Research Article

Large Language Model Interface for Home Energy Management Systems

François Michelon[1], Yihong Zhou[2], Thomas Morstyn[3]

1. Ecole Centrale de Nantes, France; 2. School of Engineering, University of Edinburgh, United Kingdom; 3. Department of Engineering Science, University of Oxford, United Kingdom

Home Energy Management Systems (HEMSs) help households tailor their electricity usage based on power system signals such as energy prices. This technology helps to reduce energy bills and offers greater demand-side flexibility that supports the power system stability. However, residents who lack a technical background may find it difficult to use HEMSs effectively, because HEMSs require well-formatted parameterization that reflects the characteristics of the energy resources, houses, and users' needs. Recently, Large-Language Models (LLMs) have demonstrated an outstanding ability in language understanding. Motivated by this, we propose an LLM-based interface that interacts with users to understand and parameterize their "badly-formatted answers", and then outputs well-formatted parameters to implement an HEMS. We further use Reason and Act method (ReAct) and few-shot prompting to enhance the LLM performance. Evaluating the interface performance requires multiple user–LLM interactions. To avoid the efforts in finding volunteer users and reduce the evaluation time, we additionally propose a method that uses another LLM to simulate users with varying expertise, ranging from knowledgeable to non-technical. By comprehensive evaluation, the proposed LLM-based HEMS interface achieves an average parameter retrieval accuracy of 88%, outperforming benchmark models without ReAct and/or few-shot prompting.

1. Introduction

In the context of global decarbonization, the proportion of renewable energy sources such as wind and solar is increasing[1]. However, these renewable sources have intermittent generation, which challenges power system stability. Demand-side flexibility is acknowledged as an effective solution and is currently being implemented in practice.

 doi.org/10.32388/EVPWM7

Home Energy Management Systems (HEMS) represent a promising approach to enable demand-side flexibility[2], which provides smart energy management for electricity generation, storage, and consumption in smart houses while respecting users' preferences[3][4]. There have been extensive studies on HEMS. In[5] proposed an open-source software platform for integrated modeling, control, and simulation of smart local energy systems, including HEMS. In[6] proposed a method that includes users as key members of the HEMS, allowing them to plan the intended power consumption and manage real-time deviations. In[7] presented a grid-integrated system for energy management for smart demand-side energy management. The effectiveness of existing HEMS models was extensively discussed in[8]. In the literature, there has been extensive study of algorithm design for HEMS. However, the practical implementation of HEMS requires input parameters that represent the physical characteristics of home appliances, description of human behavior, and the user's needs for comfort. The required number of parameters can be high, and these must be well-formatted to be understandable to the HEMS algorithms. This process can be time-consuming and demotivates the use of HEMS[9]. Furthermore, non-expert users, particularly the elderly and those with limited literacy, may struggle to identify the correct and well-formatted parameters.

A new opportunity is presented by Large Language Models (LLMs), which perform particularly well in Natural Language Processing (NLP) tasks. This motivates us to use LLMs to retrieve well-formatted parameters required for HEMS, by interacting with users in an easy-understanding and natural-language way. Recently, LLMs have found applications for several power system tasks, including modeling nonlinear power electronic circuits[10], detecting insulator defect[11], scheduling charging of electric vehicles (EVs)[12], and performing power system simulation based on a user requests[13][14]. Using LLMs to facilitate the practical implementation of HEMS is a daily topic. Ref.[15] presented an LLM-based system performing a discrete set of actions to manipulate home devices based on user instructions. Ref.[16] proposed a smart home assistant that creates and controls home automation routines based on data sources.[15] and[16] implement systems that control home devices based on user instructions. However, the proposed systems require a lot of user interactions for simple tasks that will not permit efficient energy management. In this study, we propose an intuitive and flexible LLM-based interface that helps retrieve the necessary and well-formatted parameters for a HEMS to minimize household energy bills while fulfilling numerous user requirements. Specifically, we make the following contributions:

1. We propose and develop a user-friendly LLM-based HEMS interface designed to fetch information through user–LLM interaction and convert it into a well-formatted form for optimization. This interaction, conducted in natural language, minimizes the effort and complexity of using HEMS.
2. We propose the use of ReAct and few-shot prompting to improve the LLM performance, and this improvement is verified by case studies.
3. Assessing the interface performance involves conducting multiple user–LLM tests. To minimize the effort in recruiting volunteers and decrease the evaluation time, we propose a method that employs an additional LLM to emulate users with diverse levels of expertise, from highly knowledgeable to non-technical.

The manuscript is organized as follows. Section 2 describes HEMS and lists the main necessary inputs required for HEMS. Section 3 introduces the proposed LLM-based interface and depicts its key features such as ReAct and few-shot prompting. Section 4 describes LLM users by defining the cosine similarity metrics and the three user difficulty levels that will test the interface. Section 5 presents the case study of the interface and the obtained results, showing that HEMS can be widespread when coupled with an intuitive and flexible interface. Section 6 concludes and discusses directions for future research.

2. Home Energy Management System

We consider a HEMS that minimizes the energy bill, by controlling devices such as electric heating, electric vehicle charging, user schedule, energy provider, solar panels, and simulation period and place (from which we obtain weather forecast and electricity price time series). This simple modelling can be extended to consider more detailed aspects like a more complex user schedule in future work.

2.1. Model definition

The HEMS problem we consider optimizes the energy usage of flexible appliances over a time horizon to minimize the user's cost of energy, given constraints on how the appliances can be used. In our HEMS model, the user can possess several electric vehicles (EVs), the user has a daily schedule that defines when the user is home or not ($t_{arrival}$ when the user comes back to his accommodation and $t_{departure}$ when the user leaves). We assume that while parked at home the EV can be charged. When the user comes back from work, his EV has a low battery, and when the user leaves in the morning it is

assumed his EV must be fully charged. In addition, the user also wants the house temperature to be between a minimum and maximum value. The model is more precisely described in the section 7.1 of the appendix.

3. Methodology

In most systems, users need to parameterize the HEMS by themselves (Fig. 1 step A1), which takes lots of time and can lead to input errors and control errors (Fig. 1 step A2). On the one hand, asking the user to provide too many precise parameters like EV capacity and usage schedule, will diminish the HEMS democratization as the learning curve will be too steep. On the other hand, asking for too little information will result in a non-tailored consumption energy schedule.

LLMs have exhibited versatile language understanding and prominent information retrieval ability in various contexts[12][14]. Therefore, we propose to include LLM as an extra interface between users and HEMS. This paper considers a system with three main entities: the Home Energy Management System (HEMS), the LLM agent, and the user. We aim to design an LLM-based interface that retrieves all necessary user information, stores it in a correct format, and then sends the formatted inputs to the HEMS, to obtain the optimal electricity energy schedule. Fig. 1 shows the high-level flowchart of our interface from user–LLM interaction where the user can provide badly-formatted parameters to the LLM like a textual description of its schedule instead of a formatted date (step B1) to the parametrization of the HEMS with the translated user preferences into HEMS formatted inputs (step B2) and the control of user appliances (step B3).

