Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Human-in-the-Loop Reinforcement Learning for Energy-Efficient Smart Home Recommendations

youcef GUENFAF

yguenfaf@usthb.dz

University of Sciences and Technology Houari Boumediene

youcef ZAFOUNE

University of Sciences and Technology Houari Boumediene

Amdjed Baha-eddine Chaabane

University of Sciences and Technology Houari Boumediene

Abdnour Benmeridja

University of Sciences and Technology Houari Boumediene

Dounia Fatma Zohra Rehab

University of Sciences and Technology Houari Boumediene

Douniazed Daouadji

University of Sciences and Technology Houari Boumediene

Imad Yacine Karoum

University of Sciences and Technology Houari Boumediene

Research Article

Keywords:

Posted Date: November 12th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11168153/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Human-in-the-Loop Reinforcement Learning for Energy-Efficient Smart Home Recommendations

Youcef Guenfaf[1*], Youcef Zafoune[1], Amdjed Baha-eddine Chaabane[1], Abdnour Benmeridja[1], Dounia Fatma Zohra Rehab[1], DouniaZed Daouadji[1], Imad Yacine Karoum[1]

[1*]Faculty of Computer Science, University of Science and Technology Houari Boumediene (USTHB), BP 32 El Alia, Bab Ezzouar, Algiers, 16111, Algeria.

*Corresponding author(s). E-mail(s): yguenfaf@usthb.dz;
Contributing authors: yzafoune@usthb.dz;
amdjedbahaeddine.chaabane@etu.usthb.dz;
abdenour.benmeridja@etu.usthb.dz;
douniafatmazohra.rehab@etu.usthb.dz;
douniazed.daouadji@etu.usthb.dz; imadyacine.karoum@etu.usthb.dz;

Abstract

Smart homes generate a continuous stream of sensor, occupancy and appliance data that is well suited to data-driven energy optimisation, yet the majority of deployed home energy management systems still rely on static rule-based control and manual scheduling. Such systems adapt poorly to evolving occupant routines, seasonal variation and conflicting preferences among household members, which results either in avoidable energy waste or in comfort losses. This paper presents a recommender system whose policy is learned entirely from occupant feedback: a Proximal Policy Optimization (PPO) agent is trained online from accept and reject signals, with no pre-collected preference corpus, and is paired with a deterministic rule engine that supplies a behavioural prior during the cold-start period. The system is embedded in a complete Internet of Things (IoT) architecture comprising a layered application server, a relational data model and an Android client. The agent observes a nine-dimensional state vector combining temporal, environmental, device-state and rule-baseline features, and emits a device action together with a confidence score and a natural-language justification. Recommendations are validated against a priority-ordered safety rule

set and reconciled across occupants through a combined weighted-average and majority-voting strategy before being delivered to the user. The nine PPO hyperparameters were tuned by Bayesian optimisation using a Tree-structured Parzen Estimator over a lightweight surrogate environment. The system was evaluated on a synthetic but behaviourally grounded one-year dataset generated at a thirty-minute resolution for a three-occupant, six-room dwelling, under three user-feedback regimes and against a rule-based baseline and a Deep Q-Network (DQN) variant. Under full acceptance the system reduced annual consumption by 14.2% on average, reaching 27.8% in the most favourable month, while under a realistic mixed-feedback regime it delivered 6.4% average annual savings with an agent confidence of approximately 74%. PPO outperformed the DQN variant in every month of the simulated year, with the gap widening in periods dominated by continuous set-point control. The results indicate that policy-gradient reinforcement learning driven by human feedback and anchored by a rule-based prior is an effective basis for adaptive, comfort-preserving energy recommendations in residential settings.

Keywords: Smart home, Energy efficiency, Recommender system, Human-in-the-loop learning, Reinforcement learning, Proximal policy optimization, Internet of Things

1 Introduction

The Internet of Things (IoT) has moved from a networking abstraction to an operational substrate for everyday environments, connecting sensors, actuators and appliances over shared communication infrastructure [1, 2]. Among its application domains, the residential dwelling has proved to be one of the most consequential: smart homes combine dense sensing, a heterogeneous device population and a resident population whose behaviour is both highly structured and highly variable [3, 4]. This combination makes the smart home simultaneously a rich source of data and a demanding control problem.

Energy consumption is the dimension of that problem with the clearest societal weight. Residential demand accounts for a substantial share of national electricity use, and a significant fraction of it is discretionary: lights left on in unoccupied rooms, heating and cooling set-points that ignore ambient conditions, and appliances operating outside of the periods in which their output is actually valued. Smart home energy management systems (SHEMS) have consequently attracted sustained research attention [5–7].

Despite this attention, a recurring architectural pattern limits the effectiveness of deployed systems. Control logic is predominantly static: if-then rules and manual schedules, fixed at configuration time and revised only when the occupant notices a problem. Static logic cannot track the drift in occupant routines that accompanies seasonal change, employment change or household composition change, and it cannot represent the trade-off between energy and comfort in a way that reflects an individual household's tolerance. The consequence is a system that either enforces savings at the

cost of comfort, and is subsequently disabled by its users, or defers to comfort and delivers little saving.

A second limitation concerns the treatment of the occupant. Much of the literature frames the problem as one of automated control, in which the system decides and the occupant observes. Where recommendation is used instead of automation, the recommendation logic is frequently itself static, or is learned offline from a historical corpus that does not reflect the preferences of the specific household in which the system operates. Relatively little work has examined reinforcement learning agents that treat the occupant's accept/reject response as the primary training signal and adapt continuously to it.

This paper addresses that gap. We propose, implement and evaluate a recommender system for smart home energy efficiency whose policy is learned exclusively from the occupant's own accept and reject responses under a human-in-the-loop paradigm, and in which a deterministic rule engine supplies a physically sensible prior until that feedback accumulates, with a Proximal Policy Optimization (PPO) agent [8] progressively assuming control of the recommendation. We refer to this pairing of a rule-based prior with a learned policy as a hybrid architecture; the term is used throughout in that sense, and not in the classical recommender-systems sense of combining collaborative and content-based filtering [9]. The contributions of the work are the following.

- A human-in-the-loop training scheme in which the policy is refined exclusively from occupant accept/reject signals through an asymmetric reward, without any pre-collected preference dataset, so that each household converges on a policy shaped by its own revealed preferences.
- A rule-guided hybrid architecture in which the rule engine's output is injected directly into the agent's observation vector, allowing the learned policy to treat it as an informative prior rather than as a hard constraint, and enabling safe behaviour from the first interaction.
- A complete system realisation with layered application server, thirteen-entity relational model, priority-ordered safety enforcement, multi-user preference reconciliation and an Android client, demonstrating that the learning component can be deployed without bypassing operational safety.
- A Bayesian hyperparameter optimisation procedure over nine interdependent PPO hyperparameters, executed on a surrogate environment rather than on the live system.
- An empirical evaluation over a full simulated year against three feedback regimes, a rule-based baseline and a Deep Q-Network (DQN) variant, quantifying both energy savings and their effect on comfort.

The remainder of the paper is organised as follows. Section 2 reviews related work in IoT-based energy management and in smart home recommender systems, and states the gaps that motivate this study. Section 3 describes the methods: the system architecture, the recommender design and the experimental protocol. Section 4 reports the results, and discusses them and states the limitations of the study, and Section 5 concludes.

2 Related work

2.1 IoT applications in smart home energy management

Ravichandra et al. [6] integrate IoT sensing with machine learning, cloud computing and on-site renewable generation to predict consumption patterns and adjust settings automatically. The reported outcomes are substantial with a 20–30% reduction in overall energy use, 25% savings on HVAC, up to 35% on smart lighting and 15–20% on maintenance costs, and the work is notable for demonstrating that these technologies can be composed into a single coherent deployment. Two limitations bear on the present work. First, the optimisation is formulated purely in terms of energy, with occupant comfort absent from the objective, so the reported savings carry no accompanying evidence that the resulting behaviour would be tolerated in a real household. Second, the treatment of reinforcement learning is shallow relative to the problem's sequential structure.

Geetha et al. [7] develop a home energy management system that combines big data processing with machine learning to schedule consumption, with particular emphasis on storage and local production. Its principal strength is self-regulation: the system accommodates fluctuations in tariffs and in household demand without manual intervention, and it produces an optimal energy mix across production, storage and efficiency. Its principal weaknesses are practical. Device management and interoperability across a heterogeneous appliance population represent a significant integration burden, and the capital cost of the required infrastructure constrains adoption, which motivates approaches that operate on the existing device population without additional hardware.

The field has moved quickly since these studies. Pinthurat et al. [10] survey reinforcement learning for smart home energy management and report that single-agent and multi-agent formulations now dominate the literature, while identifying the validation of learned policies outside simulation as an outstanding gap. Their assessment is consistent with the pattern visible across the works reviewed here: the algorithmic machinery has matured considerably faster than the evidence that it survives contact with real occupants.

2.2 Recommender systems for smart homes

Recommender systems originate in e-commerce and media, where content-based and collaborative filtering methods dominate [9]. Their extension to smart environments requires a shift towards context awareness, in which time, location, ambient conditions and current activity enter the decision directly [11]. Boaventura et al. [12] illustrate this direction with a state recommender for actuator devices that combines deep reinforcement learning with implicit feedback.

Zharova et al. [13] propose an explainable multi-agent recommender for appliance load management. Multiple agents improve prediction accuracy and the system enriches its predictions with real-world signals such as live weather data, while explanations make the recommendations legible to occupants. Two limitations reduce its applicability. The system is trained on the REFIT dataset, collected between 2013 and

2015, a period that predates significant shifts in both appliance efficiency and household consumption patterns. More importantly, a fixed confidence threshold governs whether a recommendation is issued at all: when the predicted probability of device usage falls below it, no recommendation is made and the device is left unmanaged. These dead zones directly erode the achievable savings.

