Leveraging Large Language Models for Enhanced Personalised User Experience in Smart Homes

Jordan Rey-Jouanchicot*†‡, André Bottaro*, Eric Campo‡, Jean-Léon Bouraoui*,
Nadine Vigouroux†, Frédéric Vella†
*Orange Innovation, Blagnac, 31700, France
†IRIT, University of Toulouse, CNRS, Toulouse INP, UT3, UT2J, Toulouse, France
‡LAAS-CNRS, University of Toulouse, CNRS, UT2J, Toulouse, France
e-mail: { jordan.reyjouanchicot@orange.com | andre.bottaro@orange.com | eric.campo@laas.fr | | jeanleon.bouraoui@orange.com | nadine.vigouroux@irit.fr | frederic.vella@irit.fr}

***Abstract*—Smart home automation systems aim to improve the comfort and convenience of users in their living environment. However, adapting automation to user needs remains a challenge. Indeed, many systems rely on hand-crafted routines. This paper presents an original smart home architecture leveraging Large Language Models (LLMs) and user preferences to push the boundaries of personalisation and intuitiveness in the home environment. This article explores a human-centred approach that uses the general knowledge provided by LLMs to learn and facilitate interactions with the environment. The advantages of the proposed model are demonstrated on a set of scenarios, as well as a comparative analysis with various LLM implementations. Some metrics are assessed to determine the system's ability to maintain comfort, safety, and user preferences. The paper details the approach to real-world implementation and evaluation. The proposed approach shows up to 52.3% increase in average grade, and with an average processing time reduced by 35.6% on Starling 7B Alpha LLM. In addition, performance is 26.4% better than the results of the larger models without preferences, with almost 20 times faster processing time.**

Keywords-Artificial Intelligence; Decision-making; Adaptivity; Smart Home Automation System; Modelisation

I. Introduction

Networks of devices are deployed to assist human beings in their daily activities, using notions of context and knowledge to decide on the best actions to take [1][2]. Indeed, many houses are covered by wireless and wired networks and equipped with electronic devices allowing the occupants to control their environment for comfort, entertainment, security, energy management, and elderly care.

However, Smart Home Automation Systems are still missing the aim of autonomously taking the best action in every situation. Aligning automation routines to meet every need for every home configuration, every set of devices, available or not, functional or not, remains a challenge. In fact, most systems today are configured for simple routines that occur frequently. For instance, setting the home for wake-up or departure time by playing music, ringing a bell, acting on lights, shutters, heating, ventilation, and air conditioning. More precise needs in less frequent situations are not covered. Furthermore, if some devices are missing from a routine, no decision is made to use alternative devices. While artificial intelligence can play a role in learning about situations and the associated actions taken by users [3], it still requires time to understand users' habits, and is never able to cover the wide range of situations encountered at home.

One of the main technical challenges of ubiquitous computing is the ability to set up a system knowing the wide variety of users' potential needs, and how it can adapt to the wide range of functions of existing devices, with devices and locations whose configuration can be very different. As their execution environment is particularly dynamic, applications need to be aware of their context and act appropriately.

Large language models [4] have the potential to give this general knowledge at once to applications, such as smart home. In essence, these models have acquired a vast amount of knowledge. They are trained on a diverse range of textual sources, enabling them to cover a variety of topics, facts, and concepts, here the expected actions of home devices to meet user needs in a wide range of situations.

Retrieval-Augmented Generation (RAG) [5] is a technique that improves the accuracy of LLMs (Large Language Models), by giving them access to more targeted and precise information. This is achieved by using a retrieval component that searches a specific database and introduces it into the model, along with users' query. To do this, an embedding model is needed to represent the sentences in vector form, so that we can find the results closest to the query.

This paper proposes a software architecture integrating the general LLM knowledge available today into a smart home automation system. The LLM is placed at the centre of the home's decision-making system and participates in the reaction to every event to deduce the next best actions. This paper investigates the inclusion of preferences with a LLM for smart home automation, The contribution also includes a user-centred representation of smart home states and actions in natural language. Finally, experiments are carried out using several LLMs with different prompting styles for decision-making in the smart home.

