DevPiolt: Operation Recommendation for IoT Devices at Xiaomi Home

Yuxiang Wang[†] Siwen Wang[§] Haowei Han[†] Ao Wang[†] Boya Liu[§] Yong Zhao[§]
Chengbo Wu[§] Bin Zhu[§] Bin Qin[§] Xiaokai Zhou[†] Xiao Yan[†] Jiawei Jiang[†] Bo Du[†]
[†] School of Computer Science, Wuhan University [§] Xiaomi
[†] {nai.yxwang,haowei.han,aowang2024,xiaokaizhou,yanxiaosunny,jiawei.jiang,dubo}@whu.edu.cn
[§] {wangsiwen,liuboya,zhaoyong6,wuchengbo1,zhubin,qinbin}@xiaomi.com

Abstract

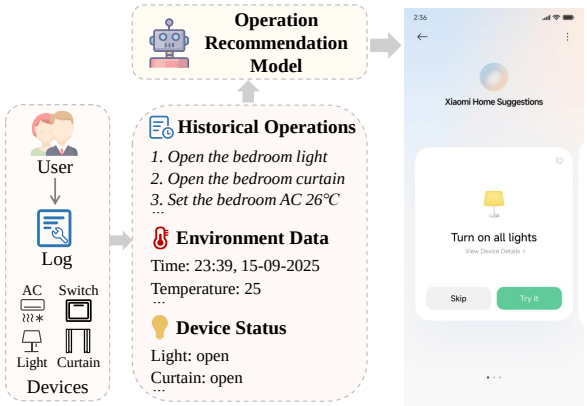
Operation recommendation for IoT devices refers to generating personalized device operations for users based on their context, such as historical operations, environment information, and device status. This task is crucial for enhancing user satisfaction and corporation profits. Existing recommendation models struggle with complex operation logic, diverse user preferences, and are sensitive to suboptimal suggestions, limiting their applicability to IoT device operations. To address these issues, we propose DevPiolt, an LLM-based recommendation model for IoT device operations. Specifically, we first equip the LLM with fundamental domain knowledge of IoT operations via the pre-training and fine-tuning on delicately constructed device operation datasets. Then, we employ direct preference optimization to align the fine-tuned LLM with specific user preferences. Finally, we design a confidence-based exposure control mechanism to avoid negative user experiences from low-quality recommendations. Extensive experiments show that DevPiolt significantly outperforms baselines on all datasets, with an average improvement of 69.5% across all metrics. DevPiolt has been practically deployed in Xiaomi Home app for one quarter, providing daily operation recommendations to 255,000 users. Online experiment results indicate a 21.6% increase in unique visitor device coverage and a 29.1% increase in page view acceptance rates.

Keywords

Recommendation, Large language model, IoT device management

1 Introduction

Xiaomi Home, a smart living ecosystem under Xiaomi, primarily oversees the company's smart home business. Xiaomi Home has developed an intelligent service domain encompassing home control, health monitoring, and security protection, using IoT technology to connect and coordinate with terminal devices. Xiaomi Home's product line encompasses three main categories: smart home, wearable devices, and smart travel devices, which are further divided into over 260 subcategories, including smart air conditioners (ACs), switches, and curtains. In 2024, its global device shipments reached 60 million units, representing a 20.1% increase from the previous year, with a cumulative total sales volume of 944 million units. To facilitate the management of these smart devices, Xiaomi Home has developed a complementary mobile application (i.e., Xiaomi Home app) dedicated to promoting the in-depth management of IoT devices in daily life scenarios. As of June 2025, the monthly active user base of the Xiaomi Home app reached 113 million, with over 20.5 million users owning five or more smart devices.



(a) Log data (b) Operation prompt (c) Operation suggestion

Figure 1: The workflow of device operation recommendation in Xiaomi Home app.

Operation recommendation is a core service of Xiaomi Home, designed to provide users with personalized device operation suggestions. This service benefits both users and businesses. For users, the increasing variety and functionality of IoT devices make it increasingly inconvenient to achieve precise control based on the current environment. Operation recommendation can reduce the frequency of manual operations, thereby enhancing user efficiency and interaction experience. For businesses, it allows for the optimization of product design and the increased sales of related devices through user feedback on these recommendations. As shown in Figure 1, the recommendation engine first collects the user's historical operation sequences (e.g., turning on lights, setting AC temperature), device lists (e.g., lights, ACs), and environmental data (e.g., time, temperature). These data are then fed into a recommendation model to predict the user's potential device operation needs. Finally, it provides operation suggestions to the user in text form. For example, if a smart device detects that someone is in the bedroom during nighttime, the operation recommendation predicts that the user may need to "*close the bedroom curtains*". If the user accepts this suggestion, the Xiaomi Home app will perform the relevant operations via a unified device control interface.

