: 2023-10-23.

[17] Azure. 2023. Azure IoT - Internet of Things. https://azure.microsoft.com/en-us/solutions/iot. Accessed: 2023-10-23.

[18] Chris Baird. 2024. The IoT Trends Shaping Industry in 2024 and Beyond. https://iotbusinessnews.com/2024/09/01/08700-the-iot-trends-shaping-industry-in-2024-and-beyond/. Accessed: 2024-11-12.

[19] Fatih Bakir, Chandra Krintz, and Rich Wolski. 2021. CAPLets: Resource Aware, Capability-Based Access Control for IoT. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. 106–120. doi:10.1145/3453142.3491289

[20] Emmanuel Bertin, Dina Hussein, Cigdem Sengul, and Vincent Frey. 2019. Access control in the Internet of Things: a survey of existing approaches and open research questions. *Annals of telecommunications* 74 (2019), 375–388.

[21] Bruhadeshwar Bezawada, Kyle Haefner, and Indrakshi Ray. 2018. Securing Home IoT Environments with Attribute-Based Access Control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control* (Tempe, AZ, USA) *(ABAC'18)*. Association for Computing Machinery, New York, NY, USA, 43–53. doi:10.1145/3180457.3180464

[22] Óscar Blanco-Novoa, Paula Fraga-Lamas, Miguel A. Vilar-Montesinos, and Tiago M Fernández-Caramés. 2020. Creating the internet of augmented things: An open-source framework to make iot devices and augmented and mixed reality systems talk to each other. *Sensors* 20, 11 (2020), 3328.

[23] Logan Blue, Samuel Marchal, Patrick Traynor, and N. Asokan. 2021. Lux: Enabling Ephemeral Authorization for Display-Limited IoT Devices. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation* (Charlottesvle, VA, USA) *(IoTDI '21)*. Association for Computing Machinery, New York, NY, USA, 15–27. doi:10.1145/3450268.3453530

[24] Dan Boneh, Xavier Boyen, and Hovav Shacham. 2004. Short Group Signatures. In *Annual international cryptology conference*. Springer, 41–55.

[25] M. Boucadair, R. Penno, and D. Wing. 2013. RFC 6970: Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF).

[26] Giovanni Campagna, Silei Xu, Rakesh Ramesh, Michael Fischer, and Monica S. Lam. 2018. Controlling Fine-Grain Sharing in Natural Language with a Virtual Assistant. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3, Article 95 (Sept. 2018), 28 pages. doi:10.1145/3264905

[27] Pratiksha Chaudhari, Yang Xiao, Mark Ming-Cheng Cheng, and Tieshan Li. 2024. Fundamentals, Algorithms, and Technologies of Occupancy Detection for Smart Buildings Using IoT Sensors. *Sensors* 7 (2024). doi:10.3390/s24072123

[28] Ye Cheng, Minghui Xu, Yue Zhang, Kun Li, Hao Wu, Yechao Zhang, Shaoyong Guo, Wangjie Qiu, Dongxiao Yu, and Xiuzhen Cheng. 2025. Say What You Mean: Natural Language Access Control with Large Language Models for Internet of Things. arXiv:2505.23835 [cs.CL] https://arxiv.org/abs/2505.23835

[29] Open connectivity foundation. 2025. Alljoyn. https://openconnectivity.org/technology/reference-implementation/alljoyn/. Accessed: 2025-06-25.

[30] Geoff Coulson, Gordon Blair, Yehia Elkhatib, and Andreas Mauthe. 2015. The design of a generalised approach to the programming of systems of systems. In *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 1–6. doi:10.1109/WoWMoM.2015.7158188

[31] CSA. 2025. Matter. https://csa-iot.org/all-solutions/matter/. Accessed: 2025-06-25.

[32] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. 2001. The ponder policy specification language. In *Policies for Distributed Systems and Networks: International Workshop, POLICY 2001 Bristol, UK, January 29–31, 2001 Proceedings*. Springer, 18–38.

[33] Jose M Del Alamo, Danny S Guaman, Boni García, and Ana Diez. 2022. A systematic mapping study on automated analysis of privacy policies. *Computing* 104, 9 (2022), 2053–2076.

[34] Dominos. 2025. Dominos Drone Delivery. https://www.dominos.co.nz/landing-pages/domino-s-drone-delivery-faqs Accessed: 2025-06-27.

[35] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. 2011. The Policy Machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture* 57, 4 (2011), 412–424. doi:10.1016/j.sysarc.2010.04.005

[36] Giancarlo Fortino, Stefano Galzarano, Raffaele Gravina, and Wenfeng Li. 2015. A framework for collaborative computing and multi-sensor data fusion in body sensor networks. *Information Fusion* 22 (2015), 50–70. doi:10.1016/j.inffus.2014.03.005

[37] Nikos Fotiou, Vasilios A. Siris, and George C. Polyzos. 2021. Capability-based access control for multi-tenant systems using OAuth 2.0 and Verifiable Credentials. In *2021 International Conference on Computer Communications and Networks (ICCCN)*. 1–9. doi:10.1109/ICCCN52240.2021.9522214

[38] Nikos Fotiou, Vasilios A. Siris, George C. Polyzos, Yki Kortesniemi, and Dmitrij Lagutin. 2022. Capabilities-based access control for IoT devices using Verifiable Credentials. In *2022 IEEE Security and Privacy*

Workshops (SPW). 222–228. doi:10.1109/SPW54247.2022.9833873

[39] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. 2015. Edge-centric Computing: Vision and Challenges. *SIGCOMM Comput. Commun. Rev.* 45, 5 (Sept. 2015), 37–42. doi:10.1145/2831347.2831354

[40] Google. 2023. Google Nest. https://safety.google/security-privacy/. Accessed: 2023-10-23.