Sobhy et al. [14] address the multi-user case, which most work sets aside. Their context-aware system distributes the recommendation computation across nodes rather than centralising it, allowing tasks to execute concurrently and reducing response latency; parallel execution of the Apriori and association-rule algorithms sustains this advantage as the dataset grows. Conflicting preferences among co-located occupants are resolved through a priority-based strategy supported by unsupervised learning. The evaluation, however, rests on simulated data, and the simulation tool used (OpenSHS) was designed for the single-user case, obliging the authors to inject conflicts into the datasets manually. This introduces an artificial bias whose effect on the reported reliability is difficult to bound.

Eirinaki et al. [15] target real-time recommendations for appliance usage, applying energy disaggregation to the UK-DALE dataset in order to attribute total household consumption to individual appliances and to detect inefficient usage episodes. The use of non-intrusive load monitoring is an important practical advantage: a single main meter suffices, avoiding both the cost of per-appliance smart plugs and the privacy concerns that a dense sensor deployment raises. By converting consumption data into immediate, personalised prompts rather than monthly summaries, the framework reports savings of 2–17%. Its weaknesses are precision and context. High recall is accompanied by lower precision, producing false alarms in which the system asks the occupant to switch off an appliance that is in active use. Furthermore, the framework relies on historical temporal patterns alone and does not incorporate weather, seasonality or the weekday/weekend distinction.

Riabchuk et al. [11] extend the multi-agent line of work with a utility-based context-aware recommender that issues load-shifting suggestions for household devices, modelling occupant preference from habitual usage patterns, availability and device operating cost, and reporting an explicit treatment of the cold-start period. Their utility formulation is instructive by contrast with the approach taken here. Preference is inferred from observed behaviour, which makes the system operable from historical data alone but leaves it unable to distinguish a habit an occupant would prefer to change from one they would defend. A recommender that learns from stated responses rather than revealed habit resolves that ambiguity, at the cost of requiring the occupant to answer. Table 1 summarises this comparison.

2.3 Feedback-driven and human-in-the-loop learning

A distinct line of work treats occupant response not as an evaluation criterion applied after the fact but as the training signal itself, and it has developed rapidly enough to warrant separate treatment.

Chen et al. [16] address personalised HVAC control through model-based offline reinforcement learning combined with meta-learning, training a thermal preference model that adapts to a new occupant from a small number of comfort responses. The

Table 1 Comparison of representative related works

Work	Strengths	Limitations
[6]	Practical integration of IoT, ML and cloud; quantified savings of 20–30% overall, 25% HVAC, up to 35% lighting	Objective ignores occupant comfort; limited depth in the use of reinforcement learning
[7]	Self-regulating adaptation to tariff and demand fluctuation; optimal mix across production, storage and efficiency	Device interoperability is a major integration barrier; high cost limits adoption
[13]	Multi-agent design improves accuracy; incorporates live weather data; explanations aid transparency	Trained on the 2013–2015 REFIT dataset; fixed confidence threshold creates unmanaged dead zones
[14]	Explicitly handles co-located users with conflicting preferences; parallel execution scales with dataset size	Relies on simulated data; conflicts injected manually into a single-user simulator, introducing bias
[15]	Non-intrusive load monitoring needs only a main meter; real-time actionable prompts yield 2–17% savings	Lower precision produces false alarms on appliances in use; ignores weather, season and weekday/weekend effects
[11]	Utility formulation models preference from usage, availability and cost; explicit cold-start treatment	Preference inferred from revealed habit only, so a habit the occupant would prefer to change is indistinguishable from one they would defend
[16]	Personalises to a new occupant from few comfort responses	Requires a pre-collected corpus to initialise the policy
[17]	Learns comfort preference from ongoing feedback with no predefined set-points; incorporates electricity price	Exercises direct control rather than issuing declinable recommendations, so occupant disagreement is not observable

work establishes that personalisation from sparse human feedback is achievable, and its offline formulation avoids the exploratory behaviour that makes on-policy learning uncomfortable to deploy in an occupied dwelling. The approach nonetheless depends on a pre-collected corpus for its initial policy, which is precisely the dependency that a rule-based prior removes.

Liang et al. [17] present the closest existing work to the system proposed in this paper. They introduce a human-in-the-loop HVAC management framework in which the controller continuously learns occupants' comfort preferences from their feedback, instead of relying on predefined temperature set-points or occupancy schedules. At the same time, the controller adapts its decisions to changing electricity prices. Their results show that this feedback-driven approach consistently outperforms traditional rule-based control and achieves performance close to that of optimisation methods with perfect knowledge of future conditions. Despite these similarities, the two systems differ in two important respects, which also highlight the advantages of the proposed approach in practical residential settings. First, Liang et al.'s controller directly operates the HVAC system, whereas our approach only generates recommendations that occupants are free to accept or reject. This makes our system inherently safer and more user-centric, since it avoids enforcing potentially undesirable actions and allows occupants to retain full control over their environment. It also reduces the risk associated with incorrect learning or transient model errors, as no irreversible control action is taken. Second, their optimisation objective explicitly accounts for electricity prices,

whereas the present work focuses solely on balancing comfort and energy efficiency. While this simplifies the decision problem, it also makes the system more broadly applicable in settings where real-time pricing information is unavailable or unreliable. Incorporating dynamic electricity pricing is identified in Section 5 as a natural direction for future work.

A more recent research direction employs large language models (LLMs) as interfaces or coordinators rather than as decision-making agents. Michelon et al. [18] use an LLM to collect household preferences through natural language interactions and translate them into a valid configuration for a conventional home energy management system. Their results show that combining reasoning with few-shot prompting significantly improves the accuracy of parameter extraction. This approach addresses an important practical challenge: non-technical occupants often struggle to express their preferences using the structured parameters expected by an energy scheduler. This work complements rather than competes with the approach proposed in this paper. An LLM is well suited to capturing a household's initial preferences during system configuration. However, adapting to evolving preferences over time requires continuous, low-effort feedback, which is more naturally provided through simple binary responses to concrete recommendations. Moreover, by learning a compact policy locally on the household's own server, the proposed system avoids transmitting detailed behavioural data to an externally hosted language model, thereby improving privacy.

2.4 Research gap addressed by this work

Three main gaps emerge from this review. First, energy efficiency and occupant comfort are rarely optimised together. In most existing studies, comfort is treated as a constraint rather than being directly included in the optimisation objective. Second, many approaches have limited ability to adapt to changes in occupant behaviour. The policy is often fixed after training or learned offline from data collected from other households. Third, personalisation remains limited, as recommendations are rarely adapted to the individual preferences of each household.

The feedback-driven approaches discussed in Section 2.3 address parts of the second and third gaps. The system proposed in this work builds on this line of research rather than introducing a completely separate approach. However, an important combination remains insufficiently addressed: a policy that learns directly from occupants' responses, can be used from the first interaction without requiring a pre-existing offline training dataset, and provides recommendations that occupants can accept or reject rather than automatically controlling the environment. In this way, disagreement from the occupant becomes explicit feedback that the system can learn from.

The proposed system addresses these three gaps through a feedback-driven personalised learning process. Comfort is incorporated directly into the reward through the occupant's acceptance or rejection of each recommendation. As a result, the policy cannot improve its performance simply by reducing energy consumption at the expense of occupant satisfaction. Adaptation is continuous: each interaction is added to the experience buffer, and the policy is updated once a predefined amount of new experience has been collected. Finally, personalisation is built into the architecture, since

each household has its own agent whose policy is learned exclusively from that household's feedback. This allows the system to progressively adapt its recommendations to the preferences and behaviour of the individual household.

3 Proposed approach

This section is organised in three parts. Sections 3.1–3.5 describe the deployed system: its overall architecture, the layered application server, the relational data model, the safety and preference-reconciliation logic, and the mobile client and its interfaces. Sections 3.6–3.10 describe the recommender itself: the rationale for pairing a rule engine with a learned policy, the rule engine, the PPO agent and its training procedure, the hyperparameter optimisation framework, and the structure of the module's output. Sections 3.11–3.14 describe how the system was evaluated: the implementation environment, the generation of the behavioural dataset, the simulation protocol, and the feedback regimes and baselines against which the system was compared.

3.1 System overview

The proposed system is organised as an instance of the four-layer IoT model, comprising a perception layer of sensors and smart devices, a network layer carrying their measurements, a processing layer hosting persistence and intelligence, and an application layer exposing the result to occupants [2]. Figure 1 shows the resulting arrangement.

Data enters from two sources: IoT devices and sensors reporting device state and usage behaviour, and an external meteorological service supplying ambient conditions. During development and evaluation the device stream is reproduced virtually using the Home Assistant platform, which allows the full pipeline to be exercised without a physical installation while preserving the parameter structure of a real deployment.

The occupant configures the dwelling through the mobile client: rooms, devices, room assignments and household membership, together with explicit device preferences. This configuration and the incoming telemetry flow into the application server, which labels and persists them while materialising the relationships between entities (devices to rooms, rooms to dwellings, occupants to rooms).

The recommender module queries a restricted set of tables rather than the full database, retrieving only environmental readings, device action history and user preferences. It computes a recommendation and returns it to the server, which validates it against the safety rule set before delivery. The occupant's response closes the loop, supplying the reward that drives the next policy update.

3.2 Server infrastructure

The server is a layered, modular application built on FastAPI, chosen for its request throughput and for the structural conventions it imposes on code organisation. Persistence is provided by PostgreSQL with SQLAlchemy as the object-relational mapping layer, allowing entity relationships to be traversed in object-oriented terms rather than through raw SQL. Six layers compose the architecture (Figure 2).