Section II presents related work in the literature. The proposed software architecture is detailed in Section III, in particular the integration of LLMs for decision-making combined with preferences. Afterwards, Section IV describes a benchmark dedicated to LLMs and prompting styles with a

set of home scenarios. The results are described in Section V and discussed in Section VI. Finally, Section VII is devoted to conclusion and future work.

II. Related work

This topic is recent and few papers have been published on LLMs for smart homes. The following papers are selected for their approaches using the knowledge of user preferences on decision-making in smart home automation systems. This section cannot be exhaustive in this larger domain.

Oliveira et al. [6] propose a multi-agent environment with a Belief-Desire-Intention cognitive model, to support adaptivity and preferences transparently in a smart home environment. Any possible interaction could be modeled in such a way as to allow alternative proposals, but this requires considerable work in semantic representation to be fully complete.

Another paper takes advantage of general knowledge information to improve system adaptivity to preferences, [7] proposes an approach using three Knowledge-Based systems, one with general knowledge, one with skill knowledge, and one with contextual information, such as device location, and generating rule models for a middleware platform based on all this information. This approach requires really strict semantic modelling to handle most scenarios, and cannot take advantage of some information that is implicitly given in context.

Shuvo [8] proposes an Actor-to-Critic (A2C)-based algorithm adapted to decision-making in smart homes for energy consumption. In this work, at each step, an A2C algorithm is applied to each device to select the best action, using as inputs the activity and price of electricity at that time. A key element is that the set of actions for each appliance depends on the category associated with the appliance, adding initial knowledge to the model to ensure action based on the importance of the appliance. This system requires learning, and any change in device availability leading to different optimal decisions will require many steps before adapting. In addition, the larger the context, the more difficult it will be to converge for each scenario to learn the best-suited device state. Zhang et al. [9] also propose a system for energy consumption in smart homes, it combines a set of control rules and reinforcement learning to reduce the adaptation time.

Peng et al. [10] describe an approach for decision-making in home automation using deep reinforcement learning. It showed the ability to learn when to turn on a light but with a really limited context supported, so the application is limited to learning when to turn on a light with schedules of 15 minutes, which is quite a large period for this application.

The first approach to smart home automation with LLM [11][12] uses a JSON (JavaScript Object Notation) [13] data representation from a smart home middleware platform and experiments with an LLM to select an action based on a user request. This approach is a first step towards the use of the general knowledge provided by these models. However, it does not support user preferences, and the idea is to select actions based on an initial user request, rather than proactively. These works have proposed ways of managing decision-making, but are limited by the contextual data supported. Works using symbolic AI methods can show great adaptability, but at the cost of extensive semantic modelling and with requirements to make them adaptable to future changes in preferences. A new method is proposed to support contextual data while adding preferences.

III. Proposed architecture

This study proposes a new architecture for decision-making in smart home automation systems. The system uses Large Language Models and proposes methods to add user preferences, in order to select an action according to context and users. It aims to be a proactive system. At every event occurring in the home, the system proposes actions on devices to align the change home state with user needs and preferences. The system supports different types of data thanks to LLMs ability to process data while generating a textual representation of the home based on device states and the action list. This reason leads directly to the use of RAG or directly injecting the knowledge into the prompts for retrieving updated preferences at every execution time of the AI, instead of fine-tuning.

A home simulator is implemented, it takes information on the configurations of sensors and then generates a textual representation, it is also used to generate the list of actions from its data.

This section details the main components designed for the proposed system architecture. Figure 1 shows this architecture, including the simulator.

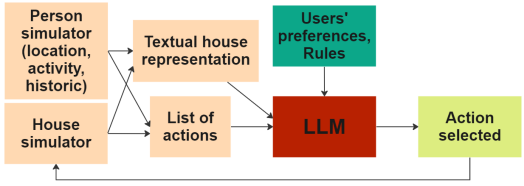

Figure 1. The implemented architecture.

The system generates a user-centred text description of the home and a list of actions, with control over connected devices. It filters the number of relevant actions that can be taken in every situation. For example, it limits some actions that may be prohibited to guarantee user safety. This list of actions is used by the model to select the optimal action.