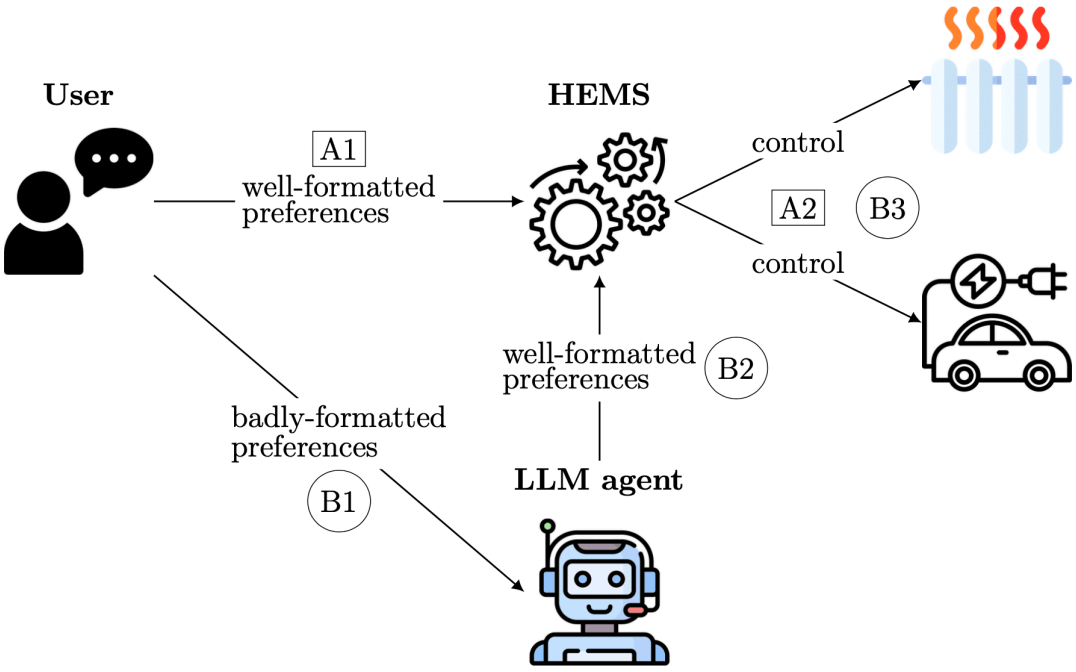


Figure 1. High-level interface LLM-HEMS flowchart Sub-flowchartA, current Home Energy Management System (**HEMS**) requires well-formatted parameters from the **User**. After that, the **HEMS** controls the different appliances of the user's house to optimize the energy schedule while respecting users' preferences. Sub-flowchart B illustrates the proposed LLM-integrated system. The **LLM agent** first interacts with the **user** by asking well-oriented questions. The **LLM agent** extracts the badly formatted preferences from the user's answers and converts them to fit the **HEMS** requirements. Finally, the **HEMS** can control the user's appliances as the user wants.

3.1. Integration of LLM

To use the HEMS problem defined in section 2, one needs to know the user's schedule (arrival and leaving time), its preferences (minimum and maximum temperatures, start and end date of simulation), and characteristics like the number of electric vehicles he owns (see Table 4 in the appendix). This is the job of an LLM-based agent that we will define in this section.

The interface needs to understand user stylometry, user dwellings specificities, user energy management preferences, user energy Home Energy Management System (HEMS) understanding. The user answers can contain vague information. Classical NLP models are not capable of performing

doi.org/10.32388/EVPWM7

on these wide ranges of complicated tasks or require huge efforts in fine-tuning, that is why using an LLM is required.

In sub flowchart B of Fig. 1, the LLM-agent interacts with the user (step B1) and the HEMS (step B2). The LLM agent must be able to communicate with both entities. To do so, we provide communication functions or tools that are all stored in a toolkit object. Each tool is described in the LLM agent instantiation prompt with the function of the tool, its inputs, and outputs. We want our LLM agent to be able to ask questions to a user based on a task, retrieve the wanted information from the user's answer, and store the retrieved parameters. To do so, we allow our LLM agent to use several tools such as ask_user and $store$ which will enable better communication between the user and the LLM agent and between the HEMS and the LLM agent. To properly use those tools, the LLM agent will have to generate function calls by providing the name of a tool and its input. The LLM agent will have to choose in the tool list to which it has access, by understanding the most appropriate tool to solve the problem it is currently working on. This method is known as function calling[17].

We need our LLM agent to generate very precise texts for function calling, to do so we will provide examples which is a common practice. The few-shot prompting technique[18] consists in adding one or several task-solving examples, giving more information to the LLM agent after the agent prompt template (see left of Fig. 2 in the appendix). This will give more context to the LLM agent and diminish hallucination.

3.2. ReAct

Many methods of prompt engineering have been developed to get the most out of LLMs in NLP tasks. In this study we have chosen to focus on ReAct[19]. This method gives a ReAct prompt to the LLM to generate both reasoning traces and task-specific actions in an interleaved manner. This allows the system to perform dynamic reasoning to create, maintain, and adjust plans for acting while enabling interaction with external environments to incorporate additional information into the reasoning. The LLM agent will break down its tasks into smaller goals to increase its success rate (see right of Fig. 2).

In the agent prompt template (see Fig. 10 in section B.1 of the appendix) given to the LLM agent type $React + example$, we first describe the main goal of the LLM agent (line 1), the tools and their description (lines 1 and 5), and then the format rules it must follow for function calling (lines 3 to 13) and for ReAct (lines 15 to 27). After receiving a task, the LLM agent will have to rephrase it, then think about the first objective it must fulfill, and act to complete it by generating a correct JSON object that

will be parsed to call a function. If the JSON blob is correct, the output of the function will be given to the LLM agent as an observation. The process continues until the LLM agent thinks that the task is completed.

4. LLM user

For testing purposes, we have designed an LLM to operate in the place of a real user to gain testing time and model various behaviors. In the following paragraphs, we will refer to this LLM as the user. To simulate the diversity of LLM agent–user interaction, we set a high LLM temperature (0.8 in our tests) that allows for more diversity by increasing the weights of less likely LLM-generated words. The LLM user is given personal information that an LLM agent will try to retrieve by asking questions. The user prompt template is fairly simple and follows the same rules as the agent prompt template, such as goal description, context, and rules.

As in a real-life scenario, user behavior can be very different from person to person. We tried to model different users by giving them different prompts that will imply different answers to the LLM agent's questions. To design the user prompts, we used synonyms and changed the formats while also adding noisy information. We have defined three levels of difficulty for the LLM agent. In the easy mode, the user's answers are considered simple and to be exactly what the agent asked for (ex: date format). We define user precision as the quality of the user's answers. It measures how badly formatted the answers of the user are (see Fig. 1). The user precision diminishes with the user difficulty mode. The user precision evaluation methodology is described in section A.3 of the appendix.

Difficulty	Easy (E)	Medium (M)	Hard (H)
Score	0.98	0.94	0.84

Table 1. Average cosine similarity score per difficulty level

5. Results

In this section, we present results demonstrating the performance of the proposed LLM-based HEMS interface. First, we explain and justify the settings used for the HEMS, the LLM user, and the LLM

agent in our case studies. The experimental setup of the study is detailed in section A.4 of the appendix.

We have chosen to make the LLM agent retrieve parameters of different types to check its generalization capability (see Tab. 4). To test the added value of ReAct and the few-shot prompting, we developed three agent types: Act, $Act + example$, and $ReAct + example$. All three use function calling to interact with the user and the HEMS, so they can all "Act" by deciding to use a communication tool. To evaluate the importance of function calling, we developed agent type Act that does not receive any example of how it should solve tasks. To evaluate the importance of ReAct, we implemented $Act + example$ agent type. It receives an example in its prompt of how it should solve tasks but does not follow the ReAct methodology which forces the agent to think and adapt to the results of its last action. An HEMS parametrization systems result is presented in Fig. 5 in the appendix. It refers to a user-optimal energy management scenario. Based on our price data the user paid around 16£. If he had consumed the same amount of energy without taking into account energy price fluctuation, he would have paid around 31£ which is a 48% energy cost reduction.

We have tested our interface with three different LLM versions of MistralAI's LLMs on 20 tests for each difficulty level[20]. For notation simplicity, we respectively denote models OpenHermes-2.5-Mistral-7B, Mistral-7B-Instruct-v0.2, and Mistral-7B-Instruct-v0.3 as V1, V2, V3. Each LLM agent type was tested for all user difficulty modes Easy, Medium, and Hard (denoted as E, M, H). We want our LLM agent to store a value for all eight parameters at each test. To monitor the test's success, we will compute the accuracy of our interface. For each test, we will check if the parameters stored by the LLM agent are exactly the expected ones as these parameters are going to be used by the home energy management system that is inflexible: for example "Oxford" is different from "OXFORD" if you want to then use this city name to look up weather forecast data. As described in section 4, we instantiate the LLM user with a prompt template of a given difficulty and personal information. For each test, we will store the user's personal information and the parameter the LLM agent stored by himself and we will count this test as a success if the parameters are exactly the ones retrieved by the agent.