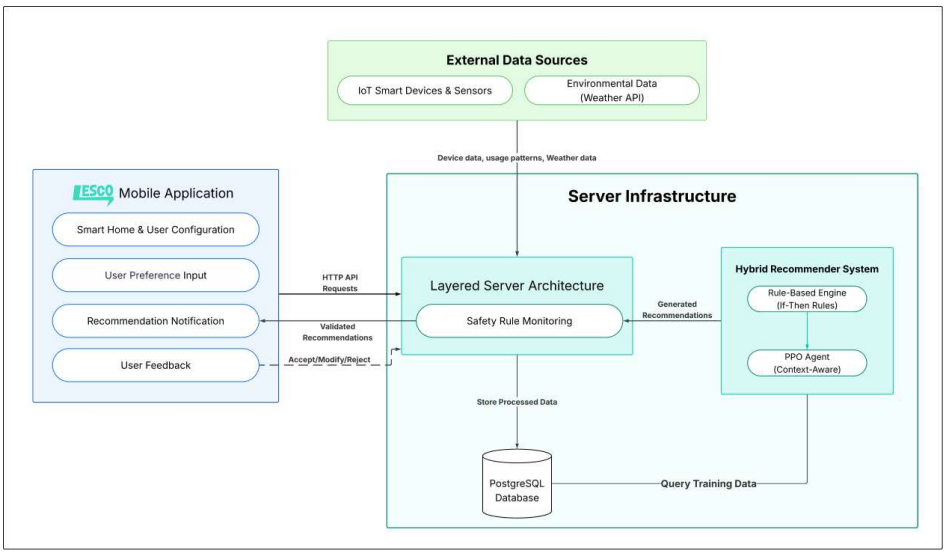


Fig. 1 General system architecture

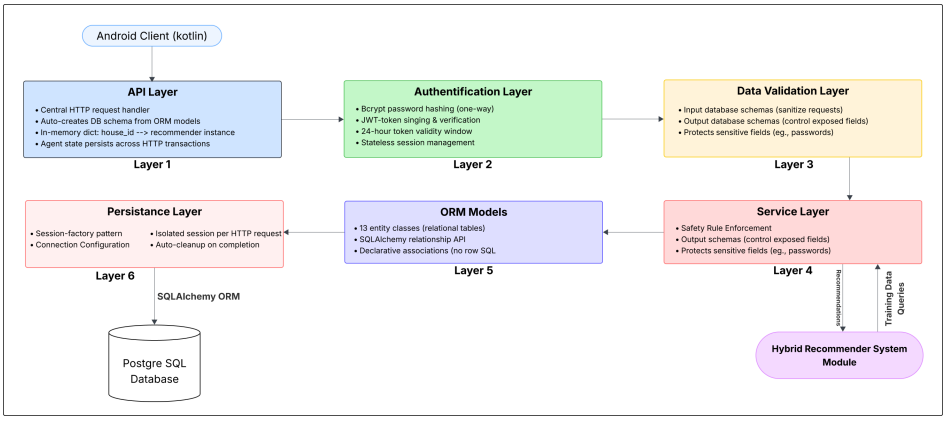


Fig. 2 The six layers of the server architecture and their relationship to the client, the relational store and the recommender module.

3.2.1 API layer

The API layer serves as the entry point for all incoming HTTP requests. During initialisation, the framework automatically derives the relational database schema from the declared ORM models, eliminating the need for manual schema creation during development. A key design decision at this layer is the use of an in-memory dictionary that maps each household identifier to its corresponding recommender instance. This allows the learning state of each agent to persist across multiple HTTP requests without requiring the model to be serialised and stored in the database after every

interaction. As a result, the agent can continuously update its policy as new feedback is received while keeping the database focused on persistent application data.

3.2.2 Persistence layer

The persistence layer manages communication with the database. Database connection parameters are loaded from environment variables at runtime, keeping configuration separate from the application code. A session factory is used to create a separate database session for each request. These sessions are managed using Python's context management mechanism, which ensures that each session is properly closed when the request finishes, including when an error occurs. This approach prevents database connection leaks and helps maintain transactional integrity by ensuring that each request operates within a well-defined database session.

3.2.3 ORM models

Thirteen entity classes correspond directly to the relational tables. All inter-entity relationships are expressed declaratively through the SQLAlchemy relationship API.

3.2.4 Data validation layer

Pydantic schemas serve a dual validation and serialisation role. Each entity carries at least two schemas: an input schema validating and sanitising client data, and an output schema controlling which fields the API exposes. The separation prevents inadvertent exposure of sensitive fields such as password hashes.

3.2.5 Authentication layer

Passwords are never stored in plaintext; bcrypt applies a one-way transformation resistant to brute-force recovery. Sessions are managed through JSON Web Tokens, in which the server signs a token carrying the caller's identity and an expiry, decoded and verified on each protected request, with a default validity window of twenty-four hours.

3.2.6 Service layer

The service layer holds the domain logic sitting above data access and below routing. It comprises safety rule enforcement, multi-user conflict resolution and weather integration, described in Section 3.4.

3.3 Data model

The conceptual-then-logical modelling methodology yields the relational schema below, in which starred attributes denote foreign keys.

- `user`(id, name, email, password)
- `house`(id, name, inviteCode)
- `membership`(id, role, userId*, houseId*)
- `room`(id, name, houseId*, roomType)

- `roomAssignment`(id, userId*, roomId*, role)
- `smartDevices`(id, name, deviceType, status, roomId*, batteryLevel)
- `deviceActionHistory`(id, deviceId*, actionType, newValue, origin, timeStamp)
- `environment`(temperature, weather, timeStamp)
- `userPreferences`(id, userId*, houseId*, category, value, time, context)
- `preferenceResolver`(strategyType, deviceType)
- `rule`(id, condition, action, priority)
- `recommendation`(id, type, content, confidenceScore, createdAt, houseId*, deviceId*)
- `userFeedback`(id, response, timeStamp, recommendationId*, userId*)
- `notification`(id, message, type, status, userId*, houseId*, recommendationId*)

3.4 Safety enforcement and multi-user reconciliation

3.4.1 Safety rules

A priority-ordered constraint checker is applied to every device control operation. When the agent proposes an action, the system first consults a dedicated rules table, where constraints are ordered according to their precedence. The proposed action is then checked against these rules to determine whether it would violate any established constraint. For example, the system can prevent the simultaneous activation of heating and cooling appliances in the same room, which would waste energy and could create hardware conflicts. If a violation is detected, the operation is rejected and the system returns a descriptive error explaining the reason for the rejection. Because the constraint check is performed on the server side rather than in the client application, modifying the client cannot be used to bypass these safety rules.

3.4.2 Preference reconciliation

Where several occupants hold divergent preferences for the same device, the resolver applies a strategy conditioned on the parameter type. For continuous parameters such as temperature it computes a weighted average of occupant preferences, assigning double weight to the household owner relative to other members and thereby encoding a practical hierarchy of authority. For binary parameters such as a light's on/off state it applies majority voting, with ties resolved in favour of the active state, which is the conservative choice from a comfort perspective.

3.4.3 Weather integration

The system retrieves current ambient weather conditions from the OpenWeatherMap service and includes them in the observation vector used by the recommender. These conditions provide the agent with information about the external environment when generating recommendations. If the weather service is temporarily unavailable or fails to respond, the system automatically uses predefined default values. This fallback mechanism allows the recommendation process to continue without interruption, even when external weather data cannot be retrieved.

3.5 Mobile client

3.5.1 Architectural role

The Android client, written in Kotlin, communicates with the server exclusively over the HTTP API. It maintains no local copy of device state or of the model; all state is authoritative on the server and retrieved on demand, and each user action follows a request–response cycle. The client therefore serves only as the interface through which users view system information and provide their responses, with authentication, safety enforcement, conflict resolution and learning delegated to the server. Beyond preventing circumvention of the safety and learning components, this separation allows additional client platforms to be built against the same API contract without duplicating logic.

The client plays a more important role in this system than a typical presentation layer. This is because the agent learns exclusively from the occupants' responses to its recommendations. The interface is therefore not only used to display the system's state; it is the main channel through which the agent receives feedback and generates its reward signal. If a recommendation is not noticed, or if responding to it is difficult or time-consuming, the agent receives no useful feedback and cannot learn from that interaction. For this reason, the interface was designed so that each recommendation leads to a clear binary choice, such as Yes/No or Accept/Reject. The response also requires minimal effort, encouraging occupants to provide feedback rather than ignore or dismiss the recommendation.

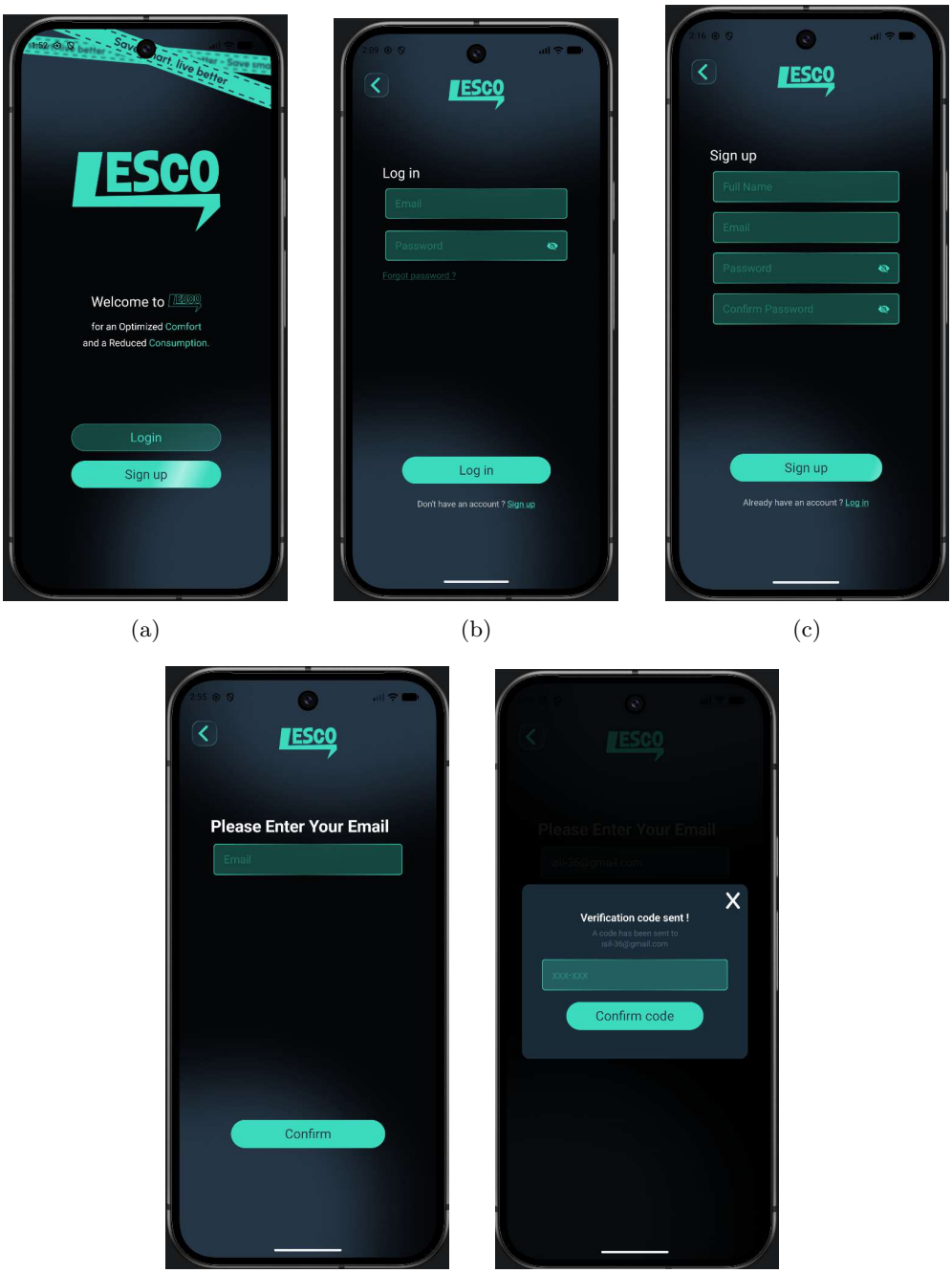
The Android client is built using Jetpack Compose and Material 3 to create the user interface. AndroidX Navigation Compose is used to manage movement between screens, while Gson converts the data received from the server into Kotlin objects that the application can use. AndroidX Lifecycle manages the application's state as the activity is created, paused, resumed, or destroyed. The application consists of four main interfaces, which are described below.