Concerning contextual data representation editing:

- User positions are listed with their current activity and the history of previous activities.
- The history of previous actions performed in the house is supported.
- All rooms, sensors, and actuators are presented using their names, which the user optionally gives.
- Some sensors and actuators give more global data and control, e.g., temperature sensors, humidity sensors, air-conditioning, etc.

The implementation supports some device categories with a dedicated natural representation, to generate more natural

```
Require: userid, devices_list
for all device_name, device_kind, device_location, device_state in devices_list do
    if device_kind = "actuator" then
        if user_location[userid] = device_location or device_state = 1 then
            devices.append(device_name is device_state)
            action_vector.append(1)
        end if
    end if
end for
devices.append("Interact with user")
action_vector.append(2)
devices.append("No action required")
action_vector.append(0)
return (action_vector, devices)
```

Figure 2. Dynamic Devices: Action builder Algorithm.

sentences adapted to some types of data: lights, CO2 sensors, smart curtains, etc. It could also support any additional data sensors with a generic representation template, using the device name in the smart home environment and data status. Using meaningful naming added by the users helps the system to understand the usage of the device.

The action proposal Algorithm 2 considerably reduces the set of possible actions. It assumes that only devices that are in the room or global devices can be switched on, but that all switched-on devices can be switched off. As far as the list of possible actions is concerned, the idea is to filter the actions supported according to some conditions and device types. This approach is made possible by LLM's native support for a change in the output action space, without requiring training.

Any type of device can be easily supported: it simply has to be added to the representation and the LLM will ingest the data thanks to its internal knowledge, learned during model pre-training. This knowledge allows the LLM to get a natural human description of the home including biased contextual data unlike many conventional home automation systems. The latter do not take advantages of information, such as the names of lights or rooms. This data is transmitted to the LLM using one of the prompting styles, prompting being the way to call the LLM with contextual arguments. The different styles of prompts will be described in the following section.

A common aim for all the prompting styles is to take advantage of the knowledge of the overall world provided by this Large Language model, to handle changes or even new types of sensors added to the representation.

The user preferences and rules block represents a database containing information about the system's basic rules, generalities about human preferences and specific user preferences.

A benchmark is established with predefined context-aware scenarios for evaluation purposes.

Regarding LLMs, recognized top-performing models (relative to their parameter count at the time of evaluation) from reputable research labs were selected. Selected models were run using a local inference engine backend.

IV. Experimentation

Different objectives are defined for the experimentations.

- Evaluate the improvements provided by adding user preferences, with various techniques of doing so.
- Evaluate the improvements of natural language representation of a smart home automation state over a JSON representation, as LLMs are trained on natural language corpus. Even if they contain other kinds of data like code.

In alignment with the objectives of experimentations, different metrics are used for these evaluations:

- Grades: The grade evaluates the response of a model combined to a prompting style when running a scenario. Grade values are defined in Table I.
- Processing time: Total runtime, including the construction of the context data and action list representation, the inferences with the prompting styles, the use of RAG if the prompting style uses it, and the processing of the formatted LLM response.

Eleven evaluation scenarios are defined as starting points, with predefined actions graded 0-2 based on user satisfaction. Table I presents the scenarios by name and associated reward values. A category is associated with each scenario, the goal being to group the scenario with the name of the main evaluated ability. The database of preferences and rules is defined in a single file for all scenarios. These data are naturally written sentences. At the end of each one, an information level is

TABLE I. Scenario responses with grades, associated answers, and evaluation categories.

Scenario Name	Grade	Associated Answer	Category
Out of bed at night	2	Turn on auxiliary light or main light with reduced luminosity level	Safety
	1	Turn on main light	
	0	Everything else	
Watching TV: late evening	2	Turn on auxiliary light or main light with reduced luminosity level	Comfort
	1	Turn on main light, open curtains, discuss	
	0	Everything else	
Out from bed issue with CO2	2	Inform user of risk	Safety
	1	Do an action and inform the user of risk	
	0	Everything else	
Going back to bed at night	2	Turn on auxiliary light or main light with reduced luminosity level	Safety
	1	Turn on main light	
	0	Everything else	
Evening sleeping: TV ON	2	Turn off TV	Preference
	1	Turn off anything on	
	0	Everything else	
At dinner watching TV	2	Turn on auxiliary light or main light with reduced luminosity level, open curtains	Preference
	1	Turn off the main light, do nothing	
	0	Everything else	
User out: TV is on	2	Turn off TV, turn off HVAC	Comfort
	1	Turn off all lights	
	0	Everything else	
Too low temperature	2	Turn on HVAC	Preference
	1	Open Curtains	
	0	Everything else	
Low luminosity day	2	Open curtains	Preference
	1	Turn on any light in the room	
	0	Everything else	
Failed curtains	2	Turn on any light of the room	Comfort
	1	Open curtains	
	0	Everything else	
Forgot to turn off lights	2	Turn off any lights, or HVAC	Preference
	1		
	0	Everything else	