The recommendation model is the core component of operation recommendation and has been extensively studied in both academia and industry [5, 17, 23, 34, 36]. These models can generally be categorized into two types [26]: Feature-based Recommendation Model (FRM) [16, 24] and LLM-based Recommendation Model (LRM) [28, 30]. FRMs use carefully designed features as input to construct a mapping relationship between users and items [36]. For

Table 1: Accuracy comparison results for CGC, DeepSeek and DevPiolt on the *Whole* dataset (refer to Section 4.1).

Method	Whole		
	EM-Acc	LM-F1	Rule
CGC [23]	19.95	17.72	23.48
DeepSeek [4]	32.97	44.68	41.45
DevPiolt	**44.33**	**57.49**	**53.03**

instance, the Customized Gate Control model [23] (CGC) dynamically controls the activation of different cross-features through learnable gating parameters, effectively filtering out the most relevant interactions for users. LRM methods, such as CALRec [13] and MLLM-MSR [29], transform historical action sequences and item features into natural language, relying on the semantic understanding capabilities of LLMs to uncover the connections between users and items. Given the powerful text generation capabilities of LLMs, which can produce suggestions for multiple devices and operations along with corresponding textual descriptions, we develop the operation recommendation based on LLMs.

Challenges. However, when applied to the operation recommendation tasks, Table 1 shows that current models exhibit low accuracy. We identify three key challenges in implementing device operation recommendations using existing recommendation models:

❶ **Complex Operation Logic**. Operation recommendations must recall multiple item combinations (e.g., devices and actions). However, existing recommendation models typically handle a single type of item. Furthermore, there is a sequential logic between the operations, such as the need to turn on the power outlet before opening the AC, rather than the reverse.

❷ **Diverse User Preferences**. User preferences significantly influence operation recommendations, encompassing both long-term stable preferences (e.g., the habit of opening curtains at a specific time every day) and short-term behavioral patterns (e.g., avoiding frequent toggling of the AC). However, current methods often struggle with diverse user preferences, thereby impeding the effective capture of these specific tendencies.

❸ **Sensitive to Inappropriate Suggestions**. The vast pool of candidate operations inevitably includes many suboptimal inappropriate that are illogical and severely degrades the user experience. Therefore, a critical challenge is the prospective identification and filtering of these inappropriate candidates to ensure only high-quality recommendations are presented to the user.

To address these challenges, we propose DevPiolt, an LLM-based operation recommendation framework, including four core components: *pre-training*, *fine-tuning*, *recommendation refinement* and *exposure control*. Specifically, for Challenge ❶, we first pre-train the LLM on a large amount of user historical operations to enable it to learn feasible operation combinations. Then, we fine-tune the LLM with fine-grained operation corpus, including historical operations, current environment data and operable devices. The fine-tuning allows LLM to understand logical relationships between operations based on context. Additionally, we pair a textual description with each operation action in the corpus, and prompt the LLM to output both operation action and corresponding descriptions. For Challenge ❷, we construct an operation dataset containing positive and negative sample pairs, which are used for direct preference optimization (DPO). This method maximizes the relative likelihood difference between positive and negative samples, enabling the LLM to learn specific user preferences from a mass of historical operations. Finally, for Challenge ❸, we introduce an exposure control technique, which determines whether to present operation suggestions to users based on confidence scores, thus avoiding suboptimal recommendations that may lead to a negative user experience.

To train the operation recommendation model, we sample 45,000 operational entries from user logs every three months for pre-training. The fine-tuning corpus contains 15,000 manually annotated samples. The test set comprises 4,882 labeled operations, covering 21 types of IoT devices. We further subdivided the test set based on the device count and device category, resulting in 7 and 17 sub-datasets, respectively. DevPiolt has been practically deployed in the Xiaomi Home app for one quarter, serving 255,000 users daily and generating 326,000 operation suggestions per day.