3.5.2 Authentication

The authentication interface is the entry point (Figure 3). It provides account creation and login, together with an email-based recovery flow in which a verification code is issued and confirmed before a password may be reset. Credentials are transmitted to the server, which performs bcrypt verification and returns a JSON Web Token; the client stores this token and attaches it to subsequent requests. No credential material is retained on the device beyond the token itself, and the token's twenty-four-hour validity window bounds the exposure of a lost or stolen handset.

3.5.3 Dwelling configuration

Before making any recommendation, the system first needs a description of the dwelling and its devices. The configuration interface (Figure 4) allows the occupant to create the rooms of the dwelling and add devices to each room. The supported device categories include lighting, air conditioning, heating, television, refrigeration,



(a) (b) (c)

(d) (e)

Fig. 3 Authentication interface: (a) welcome screen; (b) login; (c) registration; (d) email entry for password recovery; (e) verification code dialogue

dishwashing and floor lamps. For each room and device, the system creates the corresponding `room` and `smartDevices` records and links each device to its room. This organization allows the recommender to consider occupancy and preferences at the room level rather than treating the entire dwelling as a single space.

The system also supports multiple occupants. An existing occupant can invite another person using an invitation code stored in the `house` record. When the invitation is accepted, the new occupant is added to the existing dwelling through a `membership` record instead of creating a separate house. The occupant can then be associated with specific rooms and define individual preferences for the devices available in those rooms.

This multi-user configuration is essential for the conflict-resolution mechanism described in Section 3.4. A conflict occurs when multiple occupants have different preferences for the same device. Therefore, the system must first allow several occupants to define their preferences before the conflict-resolution mechanism can determine a suitable compromise.

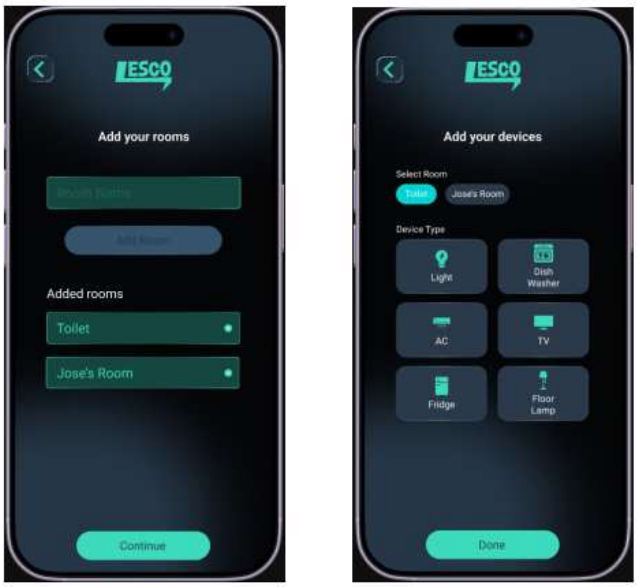


Fig. 4 Dwelling configuration interface: room declaration and per-room device assignment.

3.5.4 Dashboard

The dashboard is the main interface of the application (Figure 5). It provides a quick overview of the current state of the dwelling. At the top, a header displays the current ambient conditions, including indoor temperature, humidity and occupancy. Below this information, the connected devices are organized by room. Each device displays its current state and provides a direct toggle for turning it on or off.

A navigation drawer gives access to additional functions such as notifications and settings. When an occupant selects a specific device, the application opens a detailed view where its parameters can be adjusted and its previous actions can be examined.

The device history is particularly important because it is designed not only for control, but also to explain the system's actions. Every change in a device's state is stored in `deviceActionHistory`. The `origin` field indicates whether the action was performed manually by an occupant or automatically by the agent. Displaying this information allows occupants to understand whether a device was changed by themselves or by the system.

This transparency is important in an autonomous home-energy system. When the system can change the state of household devices without direct user intervention, occupants need to understand what happened and who initiated the action. Being able to answer a simple question such as "why is this device on?" helps build trust and makes the system easier to use over time.

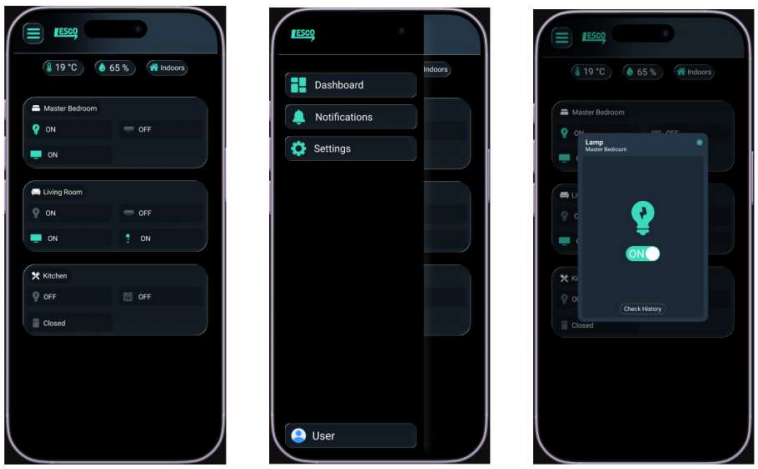


Fig. 5 Dashboard interface: room-grouped device overview with ambient conditions, the navigation drawer, and the per-device detail view exposing controls and action history.

3.5.5 Recommendation delivery and feedback capture

The recommendation interface completes the learning loop of the system (Figure 6). When the recommender generates a suggestion and it passes the safety validation stage, the server creates a `notification` record and the application displays a modal dialogue on top of the dashboard. The dialogue clearly presents three pieces of information: the condition detected by the system, the recommended action and the reason for the recommendation. It then gives the occupant two choices: accept or refuse the recommendation.

This design is directly related to the reward mechanism described in Section 3.8.5. The agent receives a reward of +1.0 when the occupant accepts a recommendation and

−0.5 when it is refused. If the occupant ignores the recommendation, no reward is generated. Therefore, an ignored recommendation provides no information for updating the policy while still creating an interaction cost.

For this reason, the recommendation is presented as a modal dialogue with two clearly labelled buttons rather than as a passive notification in a list. This makes the occupant's response explicit and reduces the number of cases where a recommendation is simply ignored. The selected response is sent to the server, which stores it in a `userFeedback` record. The corresponding reward is then added to the agent's experience buffer. Once the buffer reaches the required threshold, the system triggers a policy update using the collected feedback.

The interface also displays the reason provided by the recommender together with the suggested action. This improves transparency by allowing the occupant to understand why the recommendation was made. It also supports the learning process: when users understand the reason behind a recommendation, their response is more informed, which can provide a clearer and less noisy feedback signal to the agent.

3.6 Rationale for the rule-guided design

A reinforcement learning agent initialised at random produces arbitrary recommendations until sufficient experience accumulates. In a domestic setting this cold-start period is not merely inefficient but corrosive: occupants who receive nonsensical suggestions in the first days of operation disable the system before it can learn. The hybrid design addresses this directly. A deterministic rule engine encodes baseline energy-efficient behaviour for each device category, and its output is supplied to the agent as an observation feature. The agent therefore begins from sensible behaviour and shifts progressively towards its learned policy as feedback accumulates, without the transition requiring an explicit schedule. Figure 7 shows the module's control flow.

3.7 Rule engine

The rule engine applies predefined if-then conditions to the current state of the dwelling. These rules provide a simple and predictable way to determine the appropriate action for each device.

At night, defined as before 06:00 or after 22:00, the system turns off the lights and entertainment devices and sets the HVAC system to a setback temperature. When a room is unoccupied, all non-essential devices in that room are turned off. When a room is occupied, the action depends on the type of device. Lights and televisions respond directly to occupancy. Air conditioning is activated when the ambient temperature exceeds an upper threshold and deactivated when it falls below a lower threshold, with a comfort target maintained between the two thresholds. Heating follows the opposite logic.

The output of these rules has two purposes. First, it provides a fallback recommendation that can be used when the learning-based agent cannot make a suitable decision. Second, the resulting rule-based decision is included as the eighth observation feature used by the learning agent, as described in the following section.



Fig. 6 Recommendation delivery: a modal dialogue reporting the detected condition, the recommended action and the agent's stated reason, terminating in the accept and refuse controls that generate the reward signal.

3.8 PPO agent

The adaptive component is a PPO agent [8] operating under a human-in-the-loop paradigm. Its policy is not fitted to a static corpus but refined continuously from interaction with occupants, reflecting the fact that domestic preferences are personal, time-varying and not transferable between households.