recorded: Rules, Preferences, Generality. The idea is to transmit to the LLM the importance of each data through keywords. Generality is considered the least important, Preferences the second most important, and Rules the most important. The database includes some preferences, generality and some rules. It is designed to handle some scenarios, help in some others but does not provide a solution for all scenarios. The data are fed into a vector database so that RAG may be used instead of prompts to convey them to the LLM. Four different prompting styles are compared on all the scenarios:

- direct: A system prompt and a prompt to request direct answers in the specified format.
- directPref: A system prompt with the preferences, rules and generality from the database and a prompt to request answers in the specified format.
- OpenQuestion: a two-steps chain. First a system prompt and a prompt to request "a list of 3 main problems". For each of the 3 problems, the LLM is invited to use RAG to get the 3 closest preferences. Then a prompt to request answers in the specified format.
- ThreeQuestion: a three-steps chain. First a system prompt and a prompt to request "a list of 3 main problems". For each of the 3 problems the LLM is invited to use RAG to get the 3 closest preferences. Then a prompt to request answers in the specified format (twice). Finally a prompt to select the best answer in the specified format.

A common point between all the prompting styles is the action expected in the output: as mentioned above, most supported devices, such as lights or HVAC systems, are considered as switches in the action list; therefore, in the output of all prompting styles in addition to a "reasoning" and an "action" key, three optional keys are available: temperature, luminosity and explanation. It enables the model to respectively modify the temperature of an HVAC system when executing a related action, modify the luminosity by dimming a light or give an explanation to transmit a sentence to the user.

Two ways of representing the state of the house data are implemented, both using the same input data:

- JSON: A JSON representation
- Textual: A fully natural textual representation

The implementation of the system is evaluated using various open-sources LLMs, including:

- Starling Alpha 7B [14]- 8bpw (bits per weight)
- Qwen 1.5 14B [15] - 5bpw
- Qwen 1.5 72B [15] - 3.5bpw

The three models are selected for their performance and for covering the three main open-source model sizes. They are used to evaluate the impact of proposed prompting methods and data representation.

Qwen 1.5 72B, with around 72 billion parameters, is currently one of the best models available open-source. Starling 7B Alpha, with around 7 billion parameters, is an excellent smaller model. It is based on Mistral 7B, an efficient model for its size on various benchmarks. Qwen 1.5 14B model, a smaller version of Qwen 1.5 72B, is selected to add an intermediary model.

All these models are used with versions that are quantized [16], a technique used to reduce inference time and memory footprint, the quantization chosen for each model is given in bits per weight (bpw). With their quantization, they require around 8GB, 12GB, and 44GB of memory respectively.

Every model is evaluated on local instances, served locally with an engine-based API backend, using TabbyAPI [17], based on ExLLamaV2 multiple GPUs (graphics processing units) and without automatic splitting, using a workstation equipped with a Ryzen 9 7950x, 96GB of DDR5 memory running at 5600mhz and 2 Nvidia RTX 4090, each with 24 GB dedicated memory, running Ubuntu 23.10.

Experiments are carried out beforehand on various uncontrolled scenarios to define LLM parameters. With the sole aim of reducing non-determinism from one cycle to the next, the final parameters modified from the default engine parameters are as follows:

- max_tokens, maximum number of tokens in output: 300
- min_p, minimum percentage value that a token must reach to be considered (Value is scaled based maximum token probability): 0.05
- temperature, parameter that regulates the randomness: 0.2

The RAG is implemented using Langchain [18] with an inference engine from HuggingFace [19], to locally execute an embedding model: BAAI/bge-large-en-v1.5 [20], and an Elasticsearch [21], local instance is used as vector database, both instances running on main processor and associated memory.