[41] Gaurav Goyal, Peng Liu, and Shamik Sural. 2022. Securing smart home iot systems with attribute-based access control. In *Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*. 37–46.

[42] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[43] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. 2013. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling* 58, 5-6 (2013), 1189–1205.

[44] D. Hardt. 2012. RFC 6749: The OAuth 2.0 Authorization Framework.

[45] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. 2018. Rethinking Access Control and Authentication for the Home Internet of Things (IoT). In *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD, 255–272.

[46] John Heaps, Ram Krishnan, Yufei Huang, Jianwei Niu, and Ravi Sandhu. 2021. Access control policy generation from user stories using machine learning. In *Data and Applications Security and Privacy XXXV: 35th Annual IFIP WG 11.3 Conference, DBSec 2021, Calgary, Canada, July 19–20, 2021, Proceedings 35*. Springer, 171–188.

[47] James Hong, Amit Levy, and Philip Levis. 2016. Demo: Building Comprehensible Access Control for the Internet of Things using Beetle. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion* (Singapore, Singapore) *(MobiSys '16 Companion)*. Association for Computing Machinery, New York, NY, USA, 102. doi:10.1145/2938559.2938578

[48] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).

[49] Sakuna Harinda Jayasundara, Nalin Asanka Gamagedara Arachchilage, and Giovanni Russello. 2024. SoK: Access Control Policy Generation from High-level Natural Language Requirements. *Comput. Surveys* 57, 4 (2024), 1–37.

[50] Michael Jones, John Bradley, and Nat Sakimura. 2015. Json Web Token (JWT). https://datatracker.ietf.org/doc/html/rfc7519.

[51] Lalana Kagal. 2002. *Rei : A Policy Language for the Me-Centric Project.* Technical Report. HP Labs. http://www.hpl.hp.com/techreports/2002/HPL-2002-270.html.

[52] Yoonsang Kim, Sanket Goutam, Amir Rahmati, and Arie Kaufman. 2023. Erebus: Access Control for Augmented Reality Systems. In *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA, 929–946. https://www.usenix.org/conference/usenixsecurity23/presentation/kim-yoonsang

[53] Rajalakshmi Krishnamurthi, Adarsh Kumar, Dhanalekshmi Gopinathan, Anand Nayyar, and Basit Qureshi. 2020. An overview of IoT sensor data processing, fusion, and analysis techniques. *Sensors* 20, 21 (2020), 6076.

[54] Sunjae Lee, Minwoo Jeong, Daye Song, Junyoung Choi, Seoyun Son, Jean Y Song, and Insik Shin. 2024. FLUID-IoT: Flexible and Fine-Grained Access Control in Shared IoT Environments via Multi-user UI Distribution. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 848, 16 pages. doi:10.1145/3613904.3641991

[55] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

[56] Wyatt Lindquist, Sumi Helal, Ahmed Khaled, Gerald Kotonya, and Jaejoon Lee. 2020. MAAT: Mobile Apps As Things in the IoT. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4, Article 143 (Sept. 2020), 22 pages. doi:10.1145/3369823

[57] Sun Microsystems. 2025. Apache River. https://river.apache.org/. Accessed: 2025-06-19.

[58] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.

[59] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[60] Ollama Inc. 2025. Ollama: Run large language models locally. https://ollama.com/. Accessed on June 20, 2025.

[61] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. MOSDEN: An Internet of Things Middleware for Resource Constrained Mobile Devices. In *2014 47th Hawaii International Conference on System Sciences*. 1053–1062. doi:10.1109/HICSS.2014.137

[62] Sandeep Pirbhulal, Heye Zhang, Md Eshrat E Alahi, Hemant Ghayvat, Subhas Chandra Mukhopadhyay, Yuan-Ting Zhang, and Wanqing Wu. 2017. A Novel Secure IoT-Based Smart Home Automation System Using a Wireless Sensor Network. *Sensors* 17, 1 (2017). doi:10.3390/s17010069

[63] Samsung. 2025. Samsung Smartthings. https://www.samsung.com/us/smartthings/. Accessed: 2025-06-25.

[64] João Santos, Joel JPC Rodrigues, Bruno MC Silva, João Casal, Kashif Saleem, and Victor Denisov. 2016. An IoT-based mobile gateway for intelligent personal assistants on mobile health environments. *Journal of Network and Computer Applications* 71 (2016), 194–204.