We have defined a home energy management system that simulates the electric house consumption of a person (section 2). We can model different user behaviors with an LLM (section 4) and we can extract parameters with an LLM agent (section 3). For more detail on the parameter retrieval procedure, see section A.4 of the appendix.

5.1. Parameter retrieval effectiveness

To better test the reliability of the interface, we need to run more tests to fully evaluate the strengths and limits of our work. Tab. 2 shows each agent type's average accuracy per difficulty level. Overall, the V2-powered LLM agent achieved the best results (except in Easy mode). The accuracy of all LLMs is decreasing with rising difficulty, demonstrating the importance of user precision in extracting parameters. We observe that agent type Act is performing worse than $Act + example$ which performs slightly worse than $ReAct + example$ (except for Model V3 in Hard and Medium). Thus, it highlights the significance of examples in prompts and the added value of ReAct compared to Act-only.

With Fig. 6 in section A.4 of the appendix, we see the significance of ReAct and few-shot prompting methods in our context which defines a faster and more accurate agent. V2 with Act and V2 with $Act + examples$ are slow, and the proposed V2 with $React + example$ is faster and requires fewer questions. Even though agent type $Act + example$ has similar accuracy (see Tab. 2), this method requires more user questions than the $ReAct + example$ method in every difficulty level. As a reminder, an iteration is the process of asking the LLM agent to solve a task, i.e. to retrieve a parameter. Sometimes, the LLM agent fails to store a value, so it needs more than eight iterations to find all eight parameters. Act method is by far the longest method in terms of iteration and, therefore also in duration. We observe that $ReAct + example$ performs better than $Act + example$ in all three difficulty levels. Fig. 6 also shows that model V1 is far slower than models V3 and V2 and requires more iterations and questions.

LLM	Agent Type	Easy (E)	Medium (M)	Hard (H)
V1	Act	70.6	63.1	53.1
	Act+example	85.6	74.4	65.6
	ReAct+example	**97.5**	84.4	73.1
V2	Act	89.4	82.5	70.6
	Act+example	95	83.75	77.5
	ReAct+example	96.9	**87.5**	**78.8**
V3	Act	88.1	83.1	74.4
	Act+example	89.4	78.8	68.1
	ReAct+example	91.9	86.3	69.4

Table 2. Accuracy (%) for different agent types and difficulty levels.

6. Conclusion

With this study, we have moved toward the democratization of home energy management systems. We have developed a representative use case of a single home-owner interaction with an LLM agent that enables a person without technical knowledge to specify key parameters for home energy optimisation. The $ReAct + example$ LLM agent has shown remarkable adaptability by being able to retrieve parameters with different levels of user precision with higher retrieval accuracy and a smaller number of questions than with other methods.

Several leads can be exploited to correct the current limitations of our interface. Inferring more technical parameters for a more accurate HEMS like window-to-wall ratio or house orientation from a user non-technical description, user photos and house plan would make our approach more relevant as they can be easily obtained from a user and can provide much information the user may not be able to render to the LLM agent directly. Such tasks may require a multi-modal LLM. LLMs can provide extended user interactions. LLM-based user interactive systems can benefit from API integration like web searches or classical NLP approaches. We think that NLP algorithms and LLMs are

complementary and could be combined in future work in a faster and more efficient HEMS parameterization approach. User testing is essential to validate the usability and effectiveness of the LLM-based HEMS. The LLM-user introduced in this study models a wide variety of user behavior, pushing the LLM-agent to its limits and thus providing a solid testing approach. However, while our proposed LLM-based approach enables efficient testing, future efforts should incorporate pilot studies in diverse real households to gather feedback, evaluate performance, and refine the system for broader and more practical adoption. In future works, a more reliable LLM-based system should be tested with a large corpora of real independent users to fully evaluate the capabilities of the system. The test results will provide new objectives and highlight key points that will need to be addressed.

We might also want our LLM agent to take action based on its understanding of the user behavior, to be able to propose more adapted consumption strategies and explain them to the user. Being able to answer user questions, like "What should I do to diminish my electric bills ?" or basic ones on the algorithm documentation, would be a plus. One can imagine a multi-modal interface that could be combined with Retrieval-Augmented Generation to retrieve information from a range of data sources. In the case where a parameter has not been retrieved like house thermal characteristics, we can imagine an interface where the LLM agent could cross reference incomplete textual information from the user with other sources, such as online information about similar buildings or sensor readings.

Appendix A. Additional resources

A.1. HEMS

Here we present in detail the model behind the HEMS of the study with Eq. 1. The solar-generated electricity is directly consumed by user appliances, and if solar panels produce more energy than needed, the surplus is sold as highlighted in (Eq. 1d). Prices depend on the user's energy provider. The function the HEMS minimizes is defined in (Eq. 1a) and its variables are described in Tab. 3. We define $p_t(t)$ the total power the user is consuming at time t in (Eq. 1c). We did not consider V2G (the process of feeding the energy stored in an electric vehicle's battery back into the National Grid), which is why in (Eq. 1e) P_{EV} is always positive. (Eq. 1f) is the EV charging formula. (Eq. 1g) and (Eq. 1h) are respectively the start and end condition constraints. The temperature evolution formulas and parameters in (Eq. 1i) and (Eq. 1j) are taken from[21].

$$\min_{p_{heat}, p_{EV}} \quad \sum_{t=1}^{T} (p_t(t) - P_s(t))\pi(t) \tag{1a}$$

$$\text{s.t.} \quad \forall t = 1, \ldots, T \tag{1b}$$

$$p_t(t) = p_{heat}(t) + p_{EV}(t) + P_{other}(t) \tag{1c}$$

$$\pi(t) = \pi^e(t) 1_{p_t(t) \geq P_s(t)}(t) + \pi^s(t) 1_{p_t(t) < P_s(t)}(t) \tag{1d}$$

$$P_{EV\_max}(t) \geq p_{EV}(t) \geq P_{EV\_min}(t) \tag{1e}$$

$$E_{EV}(t + dt) = E_{EV}(t) + p_{EV}(t) dt \tag{1f}$$

$$E_{EV}(t_{arrival}) = E_{init} \tag{1g}$$

$$E_{EV}(t_{departure}) = E_{fully\_charged} \tag{1h}$$

$$\begin{aligned} T_{house}(t + dt) = & dt(\beta p_{heat}(t) + \alpha(T_{ext}(t) - T_{house}(t))) \\ & + T_{house}(t) \end{aligned} \tag{1i}$$

$$\alpha = \frac{1}{R_{th} C_{th}}, \quad \beta = \frac{\eta}{C_{th}}, \quad T_{max} \geq T_{house}(t) \geq T_{min} \tag{1j}$$

Variable	Description	Unit
p_t	total power consumption	kW
p_{heat}	heating power	kW
p_{EV}	EV(s) charging power	kW
P_{other}	power consumed by other appliances	kW
P_s	solar-generated power	kW
π^e	consumption electricity price	£/kWh
π^s	feed-in price of solar energy	£/kWh
E_{EV}	EV energy	kWh
T_{house}	house temperature	°C
T_{ext}	outdoor temperature	°C
C_{th}	thermal capacitance	kWh/°C
R_{th}	thermal resistance	°C/kW
η	coefficient of performance	1

Table 3. Variable description

Based on our HEMS in (Eq. 1), Tab. 4 describes all the parameters that are required to define the

HEMS's optimization problem and obtain the optimal consumption energy schedule. For example, with the user's location and the simulation period we can extract the outside temperature data and solar power generation[22][23]. There are many parameters of different types (dates, integers, floats, and strings). Each input needs to be perfectly formatted to be then used by the nonflexible HEMS in its optimization process. For a user without experience in HEMS or for the elderly, it is important to provide an easy-to-use interface while interacting with HEMS. An autofill interface could easily gather simple information but it can lack clarity for someone without experience. For example, a user may want to express his preferences in a detailed sentence that a single input cannot fully represent. Some parameters' names can also be confusing and too long parameter documentation can make the user puzzled. Ideally, a user should be able to precisely parametrize an HEMS with ease without being overwhelmed by the complexity of the required information. Parameters of Tab. 4 symbolize the information that can be expected from the user for the HEMS. Being able to easily retrieve them is key to addressing the complexity of HEMS parametrization.

