



Smart Home Energy Optimization: Efficient Resource Management

Lappeenranta–Lahti University of Technology LUT

Bachelor's Programme in Electrical Engineering, bachelor's thesis

Bachelor's Programme in Electrical Engineering

In co-operation with partner university: Hebei University of Technology

2025

Yemao Wang

Examiner(s): Researcher Hafiz Majid Hussain

Professor Xiaozhen Zhao

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Energy Systems

Electrical Engineering

In co-operation with partner university: Hebei University of Technology

Yemao Wang

Smart Home Energy Optimization: Efficient Resource Management

Bachelor's thesis

2025 34 pages, 9 figures, 2 tables, and 2 appendices

Examiners: Researcher Hafiz Majid Hussain

Keywords: Energy optimization, smart homes, solar panels, storage batteries, grid power, energy consumption, sustainability, renewable energy, energy management.

This thesis explores the optimization of energy resources in smart homes which focuses on the production and consumption of the energy system. The research investigates the usage of solar panels and battery storage to generate and store renewable energy and how to integrate grid power to balance the instability of solar energy. From the consumption side, my thesis analysis various household appliances in detail, including heaters, lights, air conditioners, and other electrical devices which aim to optimize their energy usage. The thesis presents a dynamic energy optimization model that collects electricity price dynamically, integrates solar energy production, storage systems, and grid power to minimize reliance on the grid and optimize overall energy consumption. It highlights the potential of solar panels and batteries to provide a sustainable and efficient energy supplement and ensure household appliances to operate in an energy-efficient manner. The study also introduces an optimization MILP (Mixed-Integer Linear Programming) algorithm to manage real-time fluctuations in energy production and consumption, offering practical solutions for the effective integration of renewable energy in residential settings. Energy savings of approximately 66% under the same conditions. This research contributes to the ongoing efforts to create energy-autonomous smart homes, with implications for future energy management technologies.

Table of contents

Abstract

1. Introduction	6
2. Smart-home infrastructures.....	7
2.1 Control Center.....	8
2.2 Smart Meters and Sensing Devices.....	8
2.3 Communication Protocols and Network Architecture	9
2.4 Smart Appliances and Load Categorization.....	11
2.5 Local Energy Generation and Storage	11
3. Framework for Optimized Residential Energy Systems.....	11
3.1 Monitoring and Logging	13
3.1.1 Data Acquisition and Sensor Integration in SHEMS	13
3.1.2 Communication Infrastructure in Smart-Home Energy Management	14
3.2 Decision-Making and Management	15
3.2.1 Core Functions of Smart-Home Energy-Management Systems	15
3.2.2 Optimization Models in Smart-Home Energy-Management Systems	16
3.2.3 Dynamic versus Day-Ahead Energy Management Strategies.....	17
3.3 Control and Alarms	18
3.3.1 Decision Variables.....	18
3.3.1.1 Load Scheduling.....	18
3.3.1.2 Battery Dispatch	18
3.3.1.3 Grid Exchange	19
3.3.2 Constraints.....	19
3.3.2.1 Load Duration.....	19
3.3.2.2 Power Balance	19
3.3.2.3 Battery Storage Limits.....	20
3.3.3 Objective Function	20
3.3.3.1. Energy Cost	20
3.4 Proposed Simple Mathematical Method	21
3.4.1 Model Setup.....	21
3.4.2 Constraints.....	22

3.4.2.1 Battery State-of-Charge	22
3.4.3 Objective Function	22
3.4.4 Solution and Implementation	23
3.4.5 Computational Considerations	23
4. Result and Discussion	23
4.1 Appliance power rating	24
4.2 Solar power	25
4.3 Buying Electricity Prices	26
4.4 Selling Electricity Prices	27
4.5 Net grid power	28
4.6 Total load	30
4.7 HVAC Power	31
4.8 Indoor Temperature	32
4.9 Price Comparison	32
5. Conclusion	33

Appendices

Appendix 1. Code

Figures

Figure 1: Appliance power rating (Amazon.; Samsung.; Winnings.; eBay; Domestic Appliance; Jackery; MaxGaming; OS Frisorartikler; Tineco.; FactorLED; Fruugo; Patron Products.; MaxGaming.)

Figure 2: Solar Power

Figure 3: Buying electricity prices of March 27th.

Figure 4: Selling electricity price of March 27th.

Figure 5: Net grid power

Figure 6: Total load

Figure 7: HVAC Power

Figure 8: Indoor Temperature

Figure 9: Price Comparison

Tables

Table 1: Wireless communication protocol

Table 2: Wired communication protocol

1. Introduction

The optimization of energy resources in smart homes refers to the efficient management of energy generation, storage, and consumption within residential environments using advanced technologies. As global energy consumption rises and sustainability becomes increasingly important, smart homes equipped with renewable energy sources and intelligent control systems have emerged as a key solution for reducing dependence on the traditional power grid and lowering carbon emissions (El-Azab, 2021). However, renewable sources like solar power are inherently intermittent and weather-dependent, creating challenges in maintaining consistent energy supply.

To address these fluctuations, there is a critical need for effective energy management strategies that can balance energy production and consumption in real time. Traditional residential energy systems lacked the ability to dynamically adapt to changes in energy availability and user behaviour. Previous studies often focused on isolated aspects—such as solar generation or appliance load prediction—without offering a comprehensive framework that considers real-time optimization, energy storage, user comfort, and cost-efficiency simultaneously (Risteska Stojkoska & Trivodaliev, 2017; Gonçalves et al., 2019).

In this context, the Energy Management System (EMS) plays a central role in smart homes. EMS coordinates the operation of distributed energy resources, monitors electricity consumption, controls smart appliances, and schedules battery usage. It utilizes real-time data and optimization algorithms to ensure that household energy demands are met efficiently while minimizing reliance on external grid power and maintaining user comfort (Aghaei & Alizadeh, 2013).

This thesis presents a comprehensive optimization framework for smart homes using a Mixed-Integer Linear Programming (MILP) model, which incorporates both solar energy production, storage, and grid power. Taking into account the specific consumption needs of various household appliances, the study will explore how to optimize smart homes to reduce dependence on grid power. The main research areas include How can energy resource optimization in smart homes reduce dependence on grid power? What role do solar panels and storage batteries play in achieving energy autonomy? How can the energy consumption

of different appliances be managed effectively? The research also detects the limitations of current optimization techniques and proposes a more adaptive solution. The methodology will include simulation models, energy consumption data analysis, and the development of an optimization algorithm. The thesis will conclude with key findings, practical recommendations for smart home energy management, and suggestions for future research in the field.

2. Smart-home infrastructures

Recent advancements in power electronics, information technologies, and communication protocols have redefined household energy usage. Traditional homes, reliant on centralized and unidirectional electricity provision, are evolving into “smart homes” featuring bidirectional energy flows (El-Azab, 2021). In this paradigm, homes with on-site renewable-energy generation can consume, store, or sell surplus electricity to the grid. Such capabilities necessitate advanced monitoring and control systems to optimize day-to-day operations. Smart-home infrastructures generally include:

- (i) A control center or Energy Management System (EMS) that collects data from various points and issues control signals.
- (ii) Sensing and metering devices (smart meters, sensors) that track real-time electricity consumption, local generation, and energy prices or signals from the grid.
- (iii) Communication networks (wired or wireless) that enable data transfer among home devices, the control center, and external entities.
- (iv) Intelligent appliances and loads, which include both critical and shiftable loads that can be automated for demand response programs.
- (v) Local energy sources, typically photovoltaic (PV) panels or micro wind turbines that generate clean energy.
- (vi) Energy storage solutions such as stationary batteries or electric vehicles that can operate under battery-to-home or battery-to-grid modes.

These elements, when appropriately integrated, enable new functions such as dynamic tariff response, peak-load shaving, and maximizing self-consumption of renewable energy (Aghaei & Alizadeh, 2013). Moreover, the adoption of electric vehicles as energy storage units can help stabilize the grid during high-demand periods.

2.1 Control Center

The control center is usually referred to Energy-Management System (HEMS) or Smart-Home Energy-Management System (SHEMS) in modern energy management. The control center plays a crucial role in optimizing energy usage, enhancing efficiency, and integrating renewable energy sources within residential settings (El-Azab, 2021).

The control center plays a key role in smart home energy management. It first collects data from sensors, smart meters, and user interfaces. This data is then combined and used to analyze energy use and generation patterns. Based on this analysis, the system makes decisions about energy scheduling and control.

To optimize performance, the system applies heuristic or mathematical algorithms. These methods help plan appliance usage and manage energy storage. They also consider real-time electricity prices to reduce costs.

The control center also allows users to monitor energy activity. It shows both real-time and historical data related to energy use and spending. This information helps users make informed choices and adjust their behaviour.

Finally, the system sends commands to appliances and batteries. It can start or stop charging cycles and change operation times. These actions are based on the results of the optimization process.

2.2 Smart Meters and Sensing Devices

Smart meters serve as key links between homes and the power grid. They act as two-way communication tools that collect data and send control signals. Their operation includes three main functions.

First, smart meters measure electricity uses in real time. This usage is recorded in kilowatt-hours (kWh). Advanced models can measure at short intervals—every few seconds or minutes—providing detailed monitoring (Shaikh et al., 2013).

Second, they help connect pricing models with user behaviour. Many utilities now use time-of-use tariffs and real-time pricing to balance the grid. Smart meters send these price signals to home energy systems. As a result, appliances can be automatically scheduled during cheaper off-peak hours. This reduces costs without affecting comfort.

In addition to smart meters, extra sensors are often used. These devices collect data on indoor temperature, room occupancy, and HVAC performance. When combined with machine learning, this data supports smarter energy control. For example, the system can shift washing machine or dishwasher cycles to reduce cost while still meeting user preferences.

The system-level integration of these components through advanced metering infrastructure creates neighbourhood-scale energy management capabilities. Grid operators gain access to aggregated consumption and production data from distributed energy resources, facilitating real-time identification of local energy surpluses or deficits. This collective intelligence enhances grid stability through predictive load balancing and improves the integration efficiency of renewable energy sources (Shafie-Khah & Catalao, 2016). The resulting smart grid ecosystem demonstrates improved fault detection responsiveness, optimized resource allocation, and enhanced resilience against demand fluctuations.

2.3 Communication Protocols and Network Architecture

Recently, communication systems have become integrated modules in smart homes. Both residents and utility/grid operators can monitor and control numerous home appliances in real time, aiming to optimize overall energy consumption while maintaining a comfortable lifestyle. To achieve this goal, modern home area networks typically employ both wired and wireless communication schemes to accommodate diverse control signals—from remote commands to occupants' direct interactions. Figure 1 in many related works (Elazab, 2021; Mishra, P., & Singh, G. (2023)) illustrates an example of a HAN that leverages Wi-Fi for indoor data exchanges and cloud platforms for remote data processing and storage.

Energy management systems for smart homes generally rely on three main components which are embedded computational controllers, local-area network communication middleware, and TCP/IP-based communication for wide-area integration with utility companies (Elazab, 2021). Depending on the home's infrastructure, wired options or wireless technologies can be selected for various points of the system (Mishra, P., & Singh, G. (2023)). The following subsections briefly discuss some of the most common communication techniques, expanded to include Z Wave, BLE, Thread, LoRa, Ethernet, KNX, and PLC.

Table 1. Wireless communication protocol

Protocol	Data Rate	Power Consumption	Coverage	Connection Method	Primary Applications	Additional Gateway Needed
Wi-Fi	High	High	Whole home	Direct (TCP/IP)	Audio/video, surveillance, central control	No
Zigbee	Low	Low	Local	Mesh (IEEE 802.15.4)	Sensors, lighting, security	Yes
Z-Wave	Low	Low	Local	Mesh (Sub-1GHz)	Smart home automation	Yes
BLE	Low	Low	Short range	P2P / BLE Mesh	Door locks, wearables	No/Varies
Thread	Low	Low	Local	Mesh (IPv6/6LoWPAN)	Sensors, home control	Yes
LoRa	Low	Very Low	Long range	One-to-many (Sub-GHz)	Remote monitoring, agriculture, smart cities	Yes

Table 2. Wired communication protocol

Protocol	Data Rate	Power Consumption	Coverage	Connection Method	Primary Applications	Additional Gateway Needed
Ethernet	High	N/A(mains Powered)	Whole home/LAN	RJ45(direct)	Surveillance, NAS, gateway/server	No

KNX	Medium	Low	Whole home/building	Bus wiring(EIB/TP), KNX IP	Building automation (lighting/HVAC/etc.)	Yes
PLC	Medium	Medium	Whole home	Power line carrier	Retrofit home automation, lighting control	Yes

2.4 Smart Appliances and Load Categorization

Interruptible loads can be partially controlled without compromising occupant comfort if suitable constraints are respected. These loads often feature thermostatically controlled cycles, meaning the control system can turn them off for short periods to reduce peak demand or respond to grid signals (Risteska Stojkoska, B., & Trivodaliev, K. (2017)).