3.8.1 Network architecture

Policy and value functions share a single network of actor–critic design. A nine-dimensional observation vector passes through two shared fully connected layers with hyperbolic tangent activations. The tanh nonlinearity was selected for its bounded range $(-1, 1)$, which promotes gradient stability in deeper layers and performs well on continuous control tasks of the kind considered here.

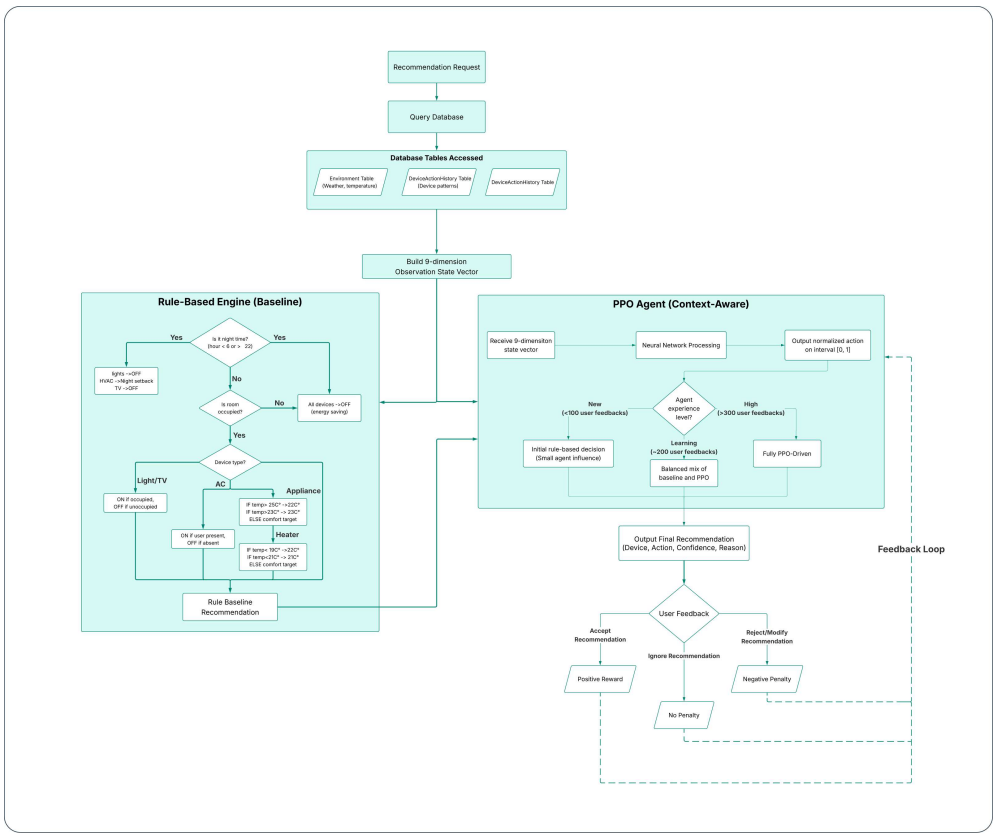


Fig. 7 Control flow of the hybrid recommender module, from database query and state construction through the parallel rule engine and PPO agent to the final recommendation and the feedback loop.

The shared representation then splits. The actor head produces a scalar passed through a sigmoid $\sigma(\cdot)$, compressing it to the unit interval. All device parameters are normalised to $[0, 1]$, and the network output is subsequently de-normalised to the appropriate physical scale, for example $[16\,^{\circ}\mathrm{C}, 28\,^{\circ}\mathrm{C}]$ for air conditioning. A learnable log-standard-deviation parameter is maintained alongside, so that the policy expresses a Gaussian distribution over actions rather than a point estimate, which is necessary for exploration. The critic head produces a scalar estimate of the expected cumulative return from the current state, used to compute the advantages that guide the policy update.

3.8.2 Observation vector

Table 2 details the nine features presented to the network at each decision step. The inclusion of the rule baseline as element eight is the mechanism through which the hybrid design operates: the agent can learn to confirm the rule engine in familiar

contexts and to deviate from it where feedback has consistently indicated a different preference.

Table 2 Composition of the nine-dimensional observation vector

Index	Feature	Description
0	Hour of day	Current hour normalised to $[0, 1]$
1	Day of week	Current weekday normalised to $[0, 1]$
2	Weekend flag	Binary indicator of a weekend day
3	Night flag	Binary indicator of night-time (before 06:00 or after 22:00)
4	Outdoor temperature	Ambient temperature normalised by a reference maximum of 40 °C
5	Rain flag	Binary precipitation indicator from the weather service
6	Device status	Current on/off state of the target device
7	Device parameter	Current normalised parameter value of the target device
8	Rule baseline	Normalised output of the deterministic rule engine for the current device and context

3.8.3 Advantage estimation

The agent uses Generalized Advantage Estimation to quantify how much better a particular action is than the average action the current policy would take in the same state. For each timestep t in the experience buffer the temporal-difference residual is

$$\delta_t = r_t + \gamma\, V(s_{t+1})\,(1 - d_t) - V(s_t), \tag{1}$$

where r_t is the received reward, γ the discount factor governing the preference for immediate over future reward, $V(s_t)$ the critic's value estimate and d_t the done flag masking the bootstrap at episode boundaries. Advantages accumulate in reverse temporal order with exponential decay controlled by λ:

$$A_t = \delta_t + \gamma\,\lambda\, A_{t+1}. \tag{2}$$

The parameter λ interpolates between a one-step temporal-difference estimate ($\lambda = 0$, low variance and high bias) and a full Monte Carlo return ($\lambda = 1$, low bias and high variance). Distinct values are used for HVAC and lighting categories, reflecting the different temporal horizons over which comfort preferences manifest for these device classes. Advantages are normalised before use in the policy update, which empirically stabilises training by preventing a single estimate from dominating the gradient.

3.8.4 Policy optimisation

Writing the probability ratio between the updated and the data-generating policy as

$$\rho_t = \frac{\pi_{\text{new}}(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}, \tag{3}$$

the clipped surrogate objective is

$$L^{\text{CLIP}} = \min(\rho_t A_t,\ \text{clip}(\rho_t, 1-\varepsilon, 1+\varepsilon)\, A_t)\,. \tag{4}$$

Clipping ensures that the gradient never displaces the policy more than a factor ε from its previous parameterisation in a single update, which is the mechanism preventing destructive policy collapse. The total objective combines three terms,

$$L^{\text{total}} = -L^{\text{CLIP}} + c_1 L^{\text{VALUE}} - c_2 L^{\text{ENTROPY}}, \tag{5}$$

where L^{VALUE} is the mean squared error between the critic's estimate and the computed return, and L^{ENTROPY} is the entropy of the action distribution, entering as a regularisation bonus that discourages premature convergence to a deterministic policy and preserves exploration. The coefficients c_1 and c_2 balance the three objectives.

Once the experience buffer accumulates a sufficient number of interactions, the network is updated over several epochs on the collected data, reshuffled into mini-batches at each epoch to reduce correlation between gradient steps. Gradient norms are clipped to a maximum threshold to prevent the exploding-gradient pathology.

3.8.5 Feedback and reward

Learning is driven by the reward set out in Table 5, which combines two terms scored directly by the simulation environment with a third derived from the occupant's response.

The comfort term measures the proximity of the recommended action to the context-dependent ideal for the device in question, and the energy term applies a penalty scaled by the power the recommended action would draw, at a fifth of the weight of comfort. Together these express the trade-off the system exists to manage, and both are computable because the simulation holds the ideal action for a given context and the power characteristics of each device.

The response term captures the occupant's own judgement, which the first two terms cannot supply. When a recommendation is served, the system records the state vector, the proposed action and the log-probability of that action under the current policy. Acceptance contributes $+1.0$, refusal -1.0, and modification of the proposed value -0.5, the smaller magnitude reflecting that a modified recommendation was directionally useful even though its specific value was not. Once the experience buffer reaches its minimum threshold a full update is triggered and the buffer cleared for the next collection cycle, allowing the agent to adapt continuously to evolving household preferences.

The same reward is used to score candidate hyperparameter configurations during the search described in Section 4.1, so that the configuration selected there is optimal with respect to the objective the agent actually optimises.

3.9 Bayesian hyperparameter optimisation

The nine PPO hyperparameters influence each other, so their combined effect can significantly affect learning speed, training stability, and final performance. Tuning them manually would be slow and would make it difficult to find a good configuration in such a large nine-dimensional search space.

To automate this process, the system uses Bayesian optimisation with a Tree-structured Parzen Estimator (TPE). TPE learns from previous trials which hyperparameter combinations tend to produce better results and uses this information to choose more promising configurations for the next trials. However, performing this optimisation directly on the live application would be too expensive. Each trial requires training and evaluating a separate agent, which would involve the database, server, and real occupants. Instead, the tuner uses a lightweight surrogate environment that reproduces the same inputs and reward mechanism as the real system, but without requiring the live infrastructure or real occupants. For each trial, the tuner:

- Selects a candidate set of PPO hyperparameters.
- Creates a new agent with this configuration.
- Trains the agent for a fixed number of episodes.
- Calculates its performance using the mean cumulative reward over the last 20 episodes.
- Gives this score to TPE, which uses it to select the next configuration.

A median pruner further reduces computation. It monitors trials during training and stops a trial early if its intermediate performance is worse than the median performance of completed trials. This prevents time from being wasted on configurations that are unlikely to become competitive. After the search finishes, the best-performing configuration is saved in a structured configuration file and then applied to the system's central configuration.

3.10 Recommender output structure

To integrate cleanly with the remainder of the server, the module returns a structured JSON object whose fields are given in Table 3.