[65] Hetvi Shastri, Walid A. Hanafy, Li Wu, David Irwin, Mani Srivastava, and Prashant Shenoy. 2025. *Poster Abstract: Rethinking Collaboration Among Mobile Devices in IoT Environments*. Association for Computing Machinery, New York, NY, USA, 676–677. https://doi.org/10.1145/3715014.3724065

[66] Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A. Selcuk Uluagac. 2020. KRATOS: Multi-User Multi-Device-Aware Access Control System for the Smart Home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Linz, Austria) *(WiSec '20)*. 1–12. doi:10.1145/3395351.3399358

[67] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. 2022. Who's controlling my device? Multi-user multi-device-aware access control system for shared smart home environment. *ACM Transactions on Internet of Things* 3, 4 (2022), 1–39.

[68] Satyajit Sinha. 2024. State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally. https://iot-analytics.com/number-connected-iot-devices/. Accessed: 2025-06-19.

[69] Pratik Sonune, Ritwik Rai, Shamik Sural, Vijayalakshmi Atluri, and Ashish Kundu. 2025. LMN: A Tool for Generating Machine Enforceable Policies from Natural Language Access Control Rules using LLMs.

arXiv preprint arXiv:2502.12460 (2025).

[70] Biljana Risteska Stojkoska and Kire Trivodaliev. 2017. A review of Internet of Things for smart home: Challenges and solutions. *Journal of Cleaner Production* 140 (2017), 1454–1464. https://api.semanticscholar.org/CorpusID:53696817

[71] Pranav Subramaniam and Sanjay Krishnan. 2024. Intent-Based Access Control: Using LLMs to Intelligently Manage Access Control. *arXiv preprint arXiv:2402.07332* (2024).

[72] Yongbin Sun, Alexandre Armengol-Urpi, Sai Nithin Reddy Kantareddy, Joshua Siegel, and Sanjay Sarma. 2019. Magichand: Interact with iot devices in augmented reality environment. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 1738–1743.

[73] Puqi Perry Tang and T-YC Tai. 1999. Network traffic characterization using token bucket model. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, Vol. 1. IEEE, 51–62.

[74] Shaun Tarves. 2023. Wyze-SDK. https://github.com/shauntarves/wyze-sdk

[75] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295* (2024).

[76] Pydantic Team. 2024. Pydantic AI. https://ai.pydantic.dev/ Accessed: 2025-06-18.

[77] A. Uszok, J.M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken. 2004. KAoS policy management for semantic Web services. *IEEE Intelligent Systems* 19, 4 (2004), 32–41. doi:10.1109/MIS.2004.31

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[79] Adarsh Vatsa, Pratyush Patel, and William Eiers. 2025. Synthesizing Access Control Policies using Large Language Models. *arXiv preprint arXiv:2503.11573* (2025).

[80] Ruchi Vishwakarma and Ankit Kumar Jain. 2020. A survey of DDoS attacking techniques and defence mechanisms in the IoT network. *Telecommunication Systems* 73, 1 (2020), 3–25. doi:10.1007/s11235-019-00599-z

[81] Jianxun Wang and Yixiang Chen. 2023. A Review on Code Generation with LLMs: Application and Evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*. 284–289. doi:10.1109/MedAI59581.2023.00044

[82] Qiong Wang, Xia Feng, Liangmin Wang, Haiqin Wu, and Boris Düdder. 2025. FECAC: Fine-Grained and Efficient Capability-Based Access Control for Enterprize-Scale IoT Systems. *IEEE Internet of Things Journal* 12, 7 (2025), 8669–8684. doi:10.1109/JIOT.2024.3504825

[83] Stacey Wolfson. 2023. Amazon Key In-Garage Delivery: What It Is and How It Works. https://www.aboutamazon.com/news/amazon-prime/what-is-amazon-key-in-garage-delivery Accessed: 2024-11-10.

[84] Wyze. 2025. Wyze. https://www.wyze.com/?srsltid=AfmBOopL3p7Rek-xibYqFso1jSLiEaps6vQxe5jLdu2DT_r58DZFIXu-. Accessed: 2025-06-25.

[85] Yutang Xia, Shengfang Zhai, Qinting Wang, Huiting Hou, Zhonghai Wu, and Qingni Shen. 2022. Automated extraction of ABAC policies from natural-language documents in healthcare systems. In *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 1289–1296.

[86] Li Da Xu, Wu He, and Shancang Li. 2014. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics* 10, 4 (2014), 2233–2243. doi:10.1109/TII.2014.2300753

[87] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* (2024), 100211.

[88] Jaeseok Yun, Il-Yeup Ahn, Sung-Chan Choi, and Jaeho Kim. 2016. TTEO (Things Talk to Each Other): Programming smart spaces based on IoT systems. *Sensors* 16, 4 (2016), 467.

[89] Eric Zeng and Franziska Roesner. 2019. Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study. In *28th USENIX Security Symposium (USENIX Security 19)*. 159–176.

[90] Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate Before Use: Improving Few-Shot Performance of Language Models. arXiv:2102.09690 [cs.CL] https://arxiv.org/abs/2102.09690