A.2. ReAct

In this section, we detail the different components behind the ReAct agent of our HEMS parametrization system. The interface flowchart diagram shown in Fig. 2 describes with more detail a parameter retrieval of agent type $React + example$ which is symbolized by steps B1 and B2 in Fig. 1. The LLM agent has to find each one of the user parameters thanks to the user's answers to the LLM agent's questions.

As shown in Algorithm 1, the React algorithm of our interface starts by initiating the LLM agent with a prompt template and a task (line 2). The algorithm stops when the maximum number of iterations is reached or when the LLM agent has found and stored the desired parameter that can be checked with the is_done function (line 3). The LLM agent will generate texts containing the Thought/Action/Observation format (line 5). The generated response is parsed. If the format is respected, the parser will output the result of the function called in the LLM-generated JSON (line 6); otherwise, the parser will output an error message. In each case, we concatenate the newly generated text and the parser's output into the agent prompt (line 7). Upon completion, the algorithm returns the stored value of the LLM in case of success.

Algorithm 1 React LLM agent parameter retrieval algorithm

```
iter ← 0
agent_prompt ← init_prompt(task)
while not is_done() and n_iter ≥ iter do
    iter ← iter + 1
    response ← generate(agent, agent_prompt)
    output ← parser(response)
    agent_prompt ← agent_prompt + response + output
end while
return get_result(agent_prompt)
```

Algorithm 2 describes the interface in detail. A toolkit object containing the communication tools with the user and the HEMS is instantiated (line 2). It creates its tools' description that we will give to the LLM agent. The algorithm stops when the LLM agent has found and stored every desired parameter which can be checked with the $is\_complete$ function (line 3). At each step, the LLM agent is instantiated with a prompt template and a toolkit (line 5). Then, we ask the LLM to solve a task (line 6). After, the algorithm updates the task list (line 8). Finally, the HEMS function is called with the retrieved parameters. The HEMS function solves (Eq. 1) to obtain the user's optimal energy consumption schedule and setpoints for the controllable devices.

Algorithm 2 Parameters retrieval algorithm

```
1:  stored_parameters ← None
2:  toolkit ← init_tools(user, stored_parameters)
3:  while not is_done() do
4:      current_task ← task_list[0]
5:      agent ← React(LLM, toolkit)
6:      output ← agent(current_task, n_iter)
7:      if params_stored(output) then
8:          task_list.pop(0)
9:          stored_parameters.append(output)
10:     end if
11: end while
12: return HEMS(stored_parameters)
```

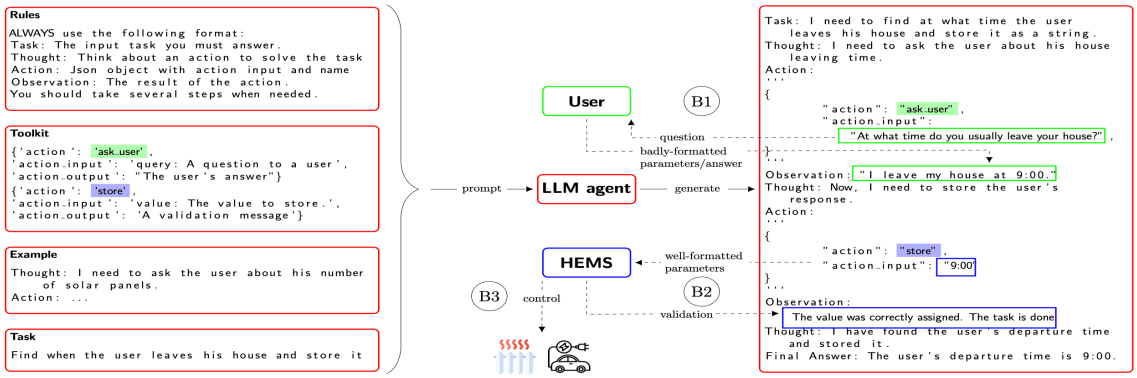


Figure 2. Interface flow diagram of React retrieval. The **LLM agent** first receives a prompt that is divided into four parts: **Rules** specifying the format it must follow, a **Toolkit** containing the communication tools the LLM agent can use when acting, an **Example** providing a hint about the expected work, and the actual **Task** it must solve. Step B1 is performed by calling the function `"ask_user"` with the user question as input, which outputs the user's answer. In Step B2, the **LLM agent** sends a formatted parameter to the **HEMS** by calling the function `"store"`. This function returns a validation message when the parameter has the correct format.When all parameters are found, the **HEMS** can control the user's appliances in Step B3.

A.3. User

We used the embedding model all-MiniLM-L6-v2, a powerful derivative of the MiniLM developed by Microsoft Research[24] to compute embedded the different textual data (user's answers and the perfect answers). We used Mistral AI instruct model V2 to generate the answers of the LLM user[20].

All LLM starts by embedding a sentence into a vector before generating the following words of the prompt. In this vector space, two sentences with similar meaning will have a similar vector. To quantify user precision, we want to compute a vector similarity metric between a perfect answer that is concise and respects the wanted format and our user answer. We computed the cosine similarity metric to quantify user answer precision in different difficulty modes[25]. For example, to the question "How many electric vehicles do you own ?", we define the perfect answer as "I own 2.". For some parameters such as the number of electric vehicles or EVs (see Tab. 4), a very concise answer might not fully represent the context of the sentence (electric vehicles in the example and not solar panels). Thus, we need to add the context of the answer -which is the question here- to obtain the full meaning of the user's answer. Therefore, to compute user precision, we compute the cosine similarity

doi.org/10.32388/EVPWM7

between the embedding of both the question and the user's answer in all three difficulty modes compared to the embedding of the same question and the perfect answer.

The similarity equation between LLM-user and defined perfect answers is presented in Eq. 2. $A(Q)$ is the answer of the user to question Q, $A^*(Q)$ is the best answer to question Q and f is the embedding function.

$$cosine(Q) = \frac{f(Q + A(Q)) \cdot f(Q + A^*(Q))}{\|f(Q + A(Q))\| \|f(Q + A^*(Q)\|} \tag{2}$$

We ask the LLM 20 times eight simple questions -that refer to each one of the eight parameters described in table 4- in each difficulty level, to get a sample of answers from which we will evaluate the average user precision. For example, to evaluate user precision for parameter EV in difficulty mode "easy E", we give the LLM Fig. 12 (see section B.2 of the appendix) and we ask the LLM this question "How many electric vehicles do you own ?" several times. Then we compute the cosine similarity between the embedding of the question and the generated answers and the embedding of the question and the perfect answer ("I own 2 electric vehicles." in the example). Tab. 1 shows that the hierarchy between the three difficulty modes is respected. Indeed the answers generated by the LLM user in easy mode are more similar to the optimal answer than those in medium mode.

Fig. 3 highlights that for all parameters the cosine similarity score of the LLM user difficulty mode Easy (E) is greater than the score in mode Medium (M) except for date parameters. Overall, there is a greater similarity between the user's answers in easy mode and the defined best answers rather than with mode medium and hard. The hierarchy is respected. In the user prompts for easy mode, we ask the LLM user to use the YYYY/MM/DD date format, and in medium mode, we ask the LLM user to use the DD-MM-YYYY format. The date tasks tell the LLM agent to store the date in YYYY/MM/DD format. So the answers in easy mode will not require conversion. Thus, the medium mode requires more work from the LLM agent. However, the embedding model does not take into account those different date formats in its final answer embedding. That is why for date parameters, mode easy and medium scores are close.