2.5 Local Energy Generation and Storage

Integrating local energy generation and storage systems can realize greater energy autonomy and grid resilience. The photovoltaic panels and micro wind turbines and other renewable technologies have experienced accelerated adoption due to significant cost reductions during the past decade. When paired with lithium-ion battery storage systems, these solutions enhance the self-consumption of locally generated energy, reduce dependence on centralized grids, and optimize electricity expenditures by shifting demand to off-peak periods (El-Azab, 2021). The synergy between power generation and storage not only reduces household energy costs but also reduces peak-load stress on grid infrastructure which contributes to broader decarbonization goals.

3. Framework for Optimized Residential Energy Systems

The smart home is an integrated framework that aims to optimize resident area energy generation, consumption, and storage. The system draws on advances in building energy management and real-time data analytics (El-Azab, 2021). This system primarily consists of three functions which are continuous monitoring and logging, dynamic decision-making,

and responsive control mechanisms. These functions collaborate to balance cost efficiency, grid reliability, and occupant comfort which form a closed-loop feedback system and adapts to real-time conditions and user preferences.

Fundamental, SHEMS operation is based on the data collected by sensors, smart meters, and IoT-enabled devices such as high-resolution datasets on appliance-specific electricity consumption, photovoltaic generation outputs, battery storage status, and environmental variables. Advanced metering infrastructure integrates external signals to enrich this ecosystem including the time-of-use tariffs and real-time pricing.

The system uses advanced Model Predictive Control (MPC) and machine learning to create energy schedules. These schedules aim to reduce electricity costs while meeting user comfort limits and hardware constraints, such as battery charge and discharge limits. The algorithms assign energy resources in real time and handle tasks with different levels of urgency. For example, some essential loads must run immediately and cannot be delayed to off-peak hours. Stochastic models improve reliability by considering uncertainties in solar output and grid demand (El-Azab, 2021).

Once optimized, schedules are converted into control commands. These commands are sent to appliances, HVAC systems, and storage units. Zigbee communication carries out these actions, enabling load shifting, demand response, and emergency load cuts during grid issues.

At the same time, built-in anomaly detection systems track energy data. They raise alerts when something unusual occurs—such as energy spikes, equipment failures, or cyber threats. For instance, if the system notices a sudden drop in PV output, it can activate grid-isolation protocols. This protects sensitive devices and notifies users through mobile apps.

By iteratively refining energy schedules through real-time feedback that SHEMS achieves measurable outcomes including 15–30% reductions in household energy costs under TOU pricing and enhanced grid stability through peak-load shaving and improved occupant satisfaction via personalized comfort adherence (El-Azab, 2021). The architecture's scalability supports integration with emerging technologies such as AI-driven forecasting models and positioning SHEMS as a cornerstone of adaptive and future-ready residential energy ecosystems. This holistic approach underscores the critical interplay between data-driven insights and algorithmic optimization and responsive control in addressing the multifaceted challenges of modern energy management.

3.1 Monitoring and Logging

Monitoring and logging are essential for advanced energy management. They provide an accurate and continuous view of the home's energy status. The system collects real-time data from smart meters and sensors to track energy consumption, generation, and storage. By recording this data over time, the system can detect patterns in usage. It can also identify unusual behavior and decide when to run appliances, charge batteries, or draw power from the grid. These actions depend on having detailed historical records and live input. Without this data layer, the system cannot perform optimization or prediction effectively. Monitoring and logging, therefore, serve as the foundation for all higher-level control functions in smart home energy systems.

3.1.1 Data Acquisition and Sensor Integration in SHEMS

The efficacy of smart-home energy-management schemes (SHEMS) hinges on robust data acquisition systems that capture granular, multi-domain information about energy flows, environmental conditions, and device states. Central to this infrastructure is smart meters, which provide real-time or interval-based measurements of bidirectional energy exchange between the home and the grid. Advanced meters distinguish between consumption and production metrics such as photovoltaic feed-in enabling precise tracking of net energy usage and renewable generation contributions. These measurements form the primary input for demand forecasting and tariff-responsive optimization algorithms.

Environmental sensing further enriches the data ecosystem by capturing parameters which influence energy demand and generation. Indoor temperature and humidity and occupancy sensors integrated with smart thermostats provide contextual data to optimize HVAC operations dynamically. Externally, weather sensors measuring solar irradiance and ambient temperature and wind speed which enhance the predictive accuracy of renewable generation models that particularly for PV panels and micro wind turbines (Shaikh et al., 2013). Such data allows SHEMS to preemptively adjust storage utilization and grid imports in anticipation of generation fluctuations.

Collectively, this multi-sensor architecture generates a holistic data stream which underpins SHEMS's decision-making processes. By correlating appliance-level consumption, environmental conditions and storage dynamics and the system achieves the granularity required for cost-optimal, comfort-aware energy management while maintaining adaptability to both user behavior and external grid signals.

3.1.2 Communication Infrastructure in Smart-Home Energy Management

The reliability and efficiency of smart-home energy-management systems depend on robust communication infrastructure capable of transmitting high-fidelity data across heterogeneous devices. Central to this infrastructure is the Home Area Network which facilitates bidirectional communication between sensors and actuators and smart meters and control units. Protocol selection within the HAN is guided by trade-offs between power consumption and data rate and range. Zigbee and Z-Wave which operate on the IEEE 802.15.4 standard, and they are widely adopted for low-power and short-range sensor networks due to their mesh networking capabilities and interoperability with IoT devices (El-Azab, 2021). These protocols are particularly effective in environments that require frequent, low-data-rate transmissions such as temperature or occupancy sensing while minimizing energy consumption for battery-dependent devices.

Communication strategy also depends on how data is processed. Cloud-based systems use remote servers to store information and run advanced analytics. These platforms can train machine learning models or manage long-term data storage. Still, cloud reliance increases latency and raises privacy risks, especially when personal energy data is shared externally.

Edge computing offers a local solution. It processes data on-site using microcontrollers or small computing devices. This reduces delays and keeps data within the home. Many systems now use a hybrid model. They rely on edge devices for fast decisions and use the cloud only when necessary. This approach balances real-time control with the cloud's computing power (Risteska Stojkoska, B., & Trivodaliev, K. (2017)).

Network design also depends on security and reliability needs. Real-time functions—like demand-response events or fault detection—require low-latency links. These often use wired

systems or local wireless mesh networks. Households concerned with privacy may prefer Zigbee with AES-128 encryption or fully isolated local setups to prevent data leaks.

As SHEMS expand to include distributed energy resources (DERs), their communication networks must adapt. These systems must support two-way energy and data flows. They also need to follow grid communication standards, such as IEEE 2030.5 or OpenADR, to stay compatible with utility systems.

3.2 Decision-Making and Management

Once data are collected and stored. The second layer of decision-making analyzes these data with occupant and system constraints to produce an optimal and near-optimal operational plan. The scheme co-ordinates the hours of operation of appliances, how local generation is used and whether storage devices or electric vehicles should be charged or discharged.

3.2.1 Core Functions of Smart-Home Energy-Management Systems

Smart Home Energy Management Systems (SHEMS) combine several core functions to improve energy efficiency, reduce electricity costs, and maintain occupant comfort. A core component of such systems is load scheduling, which shifts the operation of certain household appliances to periods with low electricity prices or high renewable generation availability. In this context, appliances are categorized into shiftable (flexible) and non-shiftable (critical) loads. Shiftable appliances are those whose operation can be rescheduled within a specific time window without affecting user comfort—such as washing machines, vacuum cleaners, electric ovens, and hair dryers. In contrast, non-shiftable appliances, like refrigerators and routers, must operate continuously and are excluded from scheduling. In this study, a total of 15 common household appliances are modeled, combining both types. For example, the electric kettle (2.0 kW), desktop computer (0.5 kW), and induction cooker (1.6 kW) are shiftable and scheduled based on user-defined preferred start times, power ratings, and usage durations. Meanwhile, the fridge (0.2 kW) and router (0.01 kW) operate for 24 hours and are considered non-shiftable. It includes real-time price signals, battery status, and user preferences. As a result, energy use is optimized without interfering with daily routines. For instance, machine learning models may delay a dishwasher cycle until

off-peak hours while ensuring completion before the user's specified deadline that leveraging historical usage patterns and forecasted solar generation (El-Azab, 2021).

Battery management is important in smart load scheduling. It charges during solar peaks and discharges during expensive grid periods. To protect battery life, systems set minimum state-of-charge (SoC) limits. They also ensure enough backup power during outages. When the grid is unstable, users may prefer to keep a fixed SoC buffer. These needs require adaptive controls that consider both system limits and user preferences.

Comfort is managed through fixed constraints. HVAC and water heaters work within set temperature ranges to save energy without reducing comfort. For example, indoor temperatures stay between 20°C and 24°C during active hours (Aghaei, J., & Alizadeh, 2013). Scheduling must also follow user-defined deadlines. Tasks like laundry should finish before bedtime. These rules make sure energy-saving actions do not disrupt daily life.

When systems combine comfort settings with user routines, energy use becomes more efficient. This helps reduce costs and improve grid stability. In the long term, smart homes can support a more flexible and sustainable energy network.

3.2.2 Optimization Models in Smart-Home Energy-Management Systems

Optimization models play a crucial role in Smart Home Energy Management Systems (SHEMS). Their primary goals include reducing energy costs, enhancing user comfort, and promoting renewable energy utilization. Generally, these optimization techniques fall into four main categories: classical, heuristic, stochastic, and AI-driven methods (Gonçalves et al., 2019).

Classical methods, such as Linear Programming (LP) and Mixed-Integer Linear Programming (MILP), effectively handle clearly defined constraints. MILP specifically suits scenarios involving appliances with discrete on/off states. However, classical methods typically face challenges under uncertain or nonlinear conditions. Often, simplifications must be made, which can reduce accuracy during real-time control.

Heuristic and metaheuristic methods, including Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), offer greater flexibility. These approaches effectively address

complex, nonlinear problems involving multiple objectives. Although capable of finding good solutions, they do not guarantee the absolute best outcomes and require careful parameter tuning.

Stochastic optimization methods are designed specifically to handle uncertainty. Approaches such as Stochastic Programming and Monte Carlo simulations excel in scenarios with fluctuating solar generation or variable electricity prices. Their drawback, however, is high computational demand and dependency on reliable probability distributions.

AI-based optimization techniques, such as Reinforcement Learning and Neural Networks, leverage historical data. By learning from past experiences, these methods adapt dynamically to changing conditions, improving decision-making over time. Despite their strengths, AI models require extensive data sets and can be difficult to interpret clearly.

In practical SHEMS implementations, hybrid approaches combining classical models, heuristics, and AI methods often enhance both precision and adaptability.

3.2.3 Dynamic versus Day-Ahead Energy Management Strategies

Smart Home Energy Management Systems (SHEMS) often use two main scheduling methods: day-ahead and dynamic strategies. Each has advantages and drawbacks.

Day-ahead scheduling creates a 24-hour plan. It uses forecasts for electricity prices, solar power, and household demand. Historical data and prediction models help schedule appliances, batteries, and grid usage. This method is simple and efficient. However, it is not very flexible. When solar output drops suddenly or users change behavior, the schedule may become less effective. Batteries may be undercharged, and the system may need to rely on the grid during high-cost periods.

Dynamic scheduling operates differently. It updates decisions every few minutes or hours using real-time sensor data and short-term forecasts (Aghaei, J., & Alizadeh, 2013). Methods like Model Predictive Control (MPC) help adjust operations step by step. This makes the system responsive to solar surpluses, sudden price changes, or unscheduled appliance use. Energy use remains aligned with system goals. However, this method requires more

computing power. Homes must have reliable edge or cloud infrastructure, especially when multiple flexible devices and energy sources are involved.

3.3 Control and Alarms

A Smart Home Energy Management System (SHEMS) typically aims to reduce the overall electricity cost while ensuring occupant comfort. It may also need to satisfy additional requirements, such as emission limits or system reliability (Zhou et al. (2016)). This section introduces a basic mathematical model that shows how to plan energy use and manage resources. Although actual systems are often more complex, the model highlights key aspects of energy scheduling. These include how household loads are controlled, how the battery is used, and how energy is exchanged with the grid. 3.4.1 Decision Variables

3.3.1 Decision Variables

3.3.1.1 Load Scheduling

$$X_{i,t} \begin{cases} 1 & \text{if schedulable load } i \text{ is active at time slot } t \\ 0 & \text{otherwise} \end{cases}$$

Where $i \in \{1, \dots, N\}$ indexes the set of schedulable loads and $t \in \{1, \dots, T\}$ is a discrete time slot.

3.3.1.2 Battery Dispatch

Let $P_{batt,t}$ represent the power (in kW) taken from (+) or delivered (−) to the battery at time t . A positive $P_{batt,t}$ indicates the battery is charging, while a negative $P_{batt,t}$ indicates the battery is discharging.