We evaluate DevPiolt and its designs against baselines on various datasets from Xiaomi Home. We also introduce three evaluation metrics for the qualitative assessment: exact match accuracy, loose match F1 score and rule score. The main results show that DevPiolt outperforms the baselines on all datasets. Compared to the best-performing baseline, the average gains in the three metrics are 95.5%, 58.5%, and 54.0%, respectively. Ablation studies validate that each proposed technique incrementally improves accuracy when added to a base LLM. Additionally, parameter sensitivity experiments find that model accuracy improves with LLM scale and that setting the context and history sequences to a moderate length yields optimal results. Online experiments indicate that DevPiolt achieve a 21.6% increase in unique visitor (UV) device coverage and a 29.1% increase in page view (PV) acceptance rates.

In this paper, our contributions are summarized as follows:

- We find that existing methods cannot work well for operation recommendation tasks and identify three major challenges: complex operation logic, diverse user preferences, and sensitive to suboptimal suggestions.
- We propose DevPiolt and design pre-training and fine-tuning to learn operation knowledge on operation datasets [1], a DPO-based method to reinforce specific user preferences, and an exposure control technique to filter unsatisfied recommendations.
- We conduct extensive experiments to evaluate DevPiolt. The results show that DevPiolt significantly outperforms the baselines, and our designs are effective in improving accuracy.

2 Preliminaries

Operation Record Data. To enable the LLM to learn users' operation preferences, we construct a high-quality dataset of device operation data. As shown in Figure 2 (a) and (b), we aggregate historical operations from the raw environment and action logs of the Xiaomi Home app. Environment logs are the environmental detection data from IoT sensors, such as room temperature and

[1]We will open-source the operation datasets to enhance further exploration in the community.

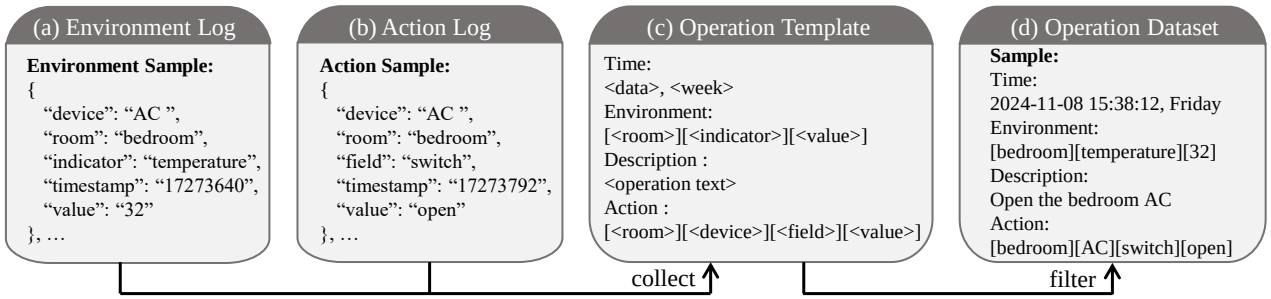


Figure 2: The process of constructing operation dataset.

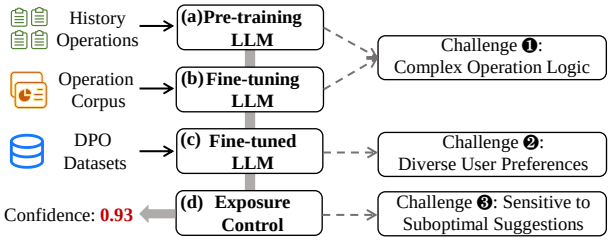


Figure 3: The overview of DevPiolt.

humidity. Action logs are user operation records on smart devices, either through voice or direct control. We record the *device*, *room* and *timestamp* of the operation action, and use *field* and *value* to represent the type and content of operation, respectively.