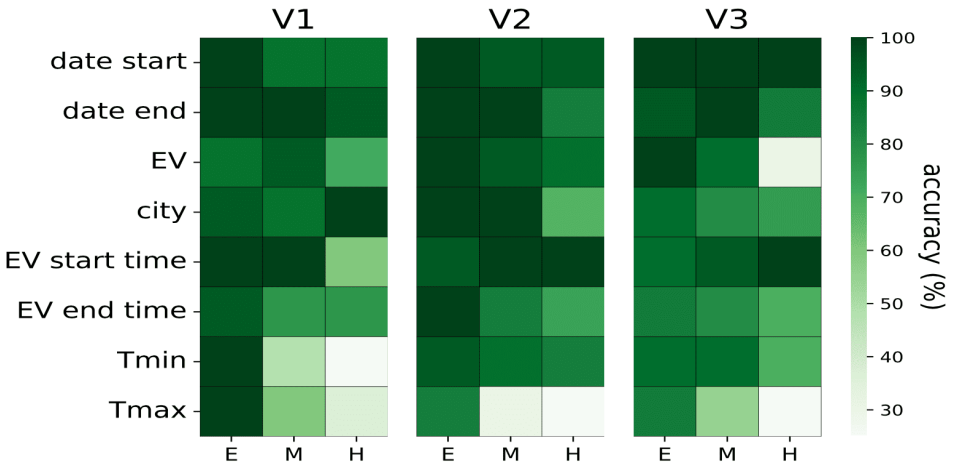


Figure 3. Average cosine similarity score of user precision. This figure shows the cosine similarity score between a typical user answer and a perfect precise answer for each difficulty mode.

A.4. Results

To model the electricity consumption of several appliances of the HEMS, we used the dataset provided in[26]. We use the parameter settings in Tab. 5[21] for the temperature evolution formula (Eq. 1i) with $\eta = 1$ as we use electric heating and not a heat pump and we increased the thermal resistance to model a well-insulated house.

Parameter	Description	Value	Unit
C_{th}	thermal capacitance	2	kWh/°C
R_{th}	thermal resistance	10	°C/kW
η	coefficient of performance	1	1

Table 5. Typical parameter values for residential heating

As described in section 3 with Fig. 1, the interface starts with step `B1`: the user–LLM agent interaction.

doi.org/10.32388/EVPWM7

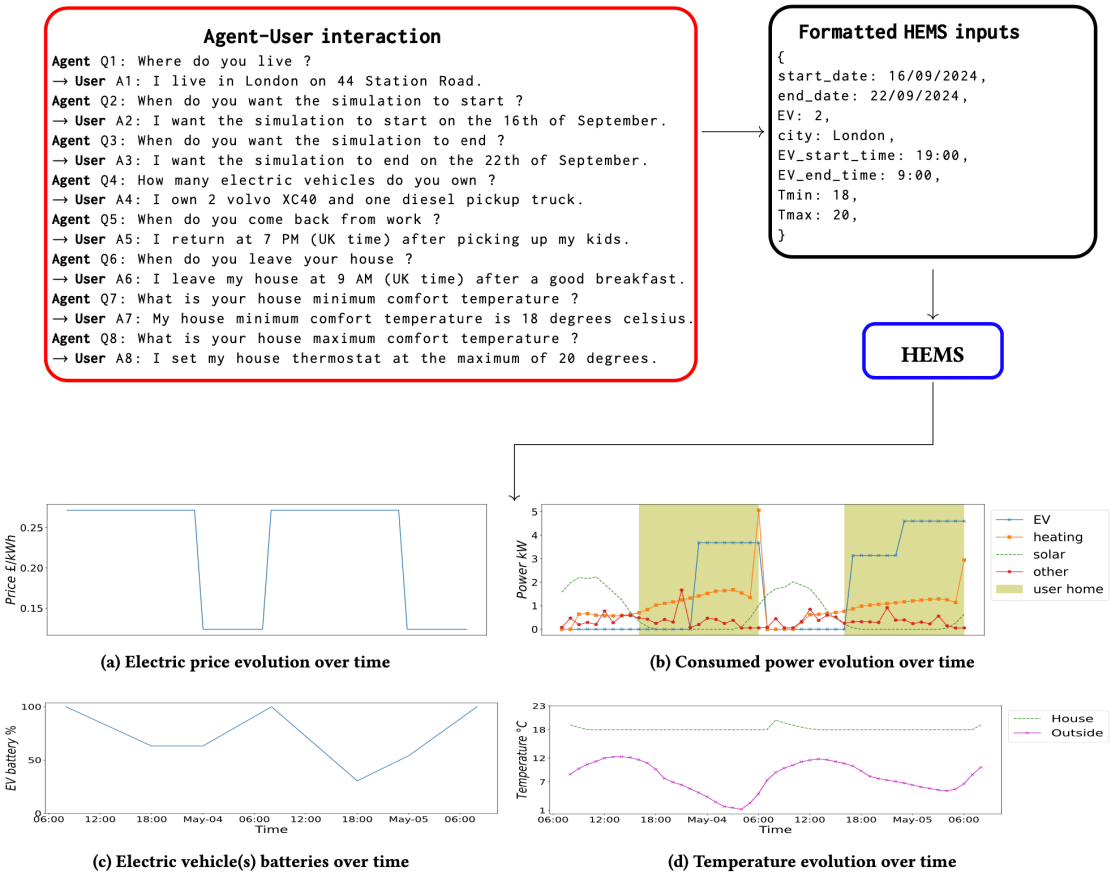
To perform this stage, we need to instantiate the LLM agent with its prompt (see appendix Fig. 10 in section B.1 and section 3) and a task. The LLM agent receives the task "Find the user's city in the United Kingdom and store it (city must be a string)." associated parameter $city$ (see Tab. 4). The LLM agent will generate Fig. 4. In line 1, the agent rephrases the main task. In line 2, the agent first decides to fulfill its first goal, i.e. to ask a question to the user in lines 3 to 9. The user answers with line 10. Here the first step B1 of Fig. 1 is completed, the agent has received badly formatted information from the user (instead of just answering "Oxford", the user decided to give extra noisy information). Then, the LLM agent decides to store this information in line 12 and use the tool "store" to do so (lines 13 to 19). It receives a validation message in line 20, completing step B2 of Fig. 1. Finally, the LLM agent understands that its main task is fulfilled and ends its text generation.

````
1  Task: Find the user's city within the United Kingdom and store
   it as a string.
2  Thought: I need to ask the user for his city.
3  Action:
4  ```
5  {
6  "action": "ask_user",
7  "action_input": "Which city in the United Kingdom do you live
   in?"
8  }
9  ```
10 Observation: I live in Oxford on Banbury Rd.
11 Thought: I need to store this parameter.
12 Action:
13 ```
14 {
15 "action": "store",
16  "action_input": "Oxford"
17 }
18 ```
19 Observation: The value was correctly assigned. The task is done
20 Thought: I have found the user's city and stored it.
21 Final Answer: The user lives in Oxford.
````

Figure 4. LLM agent retrieval example

After having successfully retrieved all user information and formatted it in a HEMS-compatible format (see right part of Fig. 2), the HEMS will compute the optimal consumption energy schedule. Fig. 5 shows the whole user-LLM interaction process from the user's perspective and the HEMS output: the most cost-efficient control of electric appliances while fulfilling the user's constraints. Fig. 5c shows that when the user leaves his house the car(s) battery is full. The car is charged when the

user is back home. The HEMS charges the car only when the electricity price is low allowing the user to use his car as he wants while diminishing his bills. With Fig. 5d, we can check that the temperature constraints are also validated as T_{house} is between T_{min} and T_{max} values. Also, in the low price period, the HEMS heats the house to the T_{max} limit, to heat less when electricity is more expensive.