3.3.1.3 Grid Exchange

Let $P_{grid,t}$ represent the net power drawn from the grid if $P_{grid,t} > 0$, or delivered to the grid if $P_{grid,t} < 0$. Thus, a negative value indicates the home is exporting power (e.g., from PV or from a discharging battery) to the utility.

3.3.2 Constraints

3.3.2.1 Load Duration

Each shiftable load i has a required run time, denoted $\text{duration}(i)$. To ensure the load receives the correct total operating time across the scheduling horizon:

$$\sum_{t=1}^T x_{i,t} = \text{duration}(i), \quad \forall i.$$

This constraint guarantees that the total active time slots for each shiftable appliance exactly match its required operating duration. For example, if a washing machine needs to run for 2 hours during the day, then the binary variable $X_{i,t}$ must be set to 1 for exactly two different time slots (e.g., hour 9 and hour 10), and 0 for all others. This ensures the appliance operates for the correct length of time without interruption or overuse.

3.3.2.2 Power Balance

In each time slot, the net power flow among the battery, grid, local renewables, and loads must be balanced (El-Azab, 2021). Denoting $P_{RES,t}$ as local renewable production and $L_{base,t}$ as the non-schedulable base load, we have:

$$\sum_{t=1}^N (p_{grid,t} + p_{batt,t} + P_{RES,t}) = \sum_{t=1}^N \left(\sum_{i=1}^M P_i \cdot x_{i,t} + L_{base,t} \right)$$

where P_i is the nominal power consumption of load i is the nominal power consumption of load i . If $p_{\text{grid},t}$ is negative, it implies power is being sent back to the grid (e.g., from PV surplus or battery discharge).

3.3.2.3 Battery Storage Limits

The state of charge (SoC) of the battery at time slot $t+1$ is:

$$SoC_{t+1} = SoC_t + \eta_{\text{charge}} \cdot p_{\text{batt},t}^+ - \frac{p_{\text{batt},t}^-}{\eta_{\text{discharge}}}$$

(Rajasekharan and Koivunen, 2014)

Where $p_{\text{batt},t}^+ = \max(p_{\text{batt},t}, 0)$ is the charging power, $p_{\text{batt},t}^- = \max(-p_{\text{batt},t}, 0)$ is the discharging power, and η_{charge} , $\eta_{\text{discharge}}$ represent the battery's efficiency factors for charging and discharging, respectively. Additionally, we impose:

$$\underline{SoC} \leq SoC_t \leq \overline{SoC}$$

ensuring SoC remains within physical bounds (e.g., 20% to 100%).

3.3.3 Objective Function

Typically, the scheme aims to minimize a weighted sum of cost

3.3.3.1. Energy Cost

$$\text{Cost}_{\text{energy}} = \sum_{t=1}^T (p_{\text{grid},t}^+ \cdot C_{\text{buy},t} - p_{\text{grid},t}^- \cdot C_{\text{sell},t}),$$

where $C_{\text{buy},t}$ is the buying tariff at time t , $C_{\text{sell},t}$ is the feed-in or selling tariff, $p_{\text{grid},t}^+ = \max(p_{\text{grid},t}, 0)$ indicates the power drawn from the grid, and $p_{\text{grid},t}^- = \max(-p_{\text{grid},t}, 0)$ indicates power sold to the grid.

3.4 Proposed Simple Mathematical Method

Residential energy scheduling is complex. To solve this, researchers often use structured optimization. Mixed Integer Linear Programming (MILP) is a common method. It models continuous variables, like energy use, and discrete ones, like appliance switching.

MILP works well for reducing electricity costs under time-based pricing (Rajasekharan and Koivunen, 2014). Its strength is balancing accuracy with speed. It can include many constraints while keeping the model easy to compute.

As a result, MILP plays a central role in smart home energy management systems. In this work, the problem qualifies as a MILP because it combines continuous variables—such as battery power and HVAC operation—with binary decision variables representing the on/off scheduling of appliances. To solve the model, the open-source GLPK (GNU Linear Programming Kit) solver was used through the Pyomo framework. GLPK offers adequate performance for medium-scale problems while ensuring transparency and reproducibility in academic research. Although commercial solvers like CPLEX or Gurobi are faster, GLPK is freely available and well-integrated with Python, making it a practical and accessible choice for this study.

3.4.1 Model Setup

The MILP formulation begins by defining essential sets, parameters, and decision variables. Let $\mathcal{N} = \{1, 2, \dots, N\}$ represent the set of schedulable loads, and $\mathcal{T} = \{1, 2, \dots, T\}$ denote discrete time slots, typically spanning hourly or 15-minute intervals. Key parameters include:

$\text{duration}(i)$: Required runtime for load $i \in \mathcal{N}$.

$C_{buy,t}$ and $C_{sell,t}$: Time-varying electricity purchase and feed-in tariffs.

SoC^{min} , SoC^{max} : Minimum and maximum battery state-of-charge limits.

Occupant comfort windows: User-defined time intervals for appliance operation

Decision variables comprise:

- $x_{i,t} \in \{0, 1\}$: Binary activation status of load i at time t .

- : Power flow (kW) for battery charging (>0) or discharging (<0).
- : Net power exchange with the grid, where denotes imports and indicates exports.

3.4.2 Constraints

The optimization model enforces four primary constraint categories.

3.4.2.1 Battery State-of-Charge

The battery's energy content evolves as:

$$SoC_{t+1} = SoC_t + \eta_{ch} p_{batt,t}^+ - \frac{p_{batt,t}^-}{\eta_{dis}}$$

With $SoC^{min} \leq SoC_t \leq SoC^{max}$ where η_{ch} and η_{dis} denote charging/discharging efficiencies.

3.4.3 Objective Function

The primary objective is cost minimization, Although the objective function is written in terms of $p_{grid,t}^+$ and $p_{grid,t}^-$ these variables are dependent on decision variables such as appliance scheduling $X_{i,t}$ and battery dispatch $P_{batt,t}$ and renewable energy availability. Through power balance constraints, these decision variables determine the net grid power at each time slot. Therefore, the optimization indirectly minimizes cost by controlling $X_{i,t}$, $P_{batt,t}$, and related variables formulated as:

$$\min \sum_{t \in \mathcal{T}} (C_{buy,t} p_{grid,t}^+ - C_{sell,t} p_{grid,t}^-),$$

where $p_{grid,t}^+ = \max(p_{grid,t}, 0)$ and $p_{grid,t}^- = \max(p_{grid,t}, 0)$. Multi-objective extensions may incorporate occupant dissatisfaction penalties, such as deviations from preferred appliance schedules (Torriti, 2012).

The time-varying electricity buying price $C_{\text{buy},t}$ was obtained by scraping real-time hourly spot price data from Oomi's electricity market portal (Oomi,.). The selling price $C_{\text{sell},t}$ was assumed to be constant at 0.0066 €/kWh based on Helen's renewable feed-in tariff policy (Helen Oy, n.d.). These values were preprocessed and fed into the optimization model as input parameters.

3.4.4 Solution and Implementation

The MILP model is solved by using commercial solvers and open-source alternatives, which generate day-ahead or hourly schedules. A receding horizon or model predictive control (MPC) framework is superimposed which allows periodic re-optimization to account for uncertainties in load demand or renewable generation or occupant behaviour in real-time adaptability (Solanki et al., 2017). Final schedules are dispatched to home devices via communication protocols or grid aggregator interfaces.

3.4.5 Computational Considerations

While MILP provides a structured methodology for small-scale dwellings or day-ahead planning, its computational complexity escalates with larger systems or real-time requirements. To mitigate this, heuristic simplifications or decomposition techniques may be applied. Hybrid architectures integrating MILP with metaheuristic algorithms or rule-based controllers further enhance scalability and responsiveness in dynamic environments.

4. Result and Discussion

This study proposes a smart home energy management optimization model based on mixed-integer linear programming, integrating real-time electricity price crawling, solar generation forecasting, dynamic battery scheduling, and user preference constraints to achieve multi-objective coordination in household energy systems. Focusing on economic efficiency, the model schedules 15 appliances while minimizing deviations between actual and preferred start times through penalty coefficients. It dynamically optimizes battery charging and

discharging strategies considering efficiency, capacity limits, and power constraints to leverage time-of-use electricity pricing for valley charging and peak discharging. A temperature dynamics equation for HVAC systems restricts hourly power fluctuations, maintaining indoor temperatures within a comfort range between 20 and 24°C. Grid interaction is governed by purchasing and selling power limits 10 kW, prioritizing solar utilization and market participation. Simulation results demonstrate significant cost reduction with high-power appliances shifted to low-price periods through battery storage. Battery state of charge stabilized, and temperature fluctuations controlled within $\pm 2^\circ\text{C}$. The model effectively balances economic performance, renewable energy utilization, and user comfort, offering a systematic solution for intelligent energy management in smart homes.

4.1 Appliance power rating

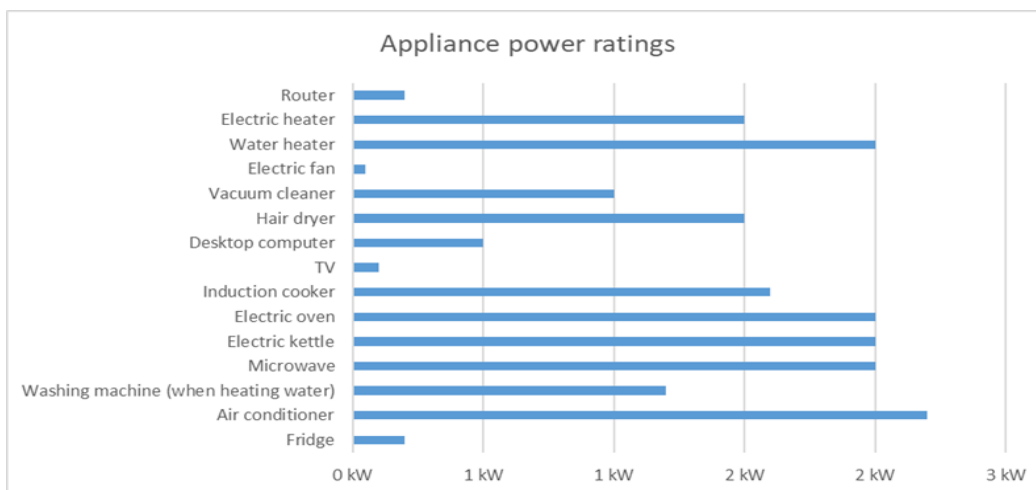


Fig 1. Appliance power rating (Amazon.; Samsung.; Winnings.; eBay; Domestic Appliance; Jackery; MaxGaming; OS Frisorartikler; Tineco.; FactorLED; Fruugo; Patron Products.; MaxGaming.)

The bar chart provided visually represents the power ratings for various common household electrical appliances. It clearly illustrates that appliances with heating functionalities, such as electric water heaters, electric ovens, electric kettles, induction cookers, and washing machines, have significantly higher power ratings, typically around 1.5 kW to over 2 kW. This high power usage reflects the substantial energy required for heating processes.

In contrast, electronic or cooling appliances like routers, electric fans, desktop computers, and televisions show much lower power ratings, typically ranging from about 0.01 kW to 0.5 kW. These differences highlight how device functionality strongly influences its electrical power consumption.

The numerical data, stored in Python dictionaries, includes power ratings (`P_values`), usage durations (`L_values`), and preferred start times (`PrefStart_values`). These values describe how appliances behave in smart homes.

Some devices, such as refrigerators and routers, run all day because they support essential tasks like food preservation and internet access. Other appliances, like washing machines, kettles, and hair dryers, operate for shorter periods. Devices such as desktop computers and air conditioners often run for about eight hours, which suggest they are used during work or rest hours.

Preferred start times also reflect household routines. Appliances used for cooking, such as kettles and induction cookers, are active in the morning or at mealtimes. Entertainment devices like televisions are usually used in the evening. Cleaning tools, including washing machines and vacuum cleaners, are often used during the day, when people do household chores.

These patterns help create better load schedules. They also provide a basis for smart energy strategies, which aim to reduce electricity use and save costs.