A representative output is the following:

```
{
  "recommendation": "Set Living Room AC temperature to 22 C",
  "action": 22,
  "confidence": 0.71,
  "reason": "PPO adjusted from rule baseline (cooling_required)"
}
```

Table 3 Structure of the recommender module output

Field	Description
`recommendation`	String presented to the occupant, including its explanation
`action`	Integer encoding either a binary on/off decision or a proposed device parameter value; device actions are distinguished by predefined value ranges
`confidence`	Floating-point score expressing the system's confidence in the relevance of the recommendation
`reason`	String describing the reasoning behind the recommendation

3.11 Implementation environment

The remainder of this section focuses on the implementation rather than the system design. Since no physical smart-home installation was available, the evaluation uses a behavioural dataset specifically generated for this study. The dataset is replayed through the deployed server and recommender, ensuring that the components being evaluated are the same ones described in the previous sections.

The agent was trained on a workstation running Windows 11 (build 26200.8246, x64). The machine is equipped with a 14-core Intel Core i7-12700H processor running at up to 4.70 GHz, 32 GB of DDR4-3200 RAM, an NVIDIA GeForce RTX 3070 Ti laptop GPU with 8 GB of VRAM, and a 2 TB PCIe Gen3 NVMe SSD.

The server and learning components were implemented in Python. The software stack is divided into two main parts:

- **Server and data management:** FastAPI provides the REST API, PostgreSQL stores the relational data, SQLAlchemy handles database access through an ORM, Pydantic performs data validation and serialisation, Passlib handles password hashing, and Requests manages outbound HTTP communication.
- **Learning and evaluation:** PyTorch implements the neural networks, NumPy handles numerical array operations, Optuna performs Bayesian hyperparameter optimisation, Stable-Baselines3 provides the RL agents used during tuning, and Gymnasium provides the surrogate environment.

Finally, Home Assistant is used as the virtual smart-home environment. It provides a controlled environment for reproducing the required inputs and behaviours and helps ensure that the parameters used during evaluation remain consistent with those expected by the deployed system.

3.12 Dataset generation

Evaluating a system whose value lies in adaptation over months requires a behavioural record spanning at least a full annual cycle. In the absence of a deployed installation, we generated such a record synthetically, adopting the parameter structure of Home Assistant so that the data would remain compatible with a subsequent physical deployment. Each row of the resulting activity log carries a timestamp, the occupant,

the room, the activity, the device and the action, the last being either a binary state or a set-point value.

The generator simulates a family inhabiting a smart home at a thirty-minute resolution across the period from 1 January to 31 December 2025, modelling daily routines, movement between rooms and appliance interaction as functions of time of day and season. The dwelling comprises six rooms : living room, kitchen, master bedroom, child's bedroom, bathroom and hallway, each containing a specific device population. The household comprises three members (father, mother and child), each configured with base sleep and wake times and with working or school days.

Two modules are responsible for generating the behavioural dataset. The person behaviour engine assigns activities and room occupancy to each occupant. Fixed schedule events, such as waking and sleeping, are given a ±20-minute variation so that the generated schedules are not completely rigid. Because the simulation uses a thirty-minute timestep, this variation is effectively quantised: an event typically falls either in its nominal time slot or in one of the adjacent slots. The ±20-minute value was chosen to balance variability and routine structure. A smaller variation would leave most events in the same time slot and therefore preserve the rigidity that the variation is intended to avoid, whereas a substantially larger variation could spread an event across three or more time slots and make the underlying routine harder for the agent to learn.

The device engine then examines the devices available in each occupied room and simulates their use according to probability, occupant preferences, and season. Device usage is therefore not determined solely by occupancy. For example, lights are switched on with certainty at night but only with a 30% probability during daylight, while air-conditioning usage depends on the season and reaches a 90% probability in summer. For devices with adjustable parameters, the engine also considers individual occupant preferences. For instance, the father prefers an air-conditioning temperature of 21–23,° C, whereas the mother prefers 22–25,° C. These different preferences allow the system's multi-user preference reconciliation mechanism to be evaluated.

The generated behavioural data are finally written to CSV format, with Table 4 providing a representative example of the resulting records.

Table 4 Structure of the generated activity log

Timestamp	Person	Room	Activity	Device	Action
2025-01-01 18:00	Father	Living_Room	Indoors	Light	ON
2025-01-01 18:00	Father	Living_Room	Indoors	TV	ON
2025-01-01 18:00	Father	Living_Room	Indoors	Heater	28 °

3.13 Simulation protocol

The simulation first parses the historical dataset to extract the timestamp, occupant, room, activity, action, and value associated with each record. Each timestamp

is then enriched with weather information obtained from the meteorological service and with device energy-consumption values based on the default values provided by Home Assistant.

The actions are grouped into thirty-minute timesteps. When multiple actions affect the same device in the same room during a single timestep, the simulation retains the most energy-intensive action. This choice deliberately makes the baseline more demanding than the proposed system, resulting in a conservative estimate of the energy savings.

The simulation then processes each timestep sequentially. First, it constructs the observation vector from the available context. The agent then generates a recommendation, after which the active scenario determines whether the occupant accepts, rejects, or modifies the recommendation. The resulting action is converted into an estimated energy consumption in watt-hours (Wh). For binary devices, the simulation assumes that the device consumes its full rated power for the entire timestep when switched on and consumes no power when switched off. For continuous devices, such as air conditioning, consumption is calculated as a function of the device state and selected set-point. An additional always-on background load, representing appliances such as a refrigerator, is added to every timestep.

Finally, a confidence metric is calculated by comparing the system's recommendation with the corresponding historical usage recorded in the dataset. The simulation records the resulting daily energy consumption and comfort scores, which are then exported to CSV for further analysis

3.14 Scenarios and baselines

Three feedback regimes bracket the range of occupant behaviour. Under *reject all*, every recommendation is refused, which reproduces unoptimised consumption and establishes the upper bound. Under *accept all*, every recommendation is followed, establishing the lower bound achievable by the policy. Under *mixed*, acceptance is probabilistic and weighted by seasonal and temporal variables (occupants are, for instance, more likely to refuse energy-efficient air conditioning recommendations in summer) which is the regime intended to approximate real behaviour. Two additional configurations establish comparative context. A rule-based baseline replaces the hybrid recommender with the rule engine alone, and a DQN-based recommender replaces the PPO agent with a value-based alternative. Both are run under the mixed regime for comparability.

4 Results and discussion

4.1 Hyperparameter search

The surrogate environment uses the same nine-dimensional observation space as the deployed system, ensuring that the policies are evaluated under the same input conditions. It uses the reward weights defined in Table 5 to calculate the reward for each action. The reward combines three main factors: comfort, measured by how closely the

recommendation matches the ideal action for the current context; energy consumption, included as a penalty proportional to the estimated power draw; and occupant feedback, represented by separate terms for acceptance, rejection, and modification of the recommendation.

Table 5 Reward function weights used during hyperparameter tuning

Term	Weight	Description
Comfort	1.0	Proximity of the recommendation to the context-dependent ideal
Energy	0.2	Penalty scaled by power consumption, enforcing efficiency
Approval bonus	1.0	Reward for simulated occupant approval
Rejection penalty	1.0	Penalty for simulated occupant rejection
Modification penalty	0.5	Smaller penalty for simulated occupant modification

Table 6 reports the search ranges and the resulting PPO configuration. Several observations can be drawn from the optimisation results. First, the learning rate converged close to the upper bound of its search range, suggesting that the reward signal is sufficiently well-conditioned to support relatively rapid policy updates. Second, the discount factor converged towards the lower end of the range, reaching 0.906. This is consistent with the short-term nature of the problem, where the effects of a recommendation are generally observed over a limited time horizon rather than accumulating over long trajectories.

Finally, the optimisation selected the smallest available mini-batch size together with the maximum number of training epochs. This combination indicates a preference for performing many small and controlled gradient updates rather than a smaller number of large updates. Such a configuration is well suited to the application because a single household produces a relatively small volume of occupant feedback.

4.2 Feedback regimes

Table 7 and Figure 11 summarise the three regimes. Each fixes how the simulated occupants respond to the recommendations they are served, and together they bracket the range of behaviour a real household might exhibit. Under *reject all*, every recommendation is refused, so the agent's proposals never take effect and consumption follows the behavioural dataset unaltered; this establishes the upper bound and the reference against which savings elsewhere are measured. Under *accept all*, every recommendation is acted on, granting the policy effective autonomy over the device population and establishing the lower bound the policy can reach. Under *mixed*, acceptance is probabilistic and weighted by seasonal and temporal variables, so that occupants are more inclined to refuse an energy-saving proposal when the comfort cost is highest, as with air conditioning in summer; this is the regime intended to approximate real behaviour and the one from which the practically meaningful figures are drawn. The three are taken in turn below, beginning with the case in which no recommendation is acted on.

Table 6 PPO hyperparameter search space and optimised configuration

Hyperparameter	Description	Search space	Optimised
`lr`	Learning rate of the actor–critic network	$[10^{-5}, 10^{-3}]$ (log)	0.000999043
`gamma`	Discount factor for future rewards	$[0.9, 0.999]$	0.905614
`gae_lambda`	GAE smoothing factor	$[0.8, 1.0]$	0.815411
`clip_epsilon`	Clipping parameter limiting policy updates	$[0.1, 0.3]$	0.25328
`value_coef`	Weight of the value function loss	$[0.1, 1.0]$	0.483424
`entropy_coef`	Entropy bonus promoting exploration	$[0.001, 0.1]$ (log)	0.0110056
`max_grad_norm`	Gradient clipping threshold	$[0.3, 1.0]$	0.641441
`ppo_epochs`	Passes over the buffer per update	$[2, 10]$ (integer)	10
`batch_size`	Mini-batch size during optimisation	$\{32, 64, 128, 256\}$	32

4.2.1 Reject-all regime

With every recommendation refused, consumption remains uniformly high across the year, between approximately 64.8 and 69.1 kWh per month, and agent confidence fluctuates in a narrow band between 0.476 and 0.491.