(a) Electric price evolution over time

(b) Consumed power evolution over time

(c) Electric vehicle(s) batteries over time

(d) Temperature evolution over time

Figure 5. The interaction process of the LLM-integrated HEMS from a user's perspective. The **Agent-User interaction** text box describes a typical usage of the interface. From the user's answers, the LLM agent stores the formatted parameters in text box **Formatted HEMS inputs**. The **HEMS** computes the optimal energy consumption schedule in Fig. 5. Fig. 5a depicts the electric price evolution over time which is an Economy 7 setup. Fig. 5b describes the consumed power evolution over time for heating (square marker orange), EV charging (x marker blue), other appliances (o marker red), and solar-generated power (dashed green). The yellow shaded areas represent when the user is home or not. Fig. 5c represents electric vehicle(s) batteries over time. Fig. 5d describes the outside temperature (x marker purple) and house temperature (dashed green) evolution over time.

doi.org/10.32388/EVPWM7

Fig. 7 show the average accuracy of the $ReAct + example$ agent for each parameter. It can be seen that the LLM agent struggled the most with $Tmin$ and $Tmax$ parameters. The models had trouble handling Celsius degrees and hallucinated by doing conversions even though it was not required in the given tasks. Temperature parameters are described to the LLM user in this way:

- easy: minimum house comfort temperature: $TMIN °C maximum house comfort temperature: $TMAX °C
- medium: Your house comfort temperature is between $TMIN °C and $TMAX °C.
- hard: You like to set your house thermostat to be between $TMIN and $TMAX degrees Celsius.

By mixing the temperature information in one sentence for difficulty mode medium and hard, the LLM user tends to add those two pieces of information or mix them in its answers to the LLM agent questions. That is where the LLM agent fails to extract the precise value. For example to the LLM agent question: "What is your preferred maximum house temperature?", the LLM user in medium or hard mode can answer "My preferred maximum house temperature is between 18 and 19 degrees Celsius". In this particular case, the LLM agent may get confused and store 18.5 instead of 19. Here, the user's answer is imprecise, so the LLM agent interprets that any values between 18 and 19 are correct. However, the LLM agent should understand that it needs to ask a second question to clarify the user's preference. From the different tests we have run, we observed that the LLM user tends to be more concise when asked about $Tmin$ which greatly helps the LLM agent. We conjecture that the LLM user is more precise with $Tmin$, because the minimum temperature is the main concern when heating your house, which may be more expressed in the datasets used by LLM developers to train LLMs.

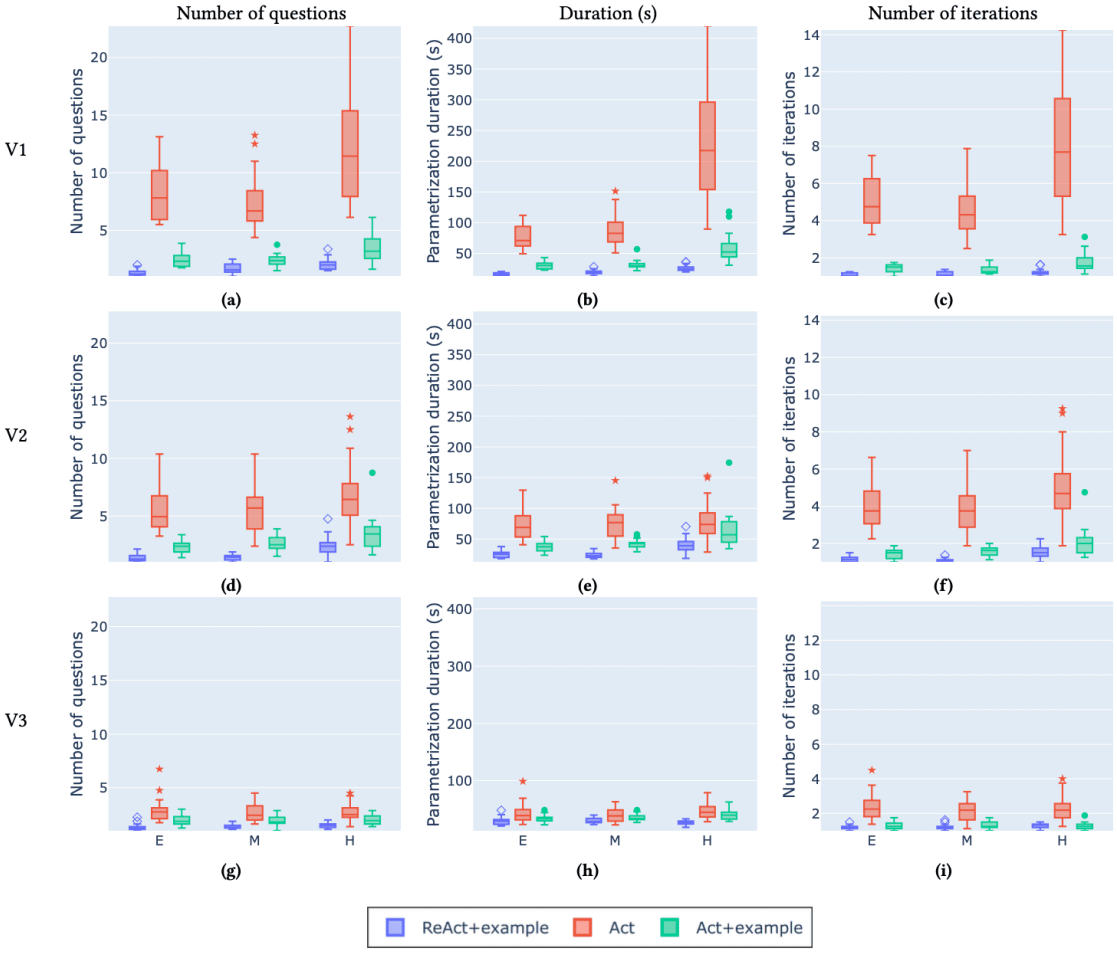


Figure 6. Boxplot distribution of test results for model V2. Subgraphs of column 1 (Fig. 6a, 6d, 6g) represent the boxplot distribution of the number of questions asked by the model to retrieve a **single** parameter for each difficulty level with three different agent types. Subgraphs of column 2 (Fig. 6b, 6e, 6h) depict the boxplot distribution of a **single** parameter retrieval duration in seconds per difficulty level. Subgraphs of column 3 (Fig. 6c, 6f, 6i) represent the boxplot distribution of the number of algorithms iterations required by the agent to retrieve a **single** parameter for each difficulty level. Subgraphs of row k (1,2 or 3) refers to the results obtained with LLM version Vk. Agent types $ReAct + example$, Act, and $Act + example$ are symbolized by blue diamonds, red stars, and green circles.

doi.org/10.32388/EVPWM7

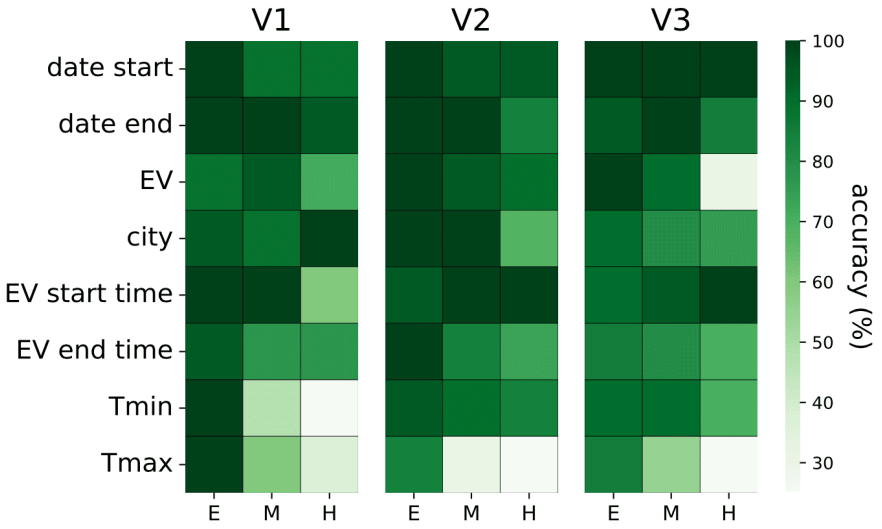


Figure 7. Accuracy for each parameter with $ReAct + example$ This figure represents the rate of retrieval per parameter per difficulty level for each LLM. E, M, and H respectively stand for Easy, Medium, and Hard difficulty levels. The models tend to underperform with temperature parameters ($Tmin$, $Tmax$) and with dates ($date\_end$, $date\_start$).