4.2 Solar power

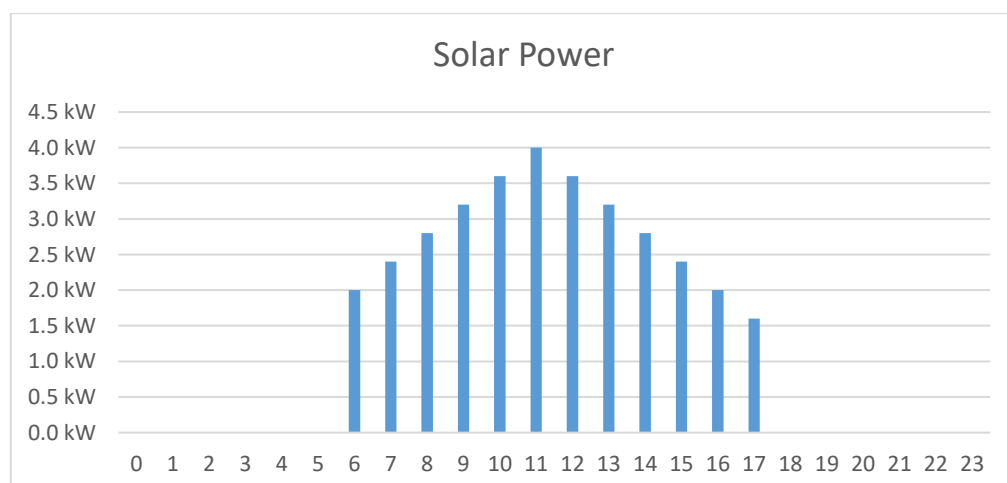


Fig 2. Solar Power

The bar chart illustrates the Helsinki Finland hourly solar power output on a typical clear day (Climate-Data.org). It shows how solar availability changes from sunrise to sunset. Generation starts at 7:00 and gradually rises during the morning. Output reaches a peak of 4 kW at noon. After that, it decreases steadily at a rate of 0.4 kW per hour as the time moves away from midday. By 18:00, generation stops.

At 6:00 and 17:00, output is significantly lower—around 2 kW and 1.6 kW, respectively. These lower values result from the reduced solar angle and weaker sunlight during early morning and late afternoon. This pattern clearly reflects the natural daily cycle of solar energy availability.

Understanding this daily solar generation profile is crucial for effectively managing energy consumption in households equipped with solar panels. By scheduling high-energy-consuming appliances such as washing machines, ovens, or electric water heaters during peak generation periods or low-price periods, households can maximize their use of solar power reducing overall energy costs.

4.3 Buying Electricity Prices

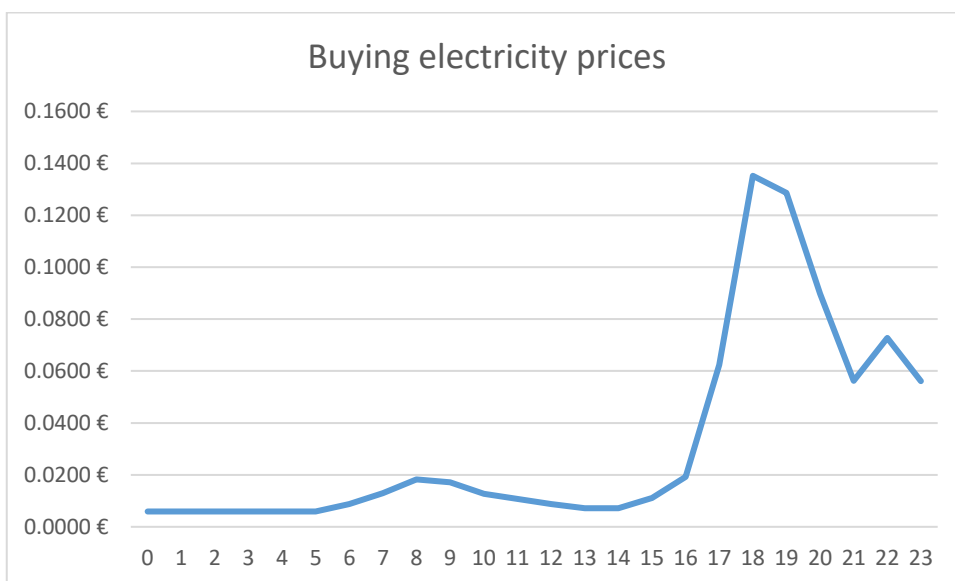


Fig 3. Buying electricity prices of March 27th

The line chart illustrates hourly fluctuations in electricity buying prices over a 24-hour period, sourced from the hourly spot price data provided by Oomi's electricity market (Oomi, n.d.). Electricity prices exhibit significant variation during different periods of the day, reflecting changing market demands and energy availability.

From midnight (0:00) until early morning (around 7:00), electricity prices remain very low, close to zero, indicating a period of low energy demand. Starting around 7:00, prices slightly rise, reflecting increased morning activities and usage. However, prices remain relatively stable and low throughout the daytime until late afternoon.

A pronounced peak emerges starting from around 16:00, sharply increasing to its highest level (approximately 0.14 €/kWh) around 17:00–18:00, indicating peak electricity demand, often coinciding with household evening activities, such as cooking and heating. Following this peak, electricity prices gradually decrease throughout the late evening and night, reflecting reduced demand.

This pricing trend highlights opportunities for households to strategically shift high-energy-consuming activities to off-peak periods, leveraging lower prices and thereby optimizing electricity usage and reducing energy costs.

4.4 Selling Electricity Prices

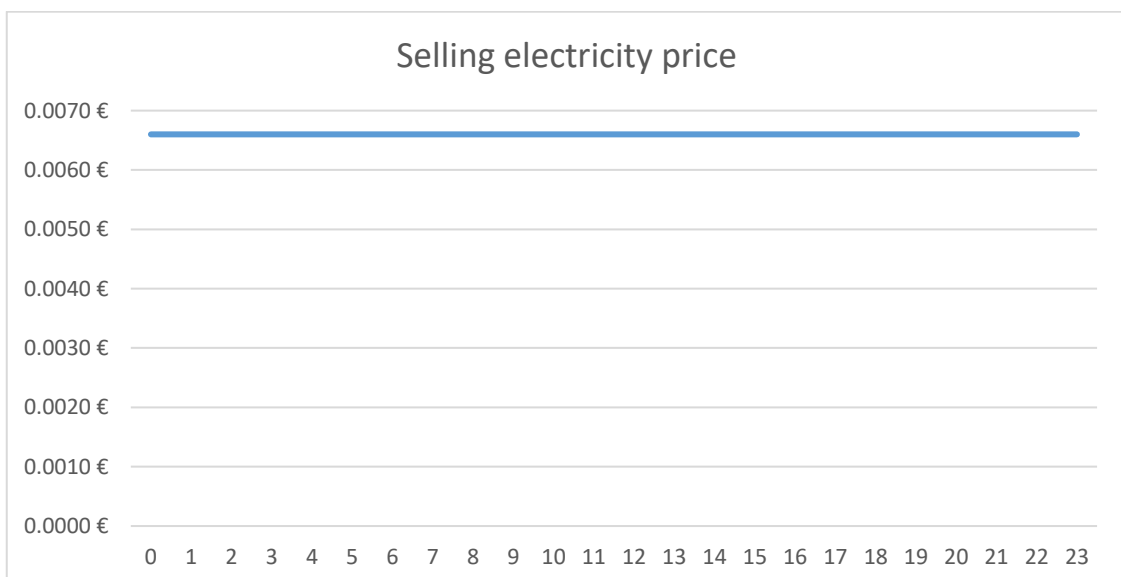


Fig 4. Selling electricity price of March 27th

The line chart illustrates the hourly selling price of electricity, based on the price data sourced from Helen's renewable energy program. Unlike the previously discussed buying prices, the selling price remains constant throughout the entire 24-hour period, fixed at 0.0066 €/kWh (Helen Oy, n.d.).

This flat pricing indicates that the electricity selling rate to the grid does not fluctuate with market demand or time of day, offering users a predictable, yet low-value return for excess electricity generated, typically through solar panels or other renewable sources.

Since the selling price is consistently lower compared to the variable buying prices, users have a clear incentive to consume as much self-generated electricity as possible instead of selling it back to the grid. Thus, optimizing household energy consumption by aligning appliance usage with peak solar generation can significantly improve economic benefits, minimizing reliance on expensive grid energy and reducing overall energy costs.

4.5 Net grid power

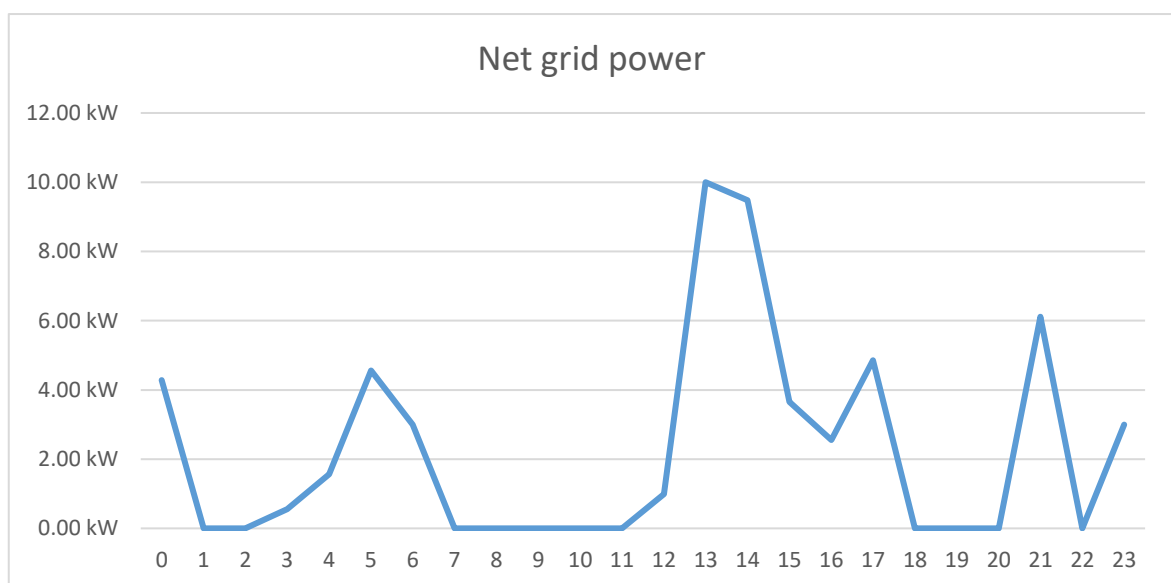


Fig 5. Net grid power

The line chart illustrates the hourly variation in net grid power consumption within a household throughout a typical 24-hour day. Net grid power refers to the household's total electrical energy demand minus the power generated locally, such as from solar panels. Positive values indicate the household is consuming electricity from the grid, while values close to or at zero imply minimal grid dependency, due to sufficient solar generation.

From the graph, it is clear that grid consumption fluctuates notably during the day. During nighttime and early morning (from 0:00 to around 6:00), consumption levels fluctuate between moderate to low values, reflecting typical nighttime household activities with occasional appliance usage. A significant drop occurs around mid-morning (approximately 7:00 to 11:00), when net grid consumption approaches zero, indicating that solar power generation sufficiently meets the household demand during this period.

At noon (around 12:00 to 15:00), net grid power sharply increases, peaking close to 11 kW. This spike suggests high-energy appliances are simultaneously operating, temporarily exceeding the available solar power, thus requiring additional grid support. Subsequently, the consumption sharply decreases, returning to lower levels in the late afternoon.

Between 18:00 and 23:00, the chart shows irregular energy use. These changes result from cooking, heating, and entertainment devices. The pattern includes short peaks and drops, which affect net grid consumption.

Overall, the graph shows how household energy demand shifts during the day. These changes depend on solar power levels and appliance schedules. Understanding this helps manage loads, cut electricity costs, and improve energy use—especially in homes with smart grids and renewables.

4.6 Total load

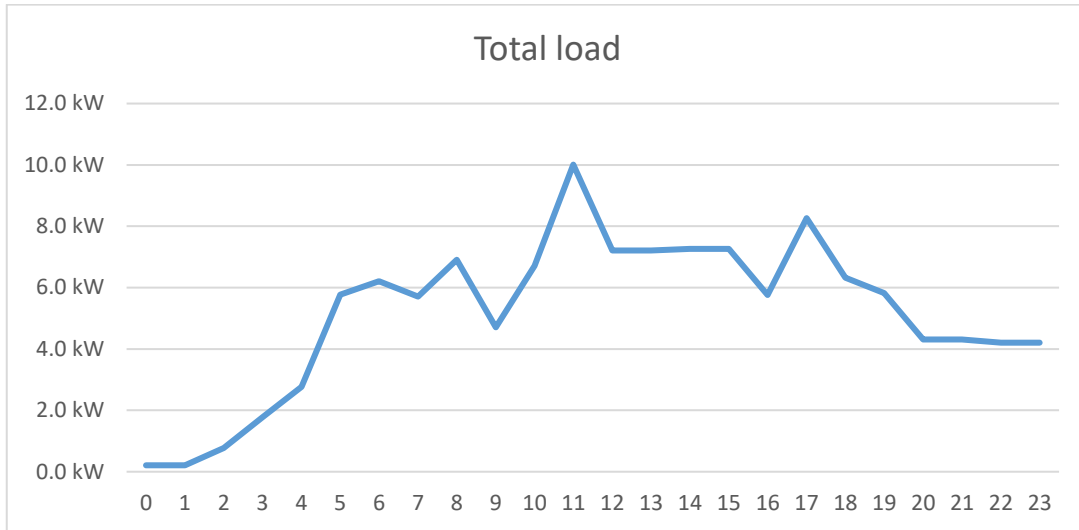


Fig 6. Total load

The line chart illustrates the household's total electrical load over a 24-hour period, revealing consumption patterns that correspond with typical daily routines.