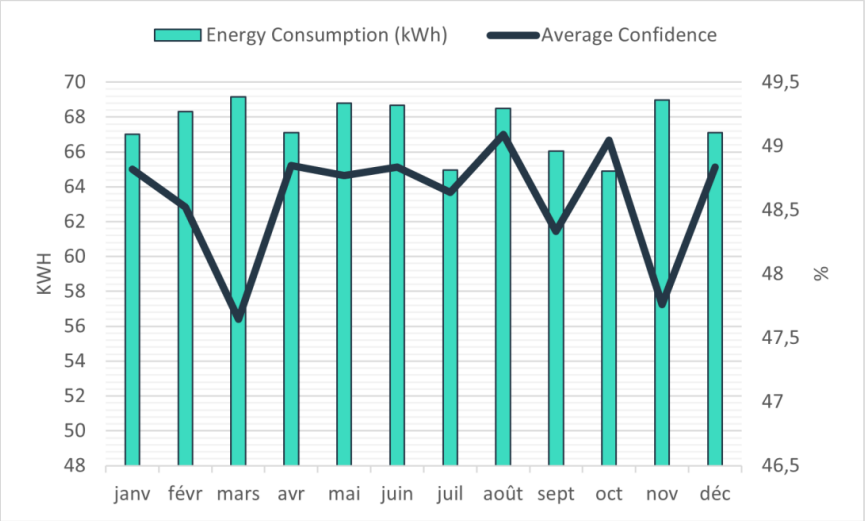


Fig. 8 Monthly energy consumption and mean agent confidence under the reject-all regime.

4.2.2 Mixed regime

Under selective acceptance, consumption varies considerably more, ranging from approximately 57.8 to 68.9 kWh per month, with the largest reductions concentrated in spring and autumn. Mean confidence stabilises between 0.730 and 0.749, consistent with a balanced feedback loop in which the policy receives both positive and negative signal.

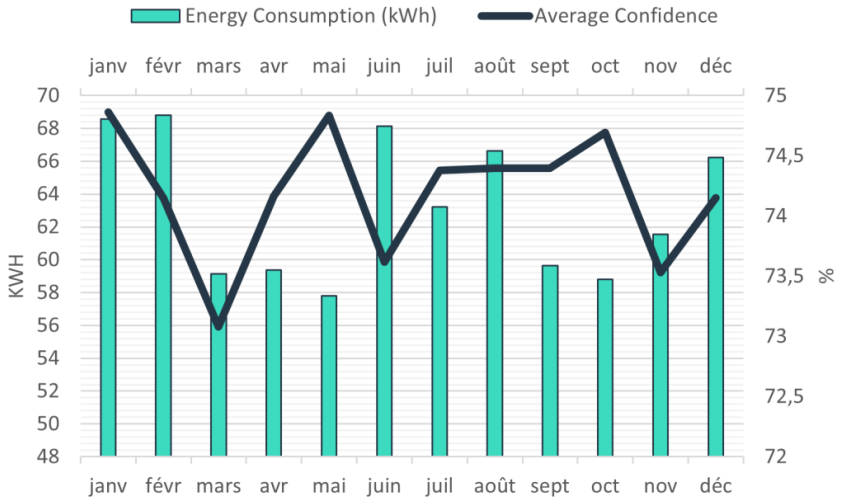


Fig. 9 Monthly energy consumption and mean agent confidence under the mixed regime.

4.2.3 Accept-all regime

With full acceptance, consumption falls to between 49.7 and 69.1 kWh per month and confidence approaches unity at approximately 0.9975, the expected consequence of uniformly positive reinforcement.

Table 7 Summary of results across the three feedback regimes

Regime	Monthly range (kWh)	Mean (kWh)	Mean confidence	Annual saving
Reject all	64.8–69.1	67.0	0.48	—
Mixed	57.8–68.9	62.7	0.74	6.4%
Accept all	49.7–69.1	57.5	0.9975	14.2%

Annual savings are computed relative to the reject-all regime

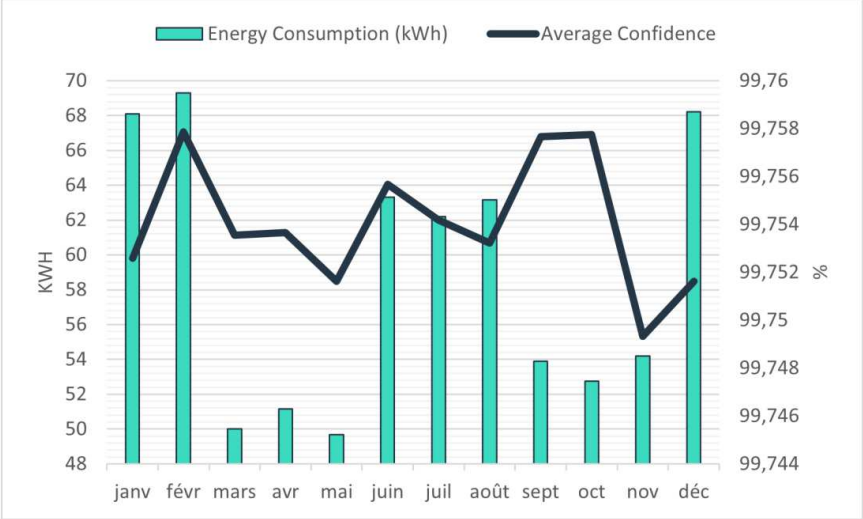


Fig. 10 Monthly energy consumption and mean agent confidence under the accept-all regime.

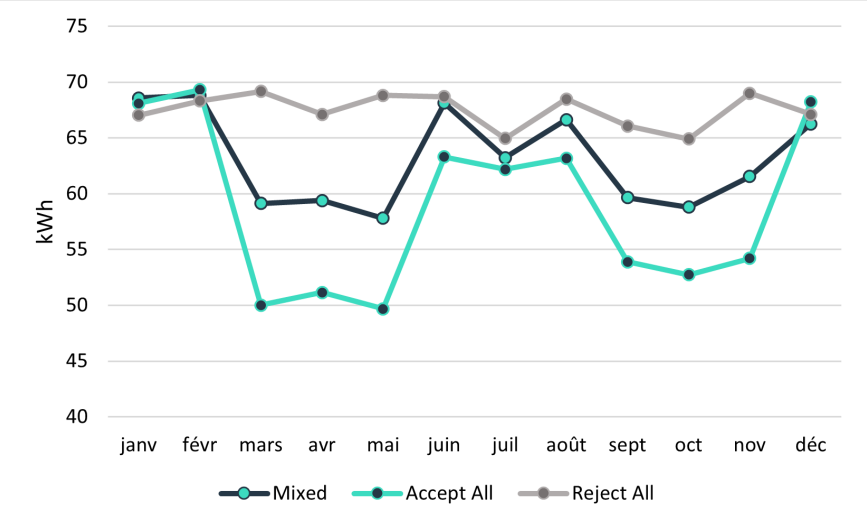


Fig. 11 Monthly energy consumption across the three feedback regimes

4.3 Comparison with rule-based and DQN recommenders

Applied under equivalent conditions, the three recommenders separate clearly. PPO ranges from roughly 58 to 71 kWh per month, DQN from approximately 63 to 71 kWh, and the rule-based baseline from roughly 63 to 91 kWh. PPO records the lowest consumption in all twelve months, with the widest margin over the baseline in spring and autumn and the widest margin over DQN in winter and summer.

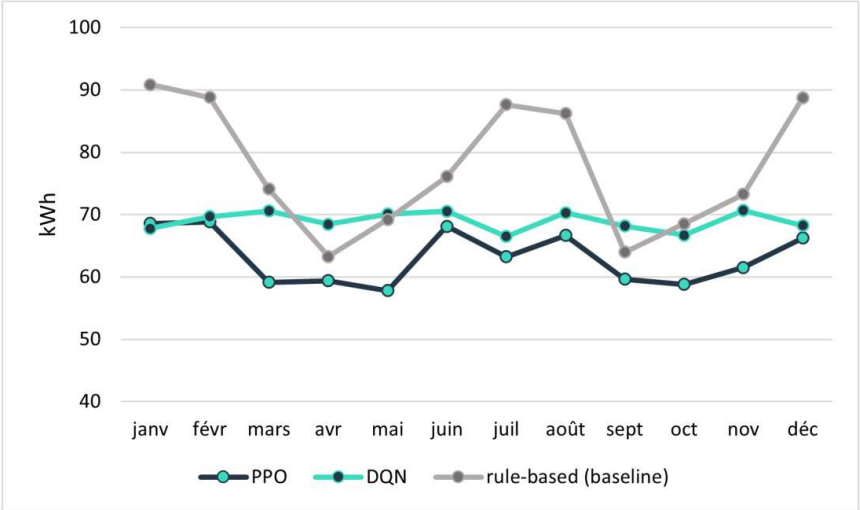


Fig. 12 Monthly energy consumption for the PPO agent, the DQN variant and the rule-based baseline under the mixed feedback regime.

4.4 Interpretation of the findings

The reject-all regime establishes that the savings observed elsewhere are attributable to the recommendations themselves rather than to the underlying occupancy pattern: with every suggestion refused, consumption shows no downward trend across the year, and the low confidence values reflect a policy receiving only negative reinforcement.

Under the mixed regime the seasonal distribution of the savings is the informative feature. Reductions cluster in the transitional seasons, precisely where thermal demand is genuinely discretionary and where a static schedule is least likely to be correct. This is the behaviour one would want from an adaptive system, and it is not a behaviour a fixed schedule can produce.

The accept-all regime speaks most directly to the comfort question, and there the shape of the curve matters more than its level. Consumption remains high in winter and summer even under full autonomy. Had the policy learned to minimise energy without regard to comfort, these months would show the largest reductions, since they carry the largest discretionary thermal load. That they do not indicates the policy has learned that heating and cooling demand in extreme months is not discretionary from the occupant's perspective, and concentrates its savings where they are.

The comparison with the alternative recommenders can be explained by the different ways they handle control decisions. PPO performs particularly better than the rule-based baseline in spring and autumn because the baseline relies on fixed temperature thresholds. During these transitional seasons, those thresholds can unnecessarily activate heating or cooling even when the ambient conditions do not require it. DQN also improves on the rule-based approach, but it consistently performs worse than PPO. This difference becomes more pronounced in winter and summer, when precise and continuous adjustment of the temperature set-point is more important. The main

limitation of DQN in this setting is that it is designed for discrete action spaces. Therefore, the continuous set-point range must be divided into a limited number of discrete actions, which introduces quantisation errors. These errors become more significant when small changes in the set-point can have a large impact on energy consumption or comfort. In contrast, PPO directly models a continuous Gaussian policy, allowing it to select set-points without discretising the action space. This provides finer control and explains why PPO is better suited to the continuous optimisation problem considered here.