To translate complex device operation into textual content that LLMs can easily understand, we design an operation template in Figure 2 (c) to filter the log data. We denote a device operation dataset $\mathcal{D} = \{\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_N\}$, where $\mathcal{H}$ denotes the user's historical operation sequences and N is the number of users. Furthermore, each sequence $\mathcal{H}_i = \{O_1, O_2, ..., O_l\}$ includes multiple operations O, where l is the operation count for one user. As shown in Figure 2 (d), each operation consists of the following four components:

- *Time* O_i^{time}. We convert the execution timestamp of an action into a date format (i.e., *yy-mm-dd*) and the day of the week.
- *Environment* O_i^{env}. We construct a triplet $O_i^{env} = \{[room], [indicator], [[value]\}$ to describe the current environment.
- *Description* O_i^{desc}. This is an operation description given by the user through voice or the Xiaomi Home app to the device.
- *Action* O_i^{act}. We extract the operation actions executed by users and represent the action content precisely using a quadruplet $\{[room], [device], [field], [value]\}$.

The Operation Recommendation Task. The goal is to generate an action quadruple O_i^{act} and a description O_i^{desc} based on the historical operations, current environment, and user's devices. Notably, O_i^{act} and O_i^{act} are semantically equivalent in terms of operation meaning.

3 The DevPiolt Framework

Overview. To address the four challenges faced in operation recommendations, we propose DevPiolt, which comprises four modules: *pre-training*, *fine-tuning*, *recommendation refinement*, and *exposure control*. As shown in Figure 3, for challenge 1, we first concatenate the user's historical operation sequences to pre-train an LLM, and

Operation Recommendation Corpus

The following is user operation history:
Time $\mathcal{O}_1^{time}$: 2024-10-08 15:38:12, Tuesday
Enviroment $\mathcal{O}_1^{env}$: [bedroom][temperature][32]
Action $\mathcal{O}_{1,1}^{act}$: [bedroom][AC][switch][on]
$\mathcal{O}_{1,2}^{act}$: [bedroom][AC][temperature][24]
Operation $\mathcal{O}_1$
...

Based upon user operation history and following candidate divices, provide the most likely device operation for the user.
Candidate Devices $\mathcal{C}$:
1. Fresh Air Unit. Room: Bedroom, Status: On
2. Xiaomi Air Conditioner Natural Wind 1.5 HP. Room: Bedroom, Status: Off
3. Xiaomi Smart Tankless Gas Water Heater S1. Room: Living Room, Status: On
Current Environment:
Time $\mathcal{O}_l^{time}$: 2024-11-25 07:52:40, Monday, workday
Environment $\mathcal{O}_l^{env}$: [bedroom][temperature][22]

Action $\mathcal{O}_l^{act}$***:***
[bedroom][AC][switch][on]
[bedroom][AC][temperature][24]
[living_room][water heater][set hot water preheating][on]
Description $\mathcal{O}_l^{desc}$***:***
Turn on the bedroom AC to 24°C and enable living room water heater preheating.

Figure 4: An operation dataset sample. The black text represents the operation recommendation prompt $\mathcal{P}$, and the italicized part indicates the target used for loss calculation. The pre-training only includes operation history, while fine-tuning additionally includes environment and device data.

then fine-tune the pre-trained LLM using a fine-grained operation corpus, enabling the model to understand the logic of the operations and generate reasonable suggestions. For challenge 2, we further utilize direct preference optimization to refine the fine-tuned LLM. This technique allows the model to learn the specific user preferences. Finally, for challenge 3, we implement exposure control by calculating a confidence score, which determines whether to display the recommendation results to the user.

3.1 Pre-training and Fine-tuning

As highlighted in the challenge 1, while LLMs can leverage their vast world knowledge to reason about intricate relationships within operations, their performance is suboptimal due to a lack of domain knowledge in IoT device operations. Furthermore, directly employing a naive LLM for the recommendation task does not guarantee that its output will conform to a legal operation structure.

Pre-training. To ensure the LLM is able to understand the device operation, we pre-train the base LLM on user historical sequence

$\mathcal{H} = \{O_1, O_2, ..., O_l\}$. As shown in the "user operation history" part of Figure 4, each operation instance O_i is serialized into a single textual sequence that includes the time O_i^{time} and environment O_i^{env}, followed by the corresponding a_i action(s) $O_i^{act} = \{O_{i,1}^{act}, ..., O_{i,a_i}^{act}\}$. For instance, in O_1, the user performs logical actions: turning on the AC and setting its temperature to 24 degrees when the bedroom temperature reaches 32 degrees. Our training approach reframes the task as next-action prediction, which is a natural fit for LLMs. We concatenate the user's historical actions $O_{<i}$, along with current time O_i^{time} and environmental information O_i^{env}, as conditional inputs for the model to predict the user's next action. This is formally achieved by minimizing the negative log-likelihood loss, a standard objective for training LLMs:

$$\mathcal{L}_{pt} = -\sum_{O_i \in \mathcal{H}} \log \mathrm{P}(O_i^{act} | O_{<i}, O_i^{time}, O_i^{env}) \tag{1}$$