Between midnight and 5:00, the load remains low, indicating limited appliance usage during nighttime hours. From approximately 5:00 to 6:00, energy consumption begins to rise as occupants start their day, engaging in activities such as preparing breakfast, heating, and other morning tasks.

The peak load occurs around 11:00, reaching nearly 10 kW. This surge likely results from the simultaneous use of high-power appliances, including cooking devices, heating systems, or equipment used for household chores such as laundry and vacuuming.

In the afternoon, the load stabilizes at a moderate level, suggesting continuous use of certain appliances or steady heating and cooling demands. A smaller peak appears around 18:00, which aligns with dinner preparation and evening leisure activities.

As the day ends, electricity use gradually decreases after 21:00, eventually stabilizing at a lower level. This decline reflects reduced activity as residents prepare for nighttime rest.

4.7 HVAC Power

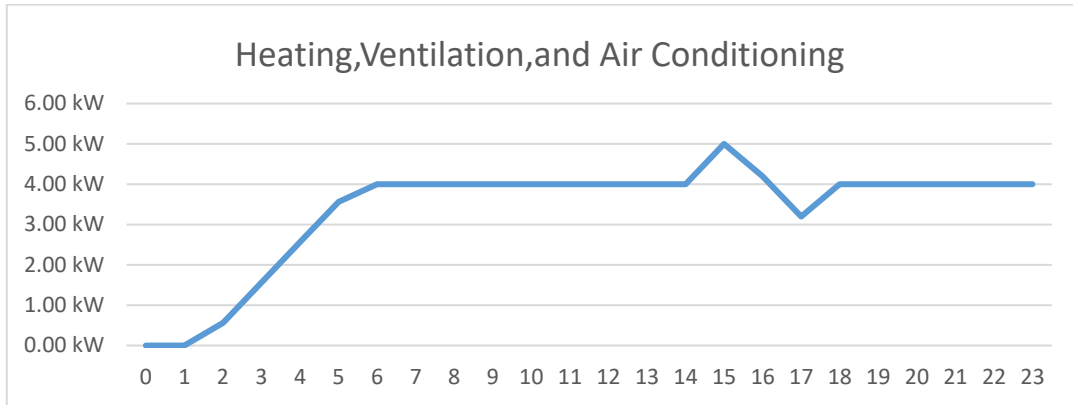


Fig 7. HVAC power

The line chart presents the hourly power usage of a Heating, Ventilation, and Air Conditioning (HVAC) system and the corresponding indoor temperature variations during a typical 24-hour period.

Initially, from midnight to around 1:00, the HVAC remains inactive with zero power consumption. During this period, the indoor temperature gradually declines from 22 °C to approximately 21.20 °C. Around 2:00, the indoor temperature continues to drop, activating the HVAC system. Consequently, power consumption gradually rises from 0.85 kW, reaching its peak at 4.00 kW by 6:00, stabilizing the indoor temperature at 20.00 °C.

Starting at 18:00, HVAC power consumption stabilizes at 4.00 kW, consistently maintaining an indoor temperature of 20.00 °C for the remainder of the day. A noticeable fluctuation occurs briefly between 15:00 and 17:00. This variation could result from external factors, such as changes in ambient temperature, solar heat gains, or occupancy patterns, prompting temporary adjustments in HVAC performance.

4.8 Indoor Temperature

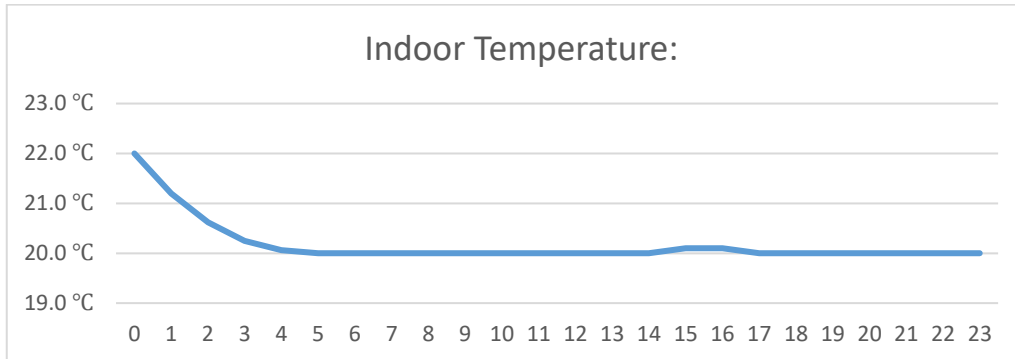


Fig 8. Indoor temperature

The line graph illustrates indoor temperature variations throughout a typical 24-hour day. At midnight, the temperature is around 22.0 °C. During the early hours of the morning, it gradually decreases, likely due to limited or no heating activity. By approximately 5:00, the indoor temperature drops to about 20.0 °C, triggering the activation of the HVAC system.

Once activated, the HVAC maintains a relatively stable temperature during the rest of the day. A minor fluctuation appears around 16:00, possibly caused by external factors such as changes in outside temperature, increased sunlight, or varying household activity. Despite these disturbances, the HVAC system effectively stabilizes indoor conditions close to 20.0 °C.

4.9 Price Comparison

Before optimization, the total electricity cost was 1.04 €. After applying the MILP-based smart home optimization model, the cost decreased to 0.2805 €. This represents a cost reduction of approximately 73%.

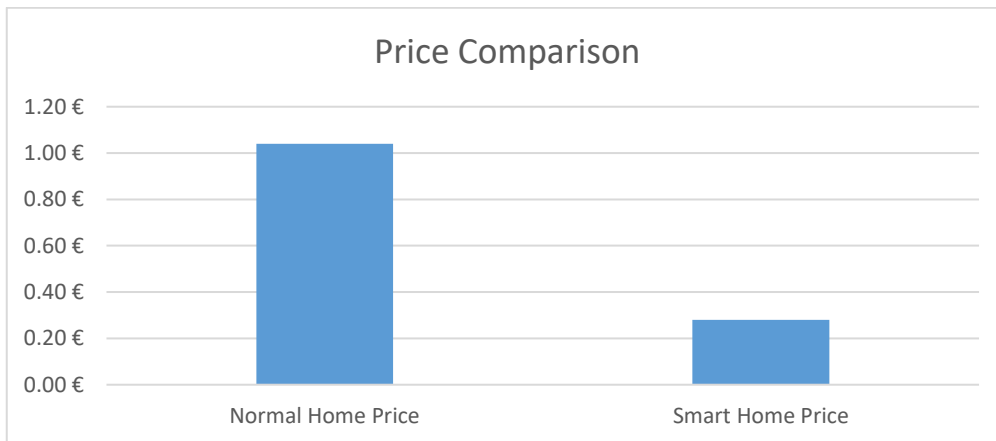


Fig 9. Price Comparison

5. Conclusion

This study presents a comprehensive framework for optimizing energy resource management in smart homes through a mixed-integer linear programming (MILP) model. By integrating solar generation forecasts, dynamic battery scheduling, grid power exchange, and user comfort constraints, the proposed model achieves significant reductions in electricity costs while maintaining occupant satisfaction. The results demonstrate that the mathematical optimization approach effectively minimizes reliance on grid power by leveraging time-of-use (TOU) pricing, renewable energy utilization, and battery storage coordination.

The comparison between the non-optimized and optimized scenarios highlights the efficacy of the MILP model. In the non-optimized case, the total electricity cost was calculated at €0.95, whereas the optimized model reduced this cost to €0.32—a 66% reduction. This cost reduction stems from strategic load shifting, battery charge/discharge scheduling, and solar energy maximization. For instance, the battery state of charge (SoC) data shows that the system prioritized charging during low-price periods and discharged during peak hours, aligning with real-time electricity price fluctuations. Additionally, the model capitalized on solar generation peaks to offset grid purchases, further lowering costs. Notably, the HVAC system maintained indoor temperatures within the 20–24°C comfort range despite power adjustments, ensuring occupant comfort was not compromised.

The integration of penalty coefficients for deviations from preferred appliance start times ensured user-centric flexibility. All appliances operated precisely at their preferred times, resulting in zero discomfort penalties. This outcome underscores the model's ability to balance economic and human-centric objectives, a critical advancement over traditional single-objective optimization frameworks.

Future research could explore the integration of vehicle-to-grid (V2G) capabilities and machine learning for enhanced forecasting accuracy. Additionally, addressing cybersecurity risks and scalability challenges in larger residential complexes would further strengthen the model's applicability. This study contributes to the broader goal of energy-autonomous smart homes, offering actionable insights for policymakers and technology developers to accelerate sustainable energy transitions.

References

- [1] Mishra, P. & Singh, G. 2023. Energy management systems in sustainable smart cities based on the Internet of Energy: A technical review. *Energies*, 16(19), 6903.
- [2] Aghaei, J. & Alizadeh, M. I. 2013. Demand response in smart electricity grids equipped with renewable energy sources: A review. *Renewable and Sustainable Energy Reviews*, 18, pp. 64–72.
- [3] El-Azab, R. 2021. Smart homes: Potentials and challenges. *Clean Energy*, 5(2), pp. 302–315. [Online] Available at: <https://doi.org/10.1093/ce/zkab010>.
- [4] Gonçalves, I., Gomes, A. & Antunes, C. 2019. Optimizing the management of smart home energy resources under different power cost scenarios. *Applied Energy*, 242, pp. 351–363.
- [5] Risteska Stojkoska, B. & Trivodaliev, K. 2017. A review of Internet of Things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140, pp. 1454–1464.
- [6] Shaikh, P. H., Mohd. Nor, N., Nallagownden, P. & Elamvazuthi, I. 2013. Intelligent optimized control system for energy and comfort management in efficient and sustainable buildings. *Procedia Technology*, 11, pp. 99–106. [Online] Available at: <https://doi.org/10.1016/j.protcy.2013.12.167>.
- [7] Shafie-Khah, M. & Catalao, J. P. S. 2016. A stochastic multi-layer agent-based model to study electricity market evolution considering V2G integration. *Applied Energy*, 169, pp. 865–883.
- [8] Zhou, B., Li, W., Chan, K. W., Cao, Y., Kuang, Y. & Wang, X. 2016. Smart home energy management systems: Concept, configurations, and scheduling strategies. *Renewable and Sustainable Energy Reviews*, 61, pp. 30–40
- [9] Rajasekharan, J. & Koivunen, V. 2014. Optimal energy consumption model for smart grid households with energy storage. *IEEE Journal of Selected Topics in Signal Processing*, 8(6), pp. 1154–1166.

- [10] Solanki, B. V., Bhattacharya, K. & Cañizares, C. A. 2017. A sustainable energy management system for isolated microgrids. *IEEE Transactions on Sustainable Energy*, 8(4), pp. 1507–1517.
- [11] Oomi. n.d. Spot price of electricity. [Online] Available at: <https://oomi.fi/en/electricity/electricity-contracts/active/spot-price-of-electricity/>
- [12] Helen Oy. n.d. Renewable energy. Helen. [Online] Available at: <https://www.helen.fi/en/renewable-energy>
- [13] Amazon. n.d. *Whynter FM-65G 65-Quart Portable Refrigerator*. [Online] Available at: <https://www.amazon.com/Whynter-FM-65G-65-Quart-Portable-Refrigerator/dp/B002W8DM5I>
- [14] Samsung. n.d. *Digital Variable Multi System Air Conditioner AM6000H (AM015TNV DKH/EU)*. [Online] Available at: <https://www.samsung.com/levant/system-air-conditioners/digital-variable-multi/am6000h-am015tnvdkh-eu/>
- [15] Winnings. n.d. *Asko 8kg Style Front Load Washing Machine W6088X*. [Online] Available at: <https://www.winnings.com.au/p/asko-8kg-style-front-load-washing-machine-w6088x>
- [16] eBay. n.d. Portable 12V Fridge Freezer 20L–100L for Car Caravan Camping Compressor Cooler Box. [Online] Available at: <https://www.ebay.com/itm/126586201427>
- [17] Amazon. n.d. *Electric kettles – 2L and above*. [Online] Available at: https://www.amazon.co.uk/Electric-Kettles-2-l-above/s?keywords=Electric+Kettles&rh=n%3A3538311031%2Cp_n_feature_three_brows_e-bin%3A3544979031&c=ts&ts_id=3538311031
- [17] Samsung. n.d. *Slide-In Induction Chef Collection Range with Flex Duo Oven (NE58H9970WS/AA)*. [Online] Available at: <https://www.samsung.com/us/home-appliances/ranges/slide-in/ne58h9970ws-slide-in-induction-chef-collection-range-with-flex-duo-oven-ne58h9970ws-aa/>
- [18] Domestic Appliance. n.d. *Bosch PIV831HB1E Series 6 80cm Induction Hob - Black*. [Online] Available at: <https://www.domestic-appliance.com/store/product/3106/bosch-piv831hb1e-series-6-80cm-induction-hob-black/>