Appendix B. Prompts

B.1. LLL-agent prompts

In this section, we describe the agent prompt template of Fig. 10, used for the $ReAct + example$ agent type. As mentioned in section 3.2, we first describe the main goal of the LLM agent (line 1). We chose to keep this first line simple for a shorter prompt. Then, the prompt continues with the tools list and their description (lines 1 and 5). Here the value $TOOL_DESCRIPTION is replaced by Fig. 8. The format rules it must follow for function calling are described on lines 3 to 13. We added two redundant messages on the type correctness of the "action_input" key after having experienced several failures due to this error. The LLM had trouble handling double quotes when working with strings. Line 5 was added to make the LLM remember the importance of using correct action names in the JSON blob. The ReAct rules on lines 15 to 27, follow a classic ReAct implementation. We used the keywords "Observation:", and "Final Answer:" as generation-stopping criteria, respectively to parse the LLM-generated text and then call the appropriate function, and to check if the task is successful.

```
{'action': 'ask_user', 'action_description': "Return the user's answer to a query", 'action_input': 'query: A question to a user', 'action_output': "The user's answer"}
{'action': 'store', 'action_description': 'Store a value by assigning it to an argument', 'action_input': 'value: The value to store.', 'action_output': 'A validation message'}
```

Figure 8. Tool description

Fig. 10 is then formatted into a chat template segmenting the agent prompt template, the example, and the current task. Such a format allows the LLM to understand its goals, examples, and current task fully. Fig. 9 describes the used chat template. Each part starts with the keyword "<|im_start|>" and ends with the keyword "<|im_end|>" to make the LLM understand the chat template segmentation. Each part is then composed of a role (system, agent, or assistant) defining the purpose of the part. "System" refers to LLM agent instructions. "Agent" is the LLM agent, so each text generated by the LLM agent will have the role "agent". "Assistant" is the role giving tasks. It is not an LLM but a part of the initialized prompt. For example, for each HEMS parameter retrieval, the "Assistant" could be: "Now you need to retrieve the minimum temperature". After the role comes the content: $PROMPT_TEMPLATE, $TASK_EXAMPLE, $EXAMPLE, $TASK, respectively replaced by the actual initialized agent prompt template of Fig. 10, a task example like "Find the user's number of solar panels and store it (number must be an integer).", how should the LLM solve this task, the current task the LLM should solve. For agent types Act and $Act + example$ the agent prompt template is shorter (no need to describe the thinking process at each step) and the chat template is without example for agent type Act.

```
<|im_start|>system
$PROMPT_TEMPLATE<|im_end|>
<|im_start|>agent
$TASK_EXAMPLE<|im_end|>
<|im_start|>assistant
$EXAMPLE<|im_end|>
<|im_start|>agent
$TASK<|im_end|>
<|im_start|>assistant
```

Figure 9. Agent chat template

```
You will have to solve tasks as best you can. To do so, you have access to the following tools: $TOOLS_DESCRIPTION

The way you use the tools is by specifying a json blob. Specifically, this json should have a `action` key (with the name of the tool to use) and a `action_input` key (with the input to the tool going here). The `action_input` key must be a valid python object (string, integer, list, float). If your `action_input` is a string you must use "" instead of ''.

The only values $TOOL_NAME that should be in the "action" field are: $TOOLS_LIST

The $JSON_BLOB should only contain a SINGLE action and MUST be formatted as markdown, do NOT return a list of multiple actions. Here is an example of a valid $JSON_BLOB:
{
  "action": $TOOL_NAME,
  "action_input": $INPUT
}

Make sure to have the $INPUT in the right format for the tool you are using, and is a correct JSON input. The $JSON_BLOB must be formatted as markdown and only use a SINGLE action at a time.

ALWAYS use the following format:
Task: The input task you must answer.
Thought: You should always think about one action to solve the task.
Action: $JSON_BLOB
Observation: The result of the action.

This Thought/Action/Observation can repeat N times, you should take several steps when needed. You should keep repeating the above format until you have enough information to answer the question without using any more tools.

You must always end your output with the following format:
Thought: I now know the final answer.
Final Answer: The final answer to the original input question.

Now begin! Reminder to ALWAYS use the exact characters `Final Answer: $ANSWER` when you provide a definitive answer.
```

Figure 10. Agent prompt template

In case the JSON blob parsed from the text generated by the LLM agent is not correct, the parser will return an error message (see Fig. 11). This message sums up the LLM agent's mistake, replacing keyword $ERROR with the actual Python error message (line 2). Then two recalls are given on type and function name correctness.

```
1 You made a mistake in your JSON blob.
2 Here is the Python error message: $ERROR
3
4 Remember, `action_input` must be a valid Python object
  (str, int, list, float).
5 Try again with the right format and the right keys.
```

Figure 11. Error message

B.2. LLL-user prompts

In Fig. 12, 13 and 14, $CITY, $EV, $TMIN, $TMAX, $ARRIVAL_TIME, $LEAVING_TIME, $DATE1 and $DATE2 are replaced with the random values for each test in a format that depends on difficulty level (for example, with date values, "2024/10/18" in easy mode and "October, 18th, 2024" in hard mode). All three prompts start with the main task of the LLM user (line 1). Then, we describe the user personal information we want our LLM user to answer with (Fig. 12 lines 4 to 11, Fig. 13 lines 4 to 10, and Fig. 12 lines 4 to 9). Along those lines, we give the same information to each LLM user in all three difficulty modes, but we change the format of this information and we have added noisy information in harder modes so that the user answers will contain too much information for the agent and its parameter retrieval will be harder. Finally, we add each prompt with rules to follow when answering (Fig. 12 lines 13 to 14, Fig. 13 lines 12 to 13, and Fig. 12 lines 11 to 12). Notice how the rule precision diminishes with the rising difficulty. We even force the LLM user to generate a more complex answer in Fig. 13 and 14.

```
You will have to answer questions about your personal information.
Here is your personal information:

city: $CITY
number of electric vehicles: $EV
minimum house comfort temperature: $TMIN C
maximum house comfort temperature: $TMAX C
time when you come back home: $ARRIVAL_TIME
time when you leave your house: $LEAVING_TIME
simulation start date: $DATE1
simulation end date: $DATE2

Answer with a concise first-person sentence of less than 25 tokens.
For date questions, use this format YYYY/MM/DD.
```

Figure 12. User prompt template easy mode

```
You will have to answer questions about your personal information.
Here is your personal information:

You live in $CITY  in England, in a house.
You own $EV volvo XC40.
Your house comfort temperature is between $TMIN °C and $TMAX °C.
You come back from work at $ARRIVAL_TIME PM (UK time) after picking up your kids.
You leave your house at $LEAVING_TIME AM (UK time) after a good breakfast.
You want the simulation to start on the $DATE1.
You want the simulation to end on the $DATE2.

Answer with a concise first-person sentence.
For date questions, use this format dd-mm-yyyy.
```

Figure 13. User prompt template medium mode

```
You will have to answer questions about your personal information.
Here is your personal information:

You live in $CITY in England on Banbury Road, in a house with your family.
You own $EV volvo XC40, one diesel pickup truck and a gas-powered  motorcycle.
You like to set your house thermostat to be between $TMIN and $TMAX degrees Celsius.
You are back from work at $ARRIVAL_TIME PM due to traffic.
You go to work at $LEAVING_TIME AM to escape traffic.
You want to simulate your electric consumption between $DATE1 and $DATE2.

You MUST Answer with two first-person sentences for each question.
Add an explanation to your answer.
```

Figure 14. User prompt template hard mode

Like the LLM agent, we use a chat template for the LLM user (see Fig. 15). Here, such a format allows the LLM to fully understand its goals, and the question it must answer. We follow the same format: start keyword, role (system, agent, and user), content, and end keyword. $USER_PROMPT, and $QUERY are respectively changed by one of the user prompts (Fig. 12, 13, and 14).

```
<|im_start|>system
$USER_PROMPT<|im_end|>
<|im_start|>agent
$QUERY<|im_end|>
<|im_start|>user
```

Figure 15. User chat template

Appendix C. Online Resources

The study's codes are available on a GitHub repository[27]. The README.md file of this repository describes the project's architecture, installation, and usage.