4.5 Relation to prior work

The mixed regime is the most practically relevant case, since neither of the two extreme regimes represents how a real household is likely to operate. Under this regime, the system achieves a 6.4% annual energy saving, with savings reaching 15.6% in the most favourable periods. These results fall within the range reported for comparable recommendation-based systems [15]. Importantly, these savings are obtained without requiring additional hardware beyond the devices already available in the household and without relying on a pre-existing offline training dataset.

The comparison with feedback-driven control is less direct, but it provides a useful point of reference. Liang et al. [17] showed that a human-in-the-loop HVAC controller that learns from occupant feedback can outperform rule-based control and approach the performance of optimisation with perfect foresight. A similar pattern is observed in the present study, although in a different application setting. The advantage of the learned policy over the rule-based baseline is greatest during the transitional months, when fixed temperature thresholds are more likely to activate heating or cooling unnecessarily. The main difference between the two approaches lies in the role of the occupant. In Liang et al., the controller acts and the occupant experiences the resulting control action. In the present system, the controller instead makes a recommendation and the occupant explicitly accepts or rejects it. Consequently, the refusal rate becomes a measurable part of the control process. This also explains why the confidence trajectories discussed in Section 4 can be observed: disagreement is explicitly expressed through rejection rather than being hidden within the controller's actions.

The results can also be considered against the three research gaps identified in Section 2.4. The joint consideration of energy consumption and occupant comfort is the most directly supported contribution. Comfort is incorporated through the occupant's response, and the accept-all evaluation provides no indication that the reported energy savings are achieved by systematically sacrificing comfort. Continuous adaptation is also supported by the seasonal distribution of the savings, which shows that the learned policy changes its behaviour according to the conditions rather than following a fixed schedule. Personalisation, however, remains the least substantiated claim. Although each household is assigned its own agent and policy, the current evaluation involves only a single household. Therefore, it cannot yet establish whether independently learned policies would meaningfully diverge across different households or whether the observed policy would generalise to other occupant profiles.

Finally, unlike the systems proposed by Zharova et al. [13] and Eirinaki et al. [15], the present approach generates a recommendation at every decision step rather than

suppressing recommendations below a predefined confidence threshold. This design avoids the periods in which no recommendation is produced and therefore preserves the potential for additional energy savings. However, it also increases the number of recommendations presented to the occupant, which creates more opportunities for inappropriate or unnecessary suggestions. The resulting trade-off is therefore between maximising the number of optimisation opportunities and limiting the interaction burden imposed on the occupant.

4.6 Limitations

Three limitations should be considered when interpreting these results. First, the evaluation relies on generated rather than real-world household data. Although the data generator reproduces the parameter structure of Home Assistant and incorporates schedule variability, probabilistic device usage, and seasonal conditions, it cannot fully capture the irregular events that occur in real households, such as guests, illness, holidays, or appliance failures. The reported energy savings should therefore be interpreted as evidence of the behaviour of the proposed mechanism under controlled conditions, rather than as a direct prediction of its performance in real-world deployment. This limitation is also present in related studies that rely on simulated occupancy data [14].

Second, the occupant feedback used in the mixed regime is simulated through a weighted probabilistic model rather than collected from real users. While this model reproduces the expected seasonal variation in acceptance and refusal behaviour, it cannot capture the individual and unpredictable nature of human decisions. As a result, the simulated confidence trajectories are likely to be smoother and more regular than those that would emerge from real occupants, where preferences and responses may change abruptly.

Third, the evaluation considers only a single household configuration consisting of three occupants and six rooms. This limits the extent to which the results can be generalised to larger or more diverse households. As the number of occupants increases, conflicts between individual preferences are likely to become more frequent, while the feedback available for each individual device may become sparser. How these two effects interact with the rate at which the policy learns and converges remains an open question and requires evaluation across multiple household sizes and configurations.

5 Conclusion

This paper has presented a recommender system for smart home energy efficiency whose policy is learned online from occupant feedback alone, anchored during the cold-start period by a deterministic rule engine, embedded in a complete IoT architecture with server-side safety enforcement, multi-user preference reconciliation and an Android client. Injecting the rule engine's output into the agent's observation vector allows the system to behave sensibly from the first interaction while shifting progressively towards a learned, household-specific policy.

Evaluated over a simulated year, the system reduced energy consumption by 6.4% on average under a realistic mixed-feedback regime and by 14.2% under full acceptance, with monthly peaks of 15.6% and 27.8% respectively, while the seasonal distribution

of savings indicates that these reductions were not obtained at the expense of comfort in thermally demanding months. PPO outperformed both a DQN variant and a static rule-based baseline in every month of the simulated year, with the margin over DQN concentrated where continuous set-point control dominates.

The natural continuation of this work is deployment in a physical dwelling connected to real IoT devices and sensors. Field validation would test the assumptions embedded in the behavioural generator and would expose the policy to the irregular events that a synthetic record cannot supply. Beyond that, three directions appear productive: extending the device population to include water heating, ventilation and large appliances whose scheduling flexibility is greater than that of the devices considered here; characterising convergence behaviour across households of varying size and preference heterogeneity; and incorporating tariff structure into the reward so that the agent optimises cost as well as consumption, which would allow it to exploit time-of-use pricing without additional architectural change.

Declarations

Funding. The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing interests. The authors have no competing interests to declare that are relevant to the content of this article.

Availability of data and materials. The synthetic dataset generated and analysed during the current study, and the source code of the application server, the recommender module and the dataset generator, are available from the corresponding author on reasonable request.

Author contributions. **Youcef Guenfaf** conceived the study, designed the system architecture and the recommender approach, directed the research and wrote the manuscript; the work forms part of his PhD research on energy efficiency in IoT systems. **Youcef Zafoune** supervised the PhD work and reviewed the manuscript. **Amdjed Baha-eddine Chaabane** and **Abdnour Benmeridja** designed, implemented and tested the reinforcement learning agent. **DouniaZed Daouadji** implemented the Home Assistant virtual smart-home environment. **Dounia Fatma Zohra Rehab** and **Imad Yacine Karoum** designed and implemented the Android mobile application. All authors read and approved the final manuscript

References

[1] Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A survey. Computer Networks **54**(15), 2787–2805 (2010) https://doi.org/10.1016/j.comnet.2010.05.010

[2] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems **29**(7), 1645–1660 (2013) https://doi.org/10.1016/j.future.2013.01.010

[3] Aguru, A.D., Erukala, S.B., Sethi, A., Rajasekhar, P.: Integrated industrial reference architecture for smart healthcare in Internet of Things: A systematic investigation. Algorithms **15**(9), 309 (2022) https://doi.org/10.3390/a15090309

[4] Chakraborty, A., Islam, M., Shahriyar, F., Islam, S., Zaman, H.U., Hasan, M.: Smart home system: A comprehensive review. Journal of Electrical and Computer Engineering **2023**, 7616683 (2023) https://doi.org/10.1155/2023/7616683

[5] Al-Wreikat, Y., *et al.*: Design and implementation of a real-time smart home management system considering energy saving. Sustainability **14**(21), 13840 (2022) https://doi.org/10.3390/su142113840

[6] Ravichandra, T., *et al.*: IoT applications in smart building energy management. Materials Today: Proceedings **56**, 2329–2337 (2022) https://doi.org/10.1016/j.matpr.2021.11.416

[7] Geetha, B.T., *et al.*: HEMS-IoT: Smart home energy saving with big data and machine learning. Materials Today: Proceedings **45**, 428–432 (2021) https://doi.org/10.1016/j.matpr.2020.10.736

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017). https://arxiv.org/abs/1707.06347

[9] Ricci, F., Rokach, L., Shapira, B. (eds.): Recommender Systems Handbook. Springer, Boston, MA (2011). https://doi.org/10.1007/978-0-387-85820-3

[10] Pinthurat, W., Surinkaew, T., Hredzak, B.: An overview of reinforcement learning-based approaches for smart home energy management systems with energy storages. Renewable and Sustainable Energy Reviews **202**, 114648 (2024) https://doi.org/10.1016/j.rser.2024.114648

[11] Riabchuk, V., Hagel, L., Germaine, F., Zharova, A.: Utility-based context-aware multi-agent recommendation system for energy efficiency in residential buildings. Information Fusion, 102559 (2024) https://doi.org/10.1016/j.inffus.2024.102559

[12] Boaventura, D.R.D., *et al.*: State recommender system for actuator devices in smart homes: Integration of deep reinforcement learning and implicit feedback. Expert Systems with Applications **213**, 118997 (2023) https://doi.org/10.1016/j.eswa.2022.118997

[13] Zharova, A., *et al.*: An explainable multi-agent recommendation system for energy-efficient decision support in smart homes. Sensors **23**(4), 1784 (2023) https://doi.org/10.3390/s23041784

[14] Sobhy, S., *et al.*: Context-aware recommender system for multi-user smart home. Multimedia Tools and Applications **81**, 18695–18725 (2022) https://doi.org/10.

1007/s11042-022-12472-7

[15] Eirinaki, M., *et al.*: Real-time recommendations for energy-efficient appliance usage in households. Future Generation Computer Systems **115**, 15–28 (2021) https://doi.org/10.1016/j.future.2020.08.017

[16] Chen, L., Meng, F., Zhang, Y.: Fast human-in-the-loop control for HVAC systems via meta-learning and model-based offline reinforcement learning. IEEE Transactions on Sustainable Computing **8**(3) (2023) https://doi.org/10.1109/TSUSC.2023.3251731

[17] Liang, X., Nijs, F., Say, B., Wang, H.: Human-in-the-loop AI for HVAC management enhancing comfort and energy efficiency. In: Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems (e-Energy '25). Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3679240.3734587

[18] Michelon, F., Zhou, Y., Morstyn, T.: Large language model interface for home energy management systems. In: Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems (e-Energy '25), pp. 590–602. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3679240.3734586