- [19] Jackery. n.d. How many watts does a TV use? [Online] Available at: <https://www.jackery.com/blogs/knowledge/how-many-watts-does-a-tv-use>
- [20] MaxGaming. n.d. *Allied Gaming Stinger Ryzen 5 5500 RX 6600 Pelitietokone*. [Online] Available at: <https://www.maxgaming.fi/fi/poytatietokoneet/allied-gaming-stinger-ryzen-5-5500-rx-6600-pelitietokone>
- [21] OS Frisorartikler. n.d. *Valera Handy 1500 W*. [Online] Available at: <https://www.osfrisorartikler.fi/valera-handy-1500-w>
- [22] Tineco. n.d. *Tineco FLOOR ONE S5 Intelligent Nass- und Trockensauger*. [Online] Available at: <https://de-store.tineco.com/en-eu/products/tineco-floor-one-s5-intelligenter-nass-und-trockensauger>
- [23] FactorLED. n.d. *LED Katulamppu 50W Area Flex OSRAM Chip DURIS E 2835*. [Online] Available at: <https://www.factorled.com/fi/led-katuvalot-asuinaluekaytto/3470-led-katulamppu-50w-area-flex-osram-chip-duris-e-2835-8435770821111.html>
- [24] Fruugo. n.d. *2kW Electric Water Heater Hot Water Boiler Inner 50L*. [Online] Available at: <https://www.fruugo.fi/2kw-electric-water-heater-hot-water-boiler-inner-50l/p-260699866-570134852>
- [25] Patron Products. n.d. *E15 120V Electric Heater*. [Online] Available at: <http://www.patronproducts.com/electric-heaters/i/e15-120v-electric-heater>
- [26] MaxGaming. n.d. *MR105 300 Mbps N 4G LTE Reititin*. [Online] Available at: <https://www.maxgaming.fi/fi/langattomat-reititin/mr105-300-mbps-n-4g-lte-reititin>
- [27] Climate-Data.org. (n.d.). *Sunlight in Helsinki, Finland*. [online] Available at: <https://www.climate.top/finland/helsinki/sunlight/>

Appendix1 Mathematical model Code

```

# Import required libraries

import urllib.request      # For sending HTTP requests to get webpage data

from bs4 import BeautifulSoup # For parsing HTML content

from datetime import datetime, timedelta # For handling date and time

import pandas as pd        # For data processing (not used later but available for future
extensions)

from pyomo.environ import * # Import Pyomo optimization modeling environment

# -----

# 1. Crawl today's 24-hour electricity price data (buying price)

# -----

def get_html(url):
    """
    Retrieve HTML content from the given URL.

    Uses a specified User-Agent to disguise the request to avoid being blocked by the
    server.
    """
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
        "AppleWebKit/537.36 (KHTML, like Gecko) "

```

```

        "Chrome/134.0.0.0 Safari/537.36"
    }

    request = urllib.request.Request(url, headers=headers)
    response = urllib.request.urlopen(request, timeout=10)
    html = response.read().decode("utf-8")
    return html

def parse_html(html):
    """
    Parse the HTML content to extract electricity price data.

    - Uses BeautifulSoup to parse the HTML.
    - Finds the table on the page and extracts the date, time, and price from each row.
    - For rows where the date is not repeated, the current recorded date is used.
    - Note: The crawled price is in c/kWh (sis. alv.) and must be converted to €/kWh (divide
    by 100).

    Returns format: [[date, time, price], ...]
    """
    soup = BeautifulSoup(html, "html.parser")
    table = soup.find("table")

    if table is None:
        print("Could not find the electricity price table on the page; returning empty data.")
        return []

    rows = table.find_all("tr")[1:] # Skip header row
    data_list = []

```

```

    current_date = None # Keep track of the current date (for rows where the date is not
repeated)

for row in rows:

    cols = row.find_all("td")

    if len(cols) < 3:

        continue

    date_text = cols[0].text.strip()

    if date_text:

        current_date = date_text # Update if a new date appears

    time_text = cols[1].text.strip()

    price_text = cols[2].text.strip().replace(",",".") # Replace comma with dot for float
conversion

    try:

        price_value = float(price_text)

        # Unit conversion: from c/kWh to €/kWh (divide by 100)

        price_value = price_value / 100.0

    except Exception as e:

        print("Error converting price:", e)

        continue

    data_list.append([current_date, time_text, price_value])

return data_list

def fetch_electricity_prices():

    """

```

Crawl electricity price data and filter for today's data (from 0:00 to 24:00).

- Retrieves webpage data from the specified URL.
- Parses the webpage to get the electricity price table data.
- Determines today's time window (from 0:00 to the next day's 0:00) and stores data with hourly indices.

Returns a dictionary where keys are hour indices (1 to 24) and values are the corresponding electricity prices (in €/kWh).

```

"""
url = "https://oomi.fi/en/electricity/electricity-contracts/active/spot-price-of-electricity/"
try:
    html = get_html(url)
    data_list = parse_html(html)
    prices = {}
    now = datetime.now()
    # Define the start of the day (00:00 today)
    start_time = datetime(now.year, now.month, now.day, 0, 0, 0)
    # Define the end time as 00:00 of the next day (24 hours)
    end_time = start_time + timedelta(days=1)
    # Filter the parsed data to records within today's time window
    for row in data_list:
        date_text, time_text, price_value = row
        try:
            # Assuming the time format on the webpage is "dd.mm.yyyy HH:MM"
            timestamp = datetime.strptime(f"{date_text} {time_text}",
"%d.%m.%Y %H:%M")

```

```

    if start_time <= timestamp < end_time:

        # Calculate the hour index for the day (1 to 24)

        hour_index = int((timestamp - start_time).total_seconds() / 3600) + 1

        prices[hour_index] = price_value

    except Exception as e:

        print("Error parsing time:", e)

        continue

    return prices

except Exception as e:

    print("Error reading website data:", e)

    return {}

# Call the function to get today's buying price data

electricity_prices = fetch_electricity_prices()

# If data is missing, use the default value (0.20 €/kWh) as a substitute

default_price = electricity_prices.get(13, 0.20)

# Construct the buying price dictionary; if a time slot is missing, fill in with the default
price

Cbuy = {t: electricity_prices.get(t, default_price) for t in range(1, 25)}

# Set the selling price uniformly to 0.0066 €/kWh

Csell = {t: 0.0066 for t in range(1, 25)}

# Output the electricity prices (time slots 0-23 correspond to dictionary index t+1)

print("Crawled today's buying electricity prices (€/kWh):")

```

```

for t in range(0, 24):

    print(f"Time slot {t}: {Cbuy[t+1]:.4f} €/kWh")

print("Uniformly set selling price: 0.0066 €/kWh\n")

# -----

# 2. Define other model parameters

# -----

# Define the number of 15 household appliance loads

num_loads = 15

# Define the time range: 1 to 24 hours (representing 0-24 hours of the day)

T = list(range(1, 25))

# Define power (in kW), operation duration (in hours), and preferred start time (in hours)
for the 15 household appliances

P_values = {

    1: 0.2, # Fridge

    2: 1.5, # Air conditioner

    3: 2.2, # Washing machine (with water heating)

    4: 1.2, # Microwave oven

    5: 2.0, # Electric kettle

    6: 2.0, # Electric oven

    7: 1.6, # Induction cooker

```

8: 0.1, # Television
9: 0.5, # Desktop computer
10: 1.5, # Hair dryer
11: 1.0, # Vacuum cleaner
12: 0.05, # Electric fan
13: 2.0, # Electric water heater
14: 1.5, # Electric heater
15: 0.01 # Router
}

L_values = {
1: 24, # Fridge (continuous operation)
2: 8, # Air conditioner
3: 1, # Washing machine
4: 1, # Microwave oven
5: 1, # Electric kettle
6: 1, # Electric oven
7: 1, # Induction cooker
8: 4, # Television
9: 8, # Desktop computer
10: 1, # Hair dryer
11: 1, # Vacuum cleaner
12: 6, # Electric fan
13: 1, # Electric water heater

```
14: 3, # Electric heater  
15: 24 # Router (continuous operation)  
}
```

```
PrefStart_values = {  
    1: 1, # Fridge (non-schedulable)  
    2: 12, # Air conditioner  
    3: 9, # Washing machine  
    4: 12, # Microwave oven  
    5: 7, # Electric kettle  
    6: 18, # Electric oven  
    7: 12, # Induction cooker  
    8: 19, # Television  
    9: 9, # Desktop computer  
    10: 8, # Hair dryer  
    11: 11, # Vacuum cleaner  
    12: 15, # Electric fan  
    13: 6, # Electric water heater  
    14: 18, # Electric heater  
    15: 1 # Router (non-schedulable)  
}
```

```
# Penalty coefficients for deviation between actual start time and preferred start time
```

```
alpha_values = {i: 10 for i in range(1, num_loads+1)}
```

```

# Battery parameters (using Tesla Powerwall 3 as an example)

battery_capacity = 13.5    # Battery capacity (kWh)

SoC0 = 6.75                # Initial battery state of charge (kWh)

eta_charge = 0.90         # Charging efficiency

eta_discharge = 0.90      # Discharging efficiency

Pmax_batt = 11.5          # Maximum continuous battery charging/discharging power
(kW)

SoC_min = 0               # Minimum battery state of charge

SoC_max = battery_capacity # Maximum battery state of charge

# Base load data; assume base load is 0 for all periods

base_load = {t: 0 for t in T}

# Solar power generation forecast (in kW)

# Assume solar generation is active from 7 to 18 hours with symmetric output and
maximum at 12

res_gen = {t: max(0, 4 - 0.4 * abs(t - 12)) if 7 <= t <= 18 else 0.0 for t in T}

# HVAC (heating, ventilation, and air conditioning) parameters

T_set = 22 # Target indoor temperature (°C)

T_min = 20 # Minimum allowed indoor temperature (°C)

T_max = 24 # Maximum allowed indoor temperature (°C)

hvac_max = 5 # Maximum HVAC power (kW)

```

```

# -----

# 3. Build the Pyomo model

# -----

# Create a ConcreteModel instance

model = ConcreteModel()

# Define the set of time periods (1 to 24 hours)

model.T = Set(initialize=T)

# Define the set of appliance indices (1 to 15)

model.I = RangeSet(1, num_loads)

def T_start_rule(model, i):
    """
    Define the selectable start periods for each appliance.

    If an appliance's operation duration is 24 hours (e.g., fridge, router), then it can only start
    at period 1.

    Otherwise, the allowed start periods are those that ensure the operation fits within the
    24-hour window.

    """
    if L_values[i] == 24:
        return [1]
    else:

```

```

    return [t for t in T if t <= 24 - L_values[i] + 1]

# Add the allowed start periods for each appliance into a set

model.T_start = Set(model.I, initialize=T_start_rule)

def allowed_start_init(model):
    """
    Initialize the set of allowed (appliance, start period) pairs.
    """
    return ((i, t) for i in model.I for t in model.T_start[i])

model.allowed_start = Set(dimen=2, initialize=allowed_start_init)

# Pass parameters to the model

model.P = Param(model.I, initialize=P_values)           # Appliance power
model.L = Param(model.I, initialize=L_values)           # Appliance operation duration
model.PrefStart = Param(model.I, initialize=PrefStart_values) # Appliance preferred
start time
model.alpha = Param(model.I, initialize=alpha_values)   # Start time deviation
penalty coefficient

# Battery and other system parameters

model.Capacity = Param(initialize=battery_capacity)
model.SoC0 = Param(initialize=SoC0)
model.eta_charge = Param(initialize=eta_charge)
model.eta_discharge = Param(initialize=eta_discharge)

```

```

model.Pmax_batt = Param(initialize=Pmax_batt)

model.SoC_min = Param(initialize=SoC_min)

model.SoC_max = Param(initialize=SoC_max)

# Base load, solar generation, and buying/selling price data

model.BaseLoad = Param(model.T, initialize=base_load)

model.ResGen = Param(model.T, initialize=res_gen)

model.Cbuy = Param(model.T, initialize=Cbuy)

model.Csell = Param(model.T, initialize=Csell)

# Define binary variable y indicating whether an appliance starts at a given allowed start
period

model.y = Var(model.allowed_start, domain=Binary)

def x_rule(model, i, t):
    """
    Define an expression that indicates whether appliance i is running at time period t.

    For appliance i, if its chosen start time s satisfies  $s \leq t < s + \text{operation duration}$ , then it
    is running at time t.
    """
    return sum(model.y[i, s] for s in list(model.T_start[i]) if (s <= t) and (t < model.L[i] + s))

# Define expression x representing whether each appliance is running at each time period

model.x = Expression(model.I, model.T, rule=x_rule)