Acknowledgments

This work was supported by EDF Energy and the Maison Française d'Oxford. I want to express my deepest appreciation to everyone involved in the Department of Engineering Science of Oxford - EDF Energy scheme that proposed the research project I have worked on and have presented in this article. EDF Energy's generous contributions have made it possible to conduct this study and have significantly contributed to its successful completion.

References

1. ^*European Environment Agency. Eea.europa.eu. https://www.eea.europa.eu/publications/flexibility-solutions-to-support/at_download/file 2023.*
2. ^*Vassilis Stavrakas, Alexandros Flamos. (2020). A modular high-resolution demand-side management model to quantify benefits of demand-flexibility in the residential sector. Energy Conversion and Management. 205:112339. doi:https://doi.org/10.1016/j.enconman.2019.112339*
3. ^*Jinsoo Han, Chang-Sic Choi, Wan-Ki Park, Ilwoo Lee. (2011). Green home energy management system through comparison of energy usage between the same kinds of home appliances. In: 2011 IEEE 15th international symposium on consumer electronics (ISCE).: Singapore: IEEE pp. 1–4. doi:10.1109/ISCE.2011.5973168*
4. ^*Bin Zhou, Wentao Li, Ka Wing Chan, Yijia Cao, Yonghong Kuang, et al. (2016). Smart home energy management systems: Concept, configurations, and scheduling strategies. Renewable and Sustainable Energy Reviews. 61:30–40. doi:https://doi.org/10.1016/j.rser.2016.03.047*
5. ^*Thomas Morstyn, Katherine A. Collett, Avinash Vijay, Matthew Deakin, Scot Wheeler, et al. (2020). OPEN: An open-source platform for developing smart local energy system applications. Applied Energy. 275:115397. doi:https://doi.org/10.1016/j.apenergy.2020.115397*
6. ^*Rodrigo Verschae, Takekazu Kato, Takashi Matsuyama. (2016). Energy management in prosumer communities: A coordinated approach. Energies. 9(7). doi:10.3390/en9070562*

7. ^Takashi Matsuyama. I-energy: Smart demand-side energy management. In: Smart grid applications and developments.: London: Springer London 2014. pp. 141–163. doi:10.1007/978-1-4471-6281-0_8. I SBN 978-1-4471-6281-0
8. ^Dasheng Lee, Chin-Chi Cheng. (2016). Energy savings by energy management systems: A review. Rene wable and Sustainable Energy Reviews. 56:760–777. doi:https://doi.org/10.1016/j.rser.2015.11.067
9. ^Chien-fei Chen, Xiaojing Xu, Jacqueline Adams, James Brannon, Fangxing Li, et al. (2020). When east meets west: Understanding residents' home energy management system adoption intention and willing ness to pay in japan and the united states. Energy Research Social Science. 69:101616. doi:https://doi.or g/10.1016/j.erss.2020.101616
10. ^Mohamed Zeid, Subir Majumder, Hasan Ibrahim, Prasad Enjeti, Le Xie, et al. Predicting DC-link capac itor current ripple in AC-DC rectifier circuits using fine-tuned large language models. 2024. Available fr om: https://arxiv.org/abs/2407.01724
11. ^Subir Majumder, Lin Dong, Fatemeh Doudi, Yuting Cai, Chao Tian, et al. (2024). Exploring the capabili ties and limitations of large language models in the electric energy sector. Joule. 8(6):1544–1549. doi:ht tps://doi.org/10.1016/j.joule.2024.05.009
12. a, bChenghao Huang, Siyang Li, Ruohong Liu, Hao Wang, Yize Chen. Large foundation models for powe r systems. 2023. Available from: https://arxiv.org/abs/2312.07044
13. ^Rodrigo S. Bonadia, Fernanda C. L. Trindade, Walmir Freitas, Bala Venkatesh. (2023). On the potential of ChatGPT to generate distribution systems for load flow studies using OpenDSS. IEEE Transactions on Power Systems. 38(6):5965–5968. doi:10.1109/TPWRS.2023.3315543
14. a, bMengshuo Jia, Zeyu Cui, Gabriela Hug. Enabling large language models to perform power system si mulations with previously unseen tools: A case of daline. 2024. Available from: https://arxiv.org/abs/24 06.17215
15. a, bMathyas Giudici, Luca Padalino, Giovanni Paolino, Ilaria Paratici, Alexandru Ionut Pascu, et al. (202 4). Designing home automation routines using an LLM-based chatbot. Designs. 8(3). doi:10.3390/desig ns8030043
16. a, bDmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, et al. (2024). AIoT smart home via autonomous LLM agents. IEEE Internet of Things Journal. :1–1. doi:10.1109/JIOT.2024.347190 4
17. ^Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, et al. (2023). Toolform er: Language models can teach themselves to use tools. ArXiv. abs/2302.04761. Available from: https://a

pi.semanticscholar.org/CorpusID:256697342

18. ^*Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. Language models a re few-shot learners. 2020. Available from: https://arxiv.org/abs/2005.14165*
19. ^*Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, et al. ReAct: Synergizing reasoning and acti ng in language models. 2023. Available from: https://arxiv.org/abs/2210.03629*
20. a, b*Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, et al. (2020). Tran sformers: State-of-the-art natural language processing. In: Proceedings of the 2020 conference on emp irical methods in natural language processing: System demonstrations.: Online: Association for Comput ational Linguistics pp. 38–45. Available from: https://www.aclweb.org/anthology/2020.emnlp-demos. 6*
21. a, b*He Hao, Borhan M. Sanandaji, Kameshwar Poolla, Tyrone L. Vincent. (2015). Aggregate flexibility of thermostatically controlled loads. IEEE Transactions on Power Systems. 30(1):189–198. doi:10.1109/TP WRS.2014.2328865*
22. ^*Stefan Pfenninger, Iain Staffell. (2016). Long-term patterns of european PV output using 30 years of v alidated hourly reanalysis and satellite data. Energy. 114:1251–1265. doi:https://doi.org/10.1016/j.energ y.2016.08.060*
23. ^*Iain Staffell, Stefan Pfenninger. (2016). Using bias-corrected reanalysis to simulate current and future wind power output. Energy. 114:1224–1239. doi:https://doi.org/10.1016/j.energy.2016.08.068*
24. ^*Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, et al. MiniLM: Deep self-attention distillati on for task-agnostic compression of pre-trained transformers. 2020. Available from: https://arxiv.org/a bs/2002.10957*
25. ^*Martina Toshevska, Frosina Stojanovska, Jovan Kalajdjieski. (2020). Comparative analysis of word em beddings for capturing word similarities. In: 6th international conference on natural language processi ng (NATP 2020).: Copenhagen, Denmark: Aircc Publishing Corporation. (NATP 2020). doi:10.5121/csit.2 020.100402*
26. ^*CLNR. Homepage - Customer-Led Network Revolution — networkrevolution.co.uk. http://www.netw orkrevolution.co.uk/ 2024.*
27. ^*EsaLaboratory. LLM. 2024. Available from: https://github.com/EsaLaboratory/llm*

Declarations

Funding: This work was supported by EDF Energy and the Maison Franaise d'Oxford.

Potential competing interests: No potential competing interests to declare.