To evaluate system performance, each scenario is executed 10 times with each prompting style, and a grade is given to each response. Results then count the total number of points for each prompting category in general, and also for each defined metric associated with each question. The system's complete processing time is also measured, in order to estimate the average latency of the different prompting styles.

The theoretical random action grade for each scenario is calculated as a baseline using the following (1).

$$\text{grade}_s = \frac{\text{num\_actions\_graded\_1}_s + 2 \times \text{num\_actions\_graded\_2}_s}{\text{total\_number\_of\_actions}_s} \quad (1)$$

With num_actions_graded_X the number of actions associated with grade X. Figure 3 shows an example of a scenario data representation using proposed natural language textual representation, it shows an example of the generated contextual representation transmitted to the LLM on scenario 1.

As previously mentioned, several data types have been included in the representations: numerical values, boolean values, and character strings. Using an LLM to process context allows the system to support all these data types.

Regarding the action list building algorithm, in one of the scenarios for example, it reduces the set of actions from more than 18 to 6 actions, which represents a 3-fold reduced set in this case. This reduced set takes into account the fact that lighting appliances are basic switches and cannot be set with

```
Current State of the House:
User 1 is in the Livingroom. User is watching TV.
Previously: User was watching TV

Livingroom: Curtains are Closed.
Lights: main, floor lamp are respectively Off, Off.
There is a TV in the room and its state is on.
CO2 level in room is 513ppm.
.....................................
House was cleaned today, expected cleaning one time a week.
Centralized HVAC system is on with objective to 20°C.
Entrance smart Door is locked.
Time: 10:21 PM
Global house temperature is 20°C, outside temperature is 5°C.
```

Figure 3. Extract of natural data representation on the scenario "Out of bed at night".

different luminosity levels. No further studies assessed the improvement provided by the action filtering for the LLMs.

This limited set of actions makes it interesting to use a model randomly selecting one action from the set as a baseline. With a reduced action space, a random answer may be good.

V. Results

This section will present the results of our simulations, focusing on various aspects. Initially, we will explore the impact of data representation, followed by an analysis of the results concerning user preferences.

A. Data representation

The first analysis of the results focuses on the advantage of using a natural representation versus a JSON representation. Table II shows an average difference between results using

TABLE II. Comparing the two contextual representations: average grade of models per execution per scenario.

Model	JSON	Natural
Qwen 1.5 72B	1.25	1.18
Qwen 1.5 14B	0.63	0.99
Starling Alpha 7B	1.03	1.18

JSON representation, or the most natural representation. It shows that the larger models are almost stable independently of the contextual representations, and on average even slightly better using JSON representation, by around 5.9%. However, on smaller models, the natural language representation greatly improves performances, with a 14.6% increase in average performance using Starling 7B, and a 57.1% increase using Qwen 1.5 14B. In terms of processing time, results are quite similar on average for models with both representations. On average, accross all models, it leads to an increase in performance of 21.9%, despite the results on the larger Qwen model, making the natural representation more efficient.

B. User preferences

Figure 4 depicts the average grades obtained by the 3 chosen models on their responses to scenarios with the 4 different prompting styles and the 2 distinct representation types.

It first shows that Qwen 72B model is much more stable than the other two and that the prompting style does not have as much impact on response quality. Regarding the two smaller models, larger inconsistency in results can be noted as varying with the chosen prompting styles, particularly with Qwen 14B model. The results of LLMs are compared with a baseline corresponding to the random choice of an action. The results remain on average behind any model without preferences (37.1% below the worst result with the "direct" prompting style). However, thanks to the algorithm reducing the set of actions and the fact that multiple responses are acceptable on each scenario, the random baseline obtains grades that are sometimes better than LLMs.