```

```

# Define continuous decision variables for battery charging, discharging, grid buying, and
grid selling power

model.p_batt_plus = Var(model.T, domain=NonNegativeReals) # Battery charging power

model.p_batt_minus = Var(model.T, domain=NonNegativeReals) # Battery discharging
power

model.p_grid_plus = Var(model.T, domain=NonNegativeReals) # Power purchased
from the grid

model.p_grid_minus = Var(model.T, domain=NonNegativeReals) # Power sold to the
grid

# Define grid power exchange limit parameter

Pmax_grid = 10

def grid_plus_limit_rule(model, t):
    """
    Limit the grid purchase power at time period t to be no more than Pmax_grid.
    """
    return model.p_grid_plus[t] <= Pmax_grid

model.grid_plus_limit_con = Constraint(model.T, rule=grid_plus_limit_rule)

def grid_minus_limit_rule(model, t):
    """
    Limit the grid selling power at time period t to be no more than Pmax_grid.
    """
    return model.p_grid_minus[t] <= Pmax_grid

model.grid_minus_limit_con = Constraint(model.T, rule=grid_minus_limit_rule)

```

```

# Define battery State of Charge (SoC) variable for time periods 1 to 25 (with 25
representing the end of period 24)

model.SoC = Var(RangeSet(1, 25), domain=NonNegativeReals)

# Define HVAC operation power variable with values between [0, hvac_max]

model.hvac = Var(model.T, domain=NonNegativeReals, bounds=(0, hvac_max))

# Define indoor temperature variable with values between [T_min, T_max]

model.T_in = Var(model.T, domain=Reals, bounds=(T_min, T_max))

# Define continuous variable for the actual start time of each appliance (used to compute
deviation)

model.start_time = Var(model.I, domain=NonNegativeReals)

# Define deviation variable to quantify the difference between actual and preferred start
times

model.dev = Var(model.I, domain=NonNegativeReals)

def one_start_rule(model, i):
    """
    Ensure that each appliance selects only one start period.
    """
    return sum(model.y[i, t] for t in model.T_start[i]) == 1

model.one_start_con = Constraint(model.I, rule=one_start_rule)

def start_time_def_rule(model, i):
    """

```

Define the actual start time for appliance i as the weighted sum of allowed start periods multiplied by the corresponding binary variable.

```

"""

return model.start_time[i] == sum(t * model.y[i, t] for t in model.T_start[i])

model.start_time_con = Constraint(model.I, rule=start_time_def_rule)

```

```
def dev_rule1(model, i):
```

```

"""

Define the deviation variable: deviation is at least the difference between the actual start
time and the preferred start time.

```

```

"""

return model.dev[i] >= model.start_time[i] - model.PrefStart[i]

model.dev_con1 = Constraint(model.I, rule=dev_rule1)

```

```
def dev_rule2(model, i):
```

```

"""

Define the deviation variable: deviation is at least the difference between the preferred
start time and the actual start time.

```

```

"""

return model.dev[i] >= model.PrefStart[i] - model.start_time[i]

model.dev_con2 = Constraint(model.I, rule=dev_rule2)

```

```
def power_balance_rule(model, t):
```

```

"""

Define the energy balance constraint for each time period:

```

Grid buying - grid selling + battery charging - battery discharging + solar generation
 must equal the base load + appliance load (power multiplied by running status) + HVAC
 power.

```
"""
```

```
return (
```

```
    model.p_grid_plus[t]
```

```
    - model.p_grid_minus[t]
```

```
    + model.p_batt_plus[t]
```

```
    - model.p_batt_minus[t]
```

```
    + model.ResGen[t]
```

```
) == (
```

```
    model.BaseLoad[t]
```

```
    + sum(model.P[i] * model.x[i, t] for i in model.I)
```

```
    + model.hvac[t]
```

```
)
```

```
model.power_balance_con = Constraint(model.T, rule=power_balance_rule)
```

```
def soc_update_rule(model, t):
```

```
    """
```

```
    Define the battery SoC update equation:
```

```
    Next period's SoC = current SoC + charged energy (multiplied by charging efficiency) -  

    discharged energy (divided by discharging efficiency)
```

```
    Note: This constraint is defined only for periods 1 to 24.
```

```
    """
```

```

if t < 25:

    return model.SoC[t+1] == model.SoC[t] + model.eta_charge * model.p_batt_plus[t] -
(1/model.eta_discharge) * model.p_batt_minus[t]

else:

    return Constraint.Skip

model.soc_update_con = Constraint(RangeSet(1, 24), rule=soc_update_rule)

```

```

def soc_limits_rule(model, t):

    """

    Ensure that the battery SoC at each period is within the range [SoC_min, SoC_max].

    """

    return (model.SoC_min, model.SoC[t], model.SoC_max)

model.soc_limits_con = Constraint(RangeSet(1, 25), rule=soc_limits_rule)

```

```

def batt_plus_limit_rule(model, t):

    """

    Limit the battery charging power at time period t to be no more than the maximum
power.

    """

    return model.p_batt_plus[t] <= model.Pmax_batt

model.batt_plus_con = Constraint(model.T, rule=batt_plus_limit_rule)

```

```

def batt_minus_limit_rule(model, t):

    """

```

Limit the battery discharging power at time period t to be no more than the maximum power.

```
"""
```

```
return model.p_batt_minus[t] <= model.Pmax_batt
```

```
model.batt_minus_con = Constraint(model.T, rule=batt_minus_limit_rule)
```

```
def switching_limit_rule(model, t):
```

```
"""
```

At time period t , ensure that the battery does not charge and discharge simultaneously, i.e., the sum of charging and discharging power does not exceed the maximum power.

```
"""
```

```
return model.p_batt_plus[t] + model.p_batt_minus[t] <= model.Pmax_batt
```

```
model.switching_limit = Constraint(model.T, rule=switching_limit_rule)
```

```
# Parameters for the effect of ambient temperature and HVAC on indoor temperature
```

```
ambient_temp = 18    # Ambient temperature (°C)
```

```
cooling_effect = 0.5 # Cooling effect of HVAC
```

```
def temp_dynamics_rule(model, t):
```

```
"""
```

Establish the dynamic model for indoor temperature:

- At the initial period, indoor temperature is set to the target temperature T_{set} .
- For subsequent periods, the indoor temperature is influenced by the previous period's temperature and the current HVAC cooling effect,

```

    weighted 0.8 and 0.2 respectively.
    """

    if t == 1:

        return model.T_in[t] == T_set

    else:

        return model.T_in[t] == 0.8 * model.T_in[t-1] + 0.2 * (ambient_temp +
cooling_effect * model.hvac[t])

model.temp_dynamics_con = Constraint(model.T, rule=temp_dynamics_rule)

def hvac_ramp_rule1(model, t):
    """
    Limit the increase in HVAC power between consecutive periods to no more than 1 kW.
    """

    if t == 1:

        return Constraint.Skip

    else:

        return model.hvac[t] - model.hvac[t-1] <= 1

model.hvac_ramp_con1 = Constraint(model.T, rule=hvac_ramp_rule1)

def hvac_ramp_rule2(model, t):
    """
    Limit the decrease in HVAC power between consecutive periods to no more than 1 kW.
    """

    if t == 1:

```

```

    return Constraint.Skip

else:

    return model.hvac[t-1] - model.hvac[t] <= 1

model.hvac_ramp_con2 = Constraint(model.T, rule=hvac_ramp_rule2)

def objective_rule(model):
    """
    Define the objective function:
    - Energy cost: cost of purchasing electricity minus revenue from selling electricity (each
    multiplied by the corresponding price).
    - Discomfort penalty: the sum over all appliances of the deviation between actual start
    time and preferred start time, multiplied by penalty coefficient alpha.
    The objective is to minimize the total cost (energy cost + discomfort penalty).
    """
    energy_cost = sum(model.Cbuy[t] * model.p_grid_plus[t] - model.Csell[t] *
    model.p_grid_minus[t] for t in model.T)
    discomfort = sum(model.alpha[i] * model.dev[i] for i in model.I)
    return energy_cost + discomfort

model.obj = Objective(rule=objective_rule, sense=minimize)

def init_soc_rule(model):
    """
    Define the initial battery SoC constraint.
    """
    return model.SoC[1] == model.SoC0

```

```

model.init_soc_con = Constraint(expr=init_soc_rule(model))

# -----

# 4. Solve the model and output the results

# -----

# Solve the model using the GLPK solver with log output turned off (tee=False)

solver = SolverFactory('glpk')

results = solver.solve(model, tee=False)

# Output the objective function value (total cost)

print("Objective function value (total cost):", value(model.obj))

print("\nAppliance start times and deviations:")

# Output the actual start time and deviation for each appliance

for i in model.I:

    start = value(model.start_time[i])

    deviation = value(model.dev[i])

    print(f"Appliance {i}: Preferred start = {value(model.PrefStart[i])}, Actual start =
{start:.2f}, Deviation = {deviation:.2f}")

# Output battery SoC (periods 0-24 correspond to internal periods 1-25)

print("\nBattery SoC:")

for t in model.SoC:

```

```

# Internal t=1 corresponds to period 0, and t=25 corresponds to period 24

print(f"Period {t-1}: SoC = {value(model.SoC[t]):.2f} kWh")

# Output HVAC power and indoor temperature (periods 0-23 correspond to internal
periods 1-24)

print("\nHVAC power and Indoor Temperature:")

for t in model.T:

    print(f"Period {t-1}: HVAC = {value(model.hvac[t]):.2f} kW, Indoor Temp =
{value(model.T_in[t]):.2f} °C")

# Output grid power exchange (positive = buying, negative = selling) for periods 0-23

print("\nGrid power exchange (positive = buying, negative = selling):")

for t in model.T:

    net_grid = value(model.p_grid_plus[t]) - value(model.p_grid_minus[t])

    print(f"Period {t-1}: Net Grid Power = {net_grid:.2f} kW")

# Output solar panel real-time power generation (periods 0-23)

print("\nSolar panel real-time power generation:")

for t in model.T:

    print(f"Period {t-1}: Solar Power = {res_gen[t]:.2f} kW")

# Output total load (15 appliances + HVAC) for periods 0-23

print("\nTotal load of 15 appliances + HVAC:")

for t in model.T:

```

```

total_load = sum(value(model.P[i] * model.x[i, t] for i in model.I) +
value(model.hvac[t])

print(f"Period {t-1}: Total Load = {total_load:.2f} kW")

```

Appendix 2 No Mathematical model

```
def get_html(url):
```

```
    """
```

Retrieve the HTML content of a webpage given a URL.

Uses a specified User-Agent to disguise the request to avoid being blocked by the server.

```
    """
```

```
    headers = {
```

```
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
```

```
            "AppleWebKit/537.36 (KHTML, like Gecko) "
```

```
            "Chrome/134.0.0.0 Safari/537.36"
```

```
    }
```

```
    request = urllib.request.Request(url, headers=headers)
```

```
    response = urllib.request.urlopen(request, timeout=10)
```

```
    html = response.read().decode("utf-8")
```

```
    return html
```

```
def parse_html(html):
```

```
    """
```

Parse the HTML content of a webpage to extract electricity price data.

- Uses BeautifulSoup to parse HTML.
- Finds the table in the page and extracts the date, time, and price from each row.
- For rows where the date is not repeated, uses the last recorded date.
- Note: The crawled prices are in c/kWh (sis. alv.) and need to be converted to €/kWh (divide by 100).

Returns: a list in the format [[date, time, price], ...].

```
"""
```

```
soup = BeautifulSoup(html, "html.parser")
```

```
table = soup.find("table")
```

```
if table is None:
```

```
    print("Could not find the electricity price table on the page, returning empty data.")
```

```
    return []
```

```
rows = table.find_all("tr")[1:] # Skip the header row
```

```
data_list = []
```

```
current_date = None # Record the current date (for rows where the date is not repeated)
```

```
for row in rows:
```

```
    cols = row.find_all("td")
```

```
    if len(cols) < 3:
```

```
        continue
```

```
    date_text = cols[0].text.strip()
```

```
    if date_text:
```

```
        current_date = date_text # Update if a new date is provided
```

```
    time_text = cols[1].text.strip()
```

```

    price_text = cols[2].text.strip().replace(",", ".") # Replace comma with dot for float
conversion
try:

    price_value = float(price_text)

    # Unit conversion: convert from c/kWh to €/kWh (divide by 100)

    price_value = price_value / 100.0

except Exception as e:

    print("Error converting price:", e)

    continue

    data_list.append([current_date, time_text, price_value])

return data_list

def fetch_electricity_prices():
    """
    Crawl electricity price data and filter out the data for the current day (from 0:00 to
    24:00).

    - Fetch webpage data from the specified URL.

    - Parse the webpage to get the electricity price table.

    - Determine the time window for the current day (from midnight to the next midnight)
    and store the data indexed by hour.

    Returns: a dictionary with keys as hour indices (1 to 24) and values as the corresponding
    electricity prices (in €/kWh).
    """
    url = "https://oomi.fi/en/electricity/electricity-contracts/active/spot-price-of-electricity/"
    try:

```

```

html = get_html(url)

data_list = parse_html(html)

prices = {}

now = datetime.now()

# Define the start time of the current day (midnight)
start_time = datetime(now.year, now.month, now.day, 0, 0, 0)

# Define the end time as the next day's midnight (24 hours)
end_time = start_time + timedelta(days=1)

# Loop through the parsed data and filter records within today's time window
for row in data_list:

    date_text, time_text, price_value = row

    try:

        # Assume the time format on the webpage is "dd.mm.yyyy HH:MM"
        timestamp = datetime.strptime(f"{date_text} {time_text}",
"%d.%m.%Y %H:%M")

        if start_time <= timestamp < end_time:

            # Calculate the hour index (1 to 24) for this timestamp within the day
            hour_index = int((timestamp - start_time).total_seconds() / 3600) + 1

            prices[hour_index] = price_value

    except Exception as e:

        print("Error parsing time:", e)

        continue

return prices

except Exception as e:

```

```

    print("Error fetching website data:", e)

    return {}

# Call the function to get the purchase electricity price data for the current day (although
not used in the current model, this part is retained)

electricity_prices = fetch_electricity_prices()

# If data is missing, fill it with a default value (0.20 €/kWh)

default_price = electricity_prices.get(13, 0.20)

# Construct the purchase electricity price dictionary; if an hour is missing, use the default
price

Cbuy = {t: electricity_prices.get(t, default_price) for t in range(1, 25)}

# The selling price of electricity is uniformly set to 0.0066 €/kWh

Csell = {t: 0.0066 for t in range(1, 25)}

print("Crawled purchase electricity prices for the current day (€/kWh):")

for t in range(0, 24):

    print(f"Hour {t}: {Cbuy[t+1]:.4f} €/kWh")

print("Uniformly set selling price: 0.0066 €/kWh\n")

# -----

# 2. Define other model parameters

# -----

# Define the number of 15 household appliance loads

```

```
num_loads = 15

# Define the time range: hours 1 to 24 (representing 0-24 hours of the day)
T = list(range(1, 25))

# Power (kW), operation duration (hours), and preferred start time (hour) for 15 types of
household appliances
P_values = {
    1: 0.2, # Refrigerator
    2: 1.5, # Air conditioner
    3: 2.2, # Washing machine (when heating water)
    4: 1.2, # Microwave
    5: 2.0, # Electric kettle
    6: 2.0, # Electric oven
    7: 1.6, # Induction cooker
    8: 0.1, # Television
    9: 0.5, # Desktop computer
    10: 1.5, # Hair dryer
    11: 1.0, # Vacuum cleaner
    12: 0.05, # Electric fan
    13: 2.0, # Electric water heater
    14: 1.5, # Electric heater
    15: 0.01 # Router
}
```

```
L_values = {  
    1: 24, # Refrigerator (continuous operation)  
    2: 8, # Air conditioner  
    3: 1, # Washing machine  
    4: 1, # Microwave  
    5: 1, # Electric kettle  
    6: 1, # Electric oven  
    7: 1, # Induction cooker  
    8: 4, # Television  
    9: 8, # Desktop computer  
    10: 1, # Hair dryer  
    11: 1, # Vacuum cleaner  
    12: 6, # Electric fan  
    13: 1, # Electric water heater  
    14: 3, # Electric heater  
    15: 24 # Router (continuous operation)  
}
```

```
PrefStart_values = {  
    1: 1, # Refrigerator (non-schedulable)  
    2: 12, # Air conditioner  
    3: 9, # Washing machine  
    4: 12, # Microwave
```

```

5: 7, # Electric kettle

6: 18, # Electric oven

7: 12, # Induction cooker

8: 19, # Television

9: 9, # Desktop computer

10: 8, # Hair dryer

11: 11, # Vacuum cleaner

12: 15, # Electric fan

13: 6, # Electric water heater

14: 18, # Electric heater

15: 1 # Router (non-schedulable)
}

# Penalty coefficients used to penalize deviations between the actual start time and the
# preferred start time

alpha_values = {i: 10 for i in range(1, num_loads+1)}

# Base load data, assuming the base load is 0 for all time periods

base_load = {t: 0 for t in T}

# -----

# 3. Build the Pyomo Model

# -----

```

```

# Create a concrete model instance

model = ConcreteModel()

# Define the set of time periods (hours 1 to 24)

model.T = Set(initialize=T)

# Define the set of appliance indices (1 to 15)

model.I = RangeSet(1, num_loads)

def T_start_rule(model, i):
    """
    Define the available start time periods for each appliance.

    If the appliance operates continuously for 24 hours (e.g., refrigerator, router), then it can
    only start at time period 1.

    Otherwise, the available start time periods are those that ensure the operation duration
    fits within the 24-hour day.

    """
    if L_values[i] == 24:
        return [1]
    else:
        return [t for t in T if t <= 24 - L_values[i] + 1]

# Add the allowed start time periods for each appliance into a set

model.T_start = Set(model.I, initialize=T_start_rule)

def allowed_start_init(model):

```

```

"""

Initialize the set of allowed (appliance, start time period) pairs.

"""

return ((i, t) for i in model.I for t in model.T_start[i])

model.allowed_start = Set(dimen=2, initialize=allowed_start_init)

# Pass parameters to the model

model.P = Param(model.I, initialize=P_values)      # Appliance power (kW)
model.L = Param(model.I, initialize=L_values)      # Appliance operation duration
(hours)

model.PrefStart = Param(model.I, initialize=PrefStart_values) # Preferred start time for
appliances (hour)

model.alpha = Param(model.I, initialize=alpha_values) # Penalty coefficient for start
time deviation

# Define binary variable y, indicating whether an appliance is scheduled to start at an
allowed start time period

model.y = Var(model.allowed_start, domain=Binary)

def x_rule(model, i, t):
    """

    Define an expression indicating whether appliance i is operating at time t.

    For appliance i, if its start time s satisfies  $s \leq t < s + \text{operation duration}$ , then it is
    considered to be operating at time t.

    """

```

```

    return sum(model.y[i, s] for s in list(model.T_start[i]) if (s <= t) and (t < model.L[i] + s))

# Define the expression x, representing whether each appliance is operating at each time
period.

model.x = Expression(model.I, model.T, rule=x_rule)

# Define HVAC operation power variable, with values between [0, hvac_max]

hvac_max = 5 # Maximum HVAC power (kW)

model.hvac = Var(model.T, domain=NonNegativeReals, bounds=(0, hvac_max))

# Define indoor temperature variable, with values between [T_min, T_max]

T_set = 22 # Target indoor temperature (°C)

T_min = 20 # Minimum allowed indoor temperature (°C)

T_max = 24 # Maximum allowed indoor temperature (°C)

model.T_in = Var(model.T, domain=Reals, bounds=(T_min, T_max))

# Define a continuous variable for the actual start time of each appliance (used for
calculating deviation)

model.start_time = Var(model.I, domain=NonNegativeReals)

# Define deviation variable to quantify the gap between the actual start time and the
preferred start time

model.dev = Var(model.I, domain=NonNegativeReals)

def one_start_rule(model, i):
    """
    Ensure that each appliance selects only one start time period.
    """
    return sum(model.y[i, t] for t in model.T_start[i]) == 1

```

```
model.one_start_con = Constraint(model.I, rule=one_start_rule)
```

```
def start_time_def_rule(model, i):
```

```
    """
```

Define the actual start time for appliance i as the weighted sum of all allowed start time periods multiplied by the corresponding binary variable.

```
    """
```

```
    return model.start_time[i] == sum(t * model.y[i, t] for t in model.T_start[i])
```

```
model.start_time_con = Constraint(model.I, rule=start_time_def_rule)
```

```
def dev_rule1(model, i):
```

```
    """
```

Define the deviation variable: the deviation must be at least the difference between the actual start time and the preferred start time.

```
    """
```

```
    return model.dev[i] >= model.start_time[i] - model.PrefStart[i]
```

```
model.dev_con1 = Constraint(model.I, rule=dev_rule1)
```

```
def dev_rule2(model, i):
```

```
    """
```

Define the deviation variable: the deviation must be at least the difference between the preferred start time and the actual start time.

```
    """
```

```
    return model.dev[i] >= model.PrefStart[i] - model.start_time[i]
```

```
model.dev_con2 = Constraint(model.I, rule=dev_rule2)
```

Since the battery and grid interaction parts have been removed, energy balance constraints are not established here.

It is assumed that all loads are met by an external power source, so we only focus on appliance scheduling and HVAC control.

```
def temp_dynamics_rule(model, t):
```

```
    """
```

```
    Establish the dynamic model for indoor temperature:
```

```
    - At the initial time, the indoor temperature is set to the target temperature T_set.
```

```
    - For subsequent time periods, the indoor temperature is influenced by the previous temperature and the current HVAC cooling effect, weighted by 0.8 and 0.2 respectively.
```

```
    """
```

```
    ambient_temp = 18    # Ambient temperature (°C)
```

```
    cooling_effect = 0.5  # HVAC cooling effect
```

```
    if t == 1:
```

```
        return model.T_in[t] == T_set
```

```
    else:
```

```
        return model.T_in[t] == 0.8 * model.T_in[t-1] + 0.2 * (ambient_temp + cooling_effect * model.hvac[t])
```

```
    model.temp_dynamics_con = Constraint(model.T, rule=temp_dynamics_rule)
```

```
def hvac_ramp_rule1(model, t):
```

```
    """
```

Limit the upward ramp rate of HVAC power, ensuring that the change between consecutive time periods does not exceed 1 kW.

```
"""
```

```
if t == 1:
```

```
    return Constraint.Skip
```

```
else:
```

```
    return model.hvac[t] - model.hvac[t-1] <= 1
```

```
model.hvac_ramp_con1 = Constraint(model.T, rule=hvac_ramp_rule1)
```

```
def hvac_ramp_rule2(model, t):
```

```
"""
```

Limit the downward ramp rate of HVAC power, ensuring that the change between consecutive time periods does not exceed 1 kW.

```
"""
```

```
if t == 1:
```

```
    return Constraint.Skip
```

```
else:
```

```
    return model.hvac[t-1] - model.hvac[t] <= 1
```

```
model.hvac_ramp_con2 = Constraint(model.T, rule=hvac_ramp_rule2)
```

```
def objective_rule(model):
```

```
"""
```

```
    Define the objective function:
```

The goal is to minimize the sum of the deviations between the actual start times and the preferred start times for all appliances, weighted by the penalty coefficient alpha (i.e., discomfort penalty).

```

"""

discomfort = sum(model.alpha[i] * model.dev[i] for i in model.I)

return discomfort

model.obj = Objective(rule=objective_rule, sense=minimize)

# -----

# 4. Solve the model and output the results

# -----

# Use the GLPK solver to solve the model, with logging disabled (tee=False)

solver = SolverFactory('glpk')

results = solver.solve(model, tee=False)

# Calculate and output the total cost at the top of the results (based on the crawled purchase
electricity prices and the total load in each time period)

total_cost = 0

for t in model.T:

    # Total load for each time period = appliance load (power * operating status) + HVAC
load

    load_t = sum(value(model.P[i] * model.x[i, t]) for i in model.I) + value(model.hvac[t])

    total_cost += Cbuy[t] * load_t

print("Total cost (based on purchase electricity prices): {:.2f} €".format(total_cost))

```

```

# Output the objective function value (total discomfort penalty)

print("Objective function value (total discomfort penalty):", value(model.obj))

print("\nAppliance start times and deviations:")

# Output the actual start time and deviation for each appliance

for i in model.I:

    start = value(model.start_time[i])

    deviation = value(model.dev[i])

    print(f"Appliance {i}: Preferred start = {value(model.PrefStart[i])}, Actual start =
{start:.2f}, Deviation = {deviation:.2f}")

# Output HVAC power and indoor temperature (time periods 0-23 corresponding to
internal time periods 1-24)

print("\nHVAC Power and Indoor Temperature:")

for t in model.T:

    print(f"Time period {t-1}: HVAC = {value(model.hvac[t]):.2f} kW, Indoor Temperature
= {value(model.T_in[t]):.2f} °C")

# Output the total load of 15 appliances + HVAC (time periods 0-23)

print("\nTotal Load of 15 Appliances + HVAC:")

for t in model.T:

    total_load = sum(value(model.P[i] * model.x[i, t]) for i in model.I) +
value(model.hvac[t])

    print(f"Time period {t-1}: Total Load = {total_load:.2f} kW")

```