The sequence of multiple questions requires the model to be consistent and to respect the expected instruction format. Furthermore, given that only prompt engineering is used to ensure the format, some prompting styles with multiple questions may lead to invalid responses, forcing the model to take a default action in the proposed setting. This default action is set to do nothing and to inform the user that it has failed to act. This reduces the performance of some models with some prompting styles. In Figure 4, failure ratio measures the ratio of invalid responses. Qwen 14B results are especially below this baseline because they failed to answer in a large number of scenarios

On average, "directPref" prompt models achieved a gain of 11.3% over basic prompt with natural representation, and even 20.0% with JSON representation.

The best results are achieved with JSON representation for "OpenQuestion" and "ThreeQuestion", leading to a 28.6% improvement over the "direct" prompting style.

With the natural representation, Qwen 14B model gives almost stable results with all prompting styles except "OpenQuestion". The main difference with "OpenQuestion" seems to be linked to a high failure rate, as shown in Figure 4. With JSON representation, failures are so high with all advanced prompting styles that the best results are obtained with the most basic prompt. In this case, "direct" prompting style is 30.7% more efficient than the "directPref" prompting style.

On Starling 7B, with both representations, the best results are obtained with "directPref", on the Starling representation, its results are 52.3% better than the direct representation and respectively 8.1% and 6.3% better than "OpenQuestion" and "ThreeQuestion" prompting styles.

In addition, the average processing time is reduced by 35.6% on Starling 7B Alpha using the "directPref" prompting style instead of "direct".

With JSON representation, the averaged grade results follow the same trend, but with lower overall values.

Table 4 shows that RAG-based prompting models can lead to better results than a single prompt containing all the data, as it is visible Qwen 72B. However, to date, LLM inference is still slow, and the use of complex prompts leads to a loss of accuracy in the test scenarios with smaller models.

Adding inference time to the balance with Table III highlights Starling 7B Alpha results with "directPref". It outperforms almost all others except Qwen 72B with JSON

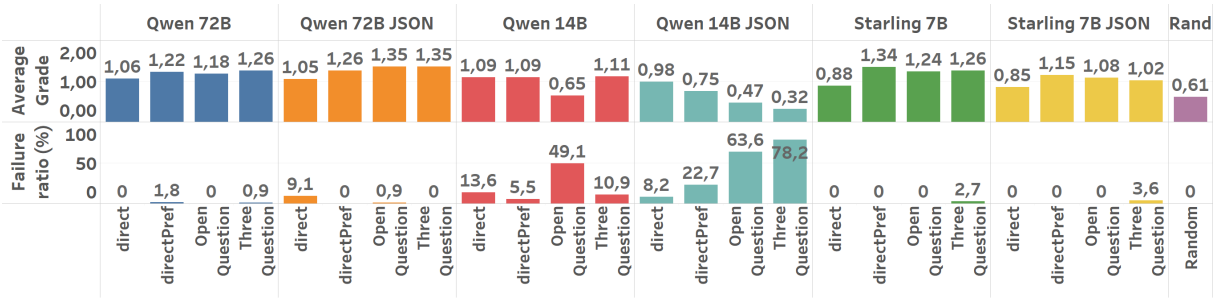

Figure 4. Average grades by model, data representation, and prompting styles.

TABLE III. COMPARISON OF AVERAGE GRADES FOR EACH PROMPTING STYLE AND MODELS WITH INFERENCE TIME.

Model	Prompting Style	Average grade	Proces. time (s)
Qwen 72	direct	1.06	9.04
	directPref	1.22	7.01
	OpenQuestion	1.18	24.43
	ThreeQuestion	1.26	42.13
Qwen 72 JSON	direct	1.05	9.02
	directPref	1.26	6.71
	OpenQuestion	1.35	25.20
	ThreeQuestion	1.35	41.47
Qwen 14	direct	1.09	1.22
	directPref	1.09	1.16
	OpenQuestion	0.65	8.83
	ThreeQuestion	1.11	15.54
Qwen 14 JSON	direct	0.98	1.28
	directPref	0.75	1.21
	OpenQuestion	0.47	9.39
	ThreeQuestion	0.32	16.62
Starling	direct	0.88	0.73
	directPref	1.34	0.47
	OpenQuestion	1.24	3.98
	ThreeQuestion	1.26	6.23
Starling JSON	direct	0.85	0.48
	directPref	1.15	0.48
	OpenQuestion	1.08	3.77
	ThreeQuestion	1.02	6.20

representation and prompt chaining, but is 53.6 times faster, with a lower grade of just 0.7%.

With Qwen 72B, the use of more complex models (OpenQuestion, ThreeQuestion) can lead to better results, as seen previously in particular with JSON representation, but this comes with a trade-off: inference time. Qwen 72B model is already much slower due to its number of parameters, and due to the number of operations required to produce a single token. For instance, an inference with "directPref" takes 6.86 seconds on average (JSON and textual representation), whereas using "OpenQuestion" (which is around 60% faster than "ThreeQuestion") is 3.6 times slower.

In the current state of this type of hardware, it is impossible to consider them as a viable alternative for managing a smart home automation system, with such reaction times.

VI. DISCUSSION

This section mainly discusses the results of the LLMs and prompting techniques chosen in order to make choices for real experiments. The advantages of this study are highlighted as well as the new challenges that are raised.

As seen previously, using larger LLM, such as Qwen 72B, allows greater stability in preference-free scenarios. Indeed, Qwen 72 is 19.9% better than Starling 7B in this case. However there are drawbacks, the first being the inference time as mentioned, and the second being the hardware infrastructure required. The quantized version of Qwen 1.5 72B requires 44GB of memory compared with around 8GB for Starling 7B Alpha with lower quantization. Compared with Qwen 72B model using JSON representation, Starling 7B Alpha with "directPref" takes advantage of natural representation and achieves almost similar performances with much more complex prompting techniques. This makes the approach of using Starling 7B Alpha with this prompting style a good choice for future work. It gives similar performances concerning grades, and has a relatively low inference time (Average: 0.47s). In addition, compared to Qwen 1.5 72B with natural representation and no preferences, Starling 7B Alpha's performance is 26.4% better using "directPref", with a processing time almost 20 times faster.

The results show the advantage of adhering to preferences. Drawbacks appear, however, with the additional average computation time for "OpenQuestion" and "ThreeQuestion" prompting styles, which use RAG. Using RAG brings no advantage in most cases. This is certainly due to the relatively small database. If the system required a larger database of preferences and rules, the results might have been different as it would not have been possible to give them directly through "directPref" prompting style. Based on current results, the best choice for a use case with a larger database would remain Starling 7B Alpha, with "OpenQuestion" prompting style and natural representation, as it provides results that are aligned with users preferences, with only 8.1% less average grade than Qwen 1.5 72B, while keeping an acceptable average inference time (3.98 seconds vs 25.20 seconds).

The use of smaller models poses the challenge of enforcing the output format. New methods have recently been proposed, such as Outlines [22], to strengthen the output grammar. They should be tested soon in order to analyze the cost in inference time and the improvement of the results of this paper.

Another approach is fine-tuning. Applying it to user preferences is impractical as they are constantly changing. Applying it to smart home domain would improve the analysis of contextual data while allowing the system to ensure the output

format.

Although the proposed system is not comparable to other works due to its completely new approach, it has the advantages of its nature, as well as drawbacks. Firstly, it is not deterministic. Each configuration has been run 10 times, but some results are different each time, which means that the results of the system cannot be certified and a safety layer must be developed to secure some actions. Secondly, LLM queries are slow and require significant computating and memory resources: adaptability without training comes at a cost.

VII. Conclusion and Future Work

This paper presents a new architecture for a smart home automation system, using LLMs with user preferences to enhance personalised user experiences. This approach leverages the general knowledge provided by LLMs and combines it with naturally written rules and preferences to make contextually relevant decisions in line with user preferences. This architecture is proactive, able to adapt to any change in the environment thanks to the robustness provided by LLMs.

The experimental results demonstrate the potential of this architecture to improve alignment with user preferences compared with an implementation without user preferences, showing up to 52.3% performance increase, with an average processing time reduced by 35.6% on Starling 7B Alpha LLM using "directPref" prompt style compared to "direct". Additionally, the performance is 26.4% better than that of the largest models evaluated without preferences, while also achieving a processing time that is nearly 20 times faster.

The study showed that, particularly with small models, using a natural representation instead of a JSON one leads to an increase in performance, with an average 21.9% increase.

Although the system shows promising results on a set of defined scenarios, it also presents challenges due to stochastic behaviour and a slower inference time compared to traditional machine learning methods. These drawbacks are offset by the system's ability to adapt dynamically - without retraining - to changes in preferences, appliances, and home configuration.

Future work will focus on implementing the system in a real-world smart home middleware system, such as OpenHAB to evaluate its performance with real users.

References

[1] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999, ISSN: 1559-1662. DOI: 10.1145/329124.329126.

[2] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, Feb. 2001, ISSN: 1617-4909. DOI: 10.1007/s007790170019.

[3] P. Rashidi and D. Cook, "Keeping the Resident in the Loop: Adapting the Smart Home to the User," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 39, no. 5, pp. 949–959, Sep. 2009, ISSN: 1083-4427, 1558-2426. DOI: 10.1109/TSMCA.2009.2025137.

[4] A. Radford *et al.*, *Language models are unsupervised multitask learners*, 2019.

[5] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 9459–9474.

[6] P. F. Oliveira, P. Novais, and P. Matos, "Adaptive environment system to manage comfort preferences and conflicts," in *Optimization, Learning Algorithms and Applications*, Cham: Springer International Publishing, 2022, pp. 685–700, ISBN: 978-3-031-23236-7.

[7] A. R. Ruiz *et al.*, "Leveraging commonsense reasoning towards a smarter Smart Home," en, *Procedia Computer Science*, vol. 192, pp. 666–675, 2021, ISSN: 18770509. DOI: 10.1016/j.procs.2021.08.069.

[8] S. S. Shuvo and Y. Yilmaz, "Home energy recommendation system (hers): A deep reinforcement learning method based on residents' feedback and activity," *IEEE Transactions on Smart Grid*, vol. 13, no. 4, pp. 2812–2821, 2022. DOI: 10.1109/TSG.2022.3158814.

[9] Z. Zhang, Y. Su, M. Tan, and R. Cao, "Fusing domain knowledge and reinforcement learning for home integrated demand response online optimization," *Engineering Applications of Artificial Intelligence*, vol. 121, p. 105 995, 2023, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2023.105995.

[10] Z. Peng, X. Li, and F. Yan, "An adaptive deep learning model for smart home autonomous system," in *2020 International Conference on Intelligent Transportation, Big Data and Smart City (ICITBS)*, Los Alamitos, CA, USA: IEEE Computer Society, Jan. 2020, pp. 707–710. DOI: 10.1109/ICITBS49701.2020.00156.

[11] E. King, H. Yu, S. Lee, and C. Julien, ""Get ready for a party": Exploring smarter smart spaces with help from large language models," *Unpublished*, 2023. DOI: 10.48550/ARXIV.2303.14143.

[12] E. King, H. Yu, S. Lee, and C. Julien, "Sasha: Creative goal-oriented reasoning in smart homes with large language models," *Unpublished*, 2023. DOI: 10.48550/ARXIV.2305.09802.

[13] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, Association for Computing Machinery, 2016, pp. 263–273.

[14] B. Zhu, E. Frick, T. Wu, H. Zhu, and J. Jiao, *Starling-7b: Improving llm helpfulness and harmlessness with rlaif*, Nov. 2023.

[15] J. Bai *et al.*, "Qwen technical report," *Unpublished*, 2023.

[16] Turboderp, *Exllamav2*, https://github.com/turboderp/exllamav2, 2023.

[17] TheRoyalLab, *Tabbyapi*, https://github.com/theroyallab/tabbyAPI, 2023.

[18] O. Topsakal and T. C. Akinci, "Creating large language model applications utilizing langchain: A primer on developing llm apps fast," in *International Conference on Applied Engineering and Natural Sciences*, vol. 1, All Sciences Academy, 2023, pp. 1050–1056.

[19] HuggingFace, *Text embeddings inference*, https://github.com/huggingface/text-embeddings-inference, 2023.

[20] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, *C-pack: Packaged resources to advance general chinese embedding*, 2023. eprint: 2309.07597 (cs.CL).

[21] Elastic, *Elasticsearch*, https://github.com/elastic/elasticsearch, 2015.

[22] B. T. Willard and R. Louf, "Efficient guided generation for llms," *arXiv preprint arXiv:2307.09702*, 2023.