By learning to predict actions from context, the model internalizes the logic of device operations. For example, it learns device-specific functionalities (e.g., a light can only be on or off, not set to 25 degrees) and valid value spaces (e.g., an AC's temperature must be a number within a reasonable range). Finally, to ensure the model retains its general knowledge and avoids catastrophic forgetting, we create a balanced training dataset by mixing our operation pre-training corpus with a general-domain corpus, such as ShareGPT [3] and WuDao [32], at a 1:1 ratio.

After the pre-training, the model is equipped to understand the domain of IoT operations. However, two key gaps remain. 1) The model *struggles to apply its operational knowledge to recommendations*, such as by failing to suggest actions for the user's available devices. 2) It is *unable to generate user-friendly natural language descriptions* (e.g., "Turn on the AC") to present to the user instead of the raw structured operations (e.g., [AC][switch][on]).

Fine-tuning. To bridge the two gaps, we fine-tune the LLM on a specialized training corpus, which contains 15,000 manually annotated operation instances. Each training instances consists of an operation recommendation prompt, the ground-truth action(s) and a corresponding textual description. As shown in Figure 4, we construct the operation recommendation prompt $\mathcal{P}$ by the user's operation history (excluding the final operation) $O_{<l}$, along with the current time O_l^{time} and environment information O_l^{env} and a list of candidate devices C, which is included to train the model to recommend valid operations based on the available devices. We fine-tune the LLM using LoRA [8]. It introduces low-rank matrices for incremental parameter updates in the pre-trained LLM, thereby training only a small number of learnable parameters.

We fine-tune the model using a multi-task objective to predict both the final action(s) O_l^{act} and the corresponding description O_l^{desc}. Our initial approach is to generate both outputs simultaneously, by minimizing the following joint probability objective:

$$\mathcal{L}'_{ft} = -\log \mathrm{P}(O_l^{act}, O_l^{desc} | \mathcal{P}) \tag{2}$$

However, we find this method yield suboptimal accuracy, as the model struggle to master both tasks at once. Therefore, the model first predicts the action and then generates the description. This factorizes the action-description objective $\mathcal{L}_{ad}$ as follows:

$$\mathcal{L}_{ad} = -(\log \mathrm{P}(O_l^{act} | \mathcal{P}) + \log \mathrm{P}(O_l^{desc} | \mathcal{P}, O_l^{act})) \tag{3}$$

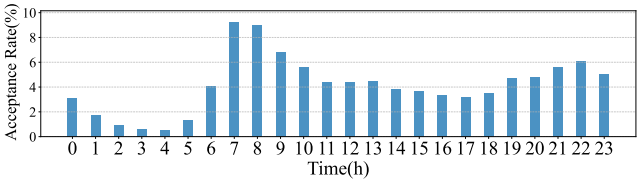


Figure 5: The acceptance rate of curtain operation recommendations by users at different times of the day.

Insight. We adopt an *action-first* strategy, generating the action before the description, to establish a more logical, concrete-to-abstract learning flow. Predicting the structured and precise action serves as the core task, providing a clear and unambiguous "anchor" for subsequent text generation. Once the action is determined, generating the description becomes a simpler and more grounded summarization task. In contrast, a *text-first* strategy, which generates the free-form description first, would require the model to infer a precise action from potentially vague or incomplete text. This process is more challenging and error-prone. Consequently, our approach ensures the model prioritizes mastering the core logic of the recommendation, thereby enhancing overall accuracy and reliability. The comparison results are presented in Section 4.3.

3.2 Recommendation Refinement

Although the model captures the general distribution of IoT operations by training on extensive user history, its performance can be hindered by special scenarios that fall outside this distribution. We observe that such scenarios are typically caused by specific user preferences, which can be broadly classified into two categories:

- **Time-sensitive Operations**: Preferences for certain operations can be strongly time-dependent. As shown in Figure 5, the smart curtain exemplifies this pattern: recommendations for its operation are often accepted in the morning and at night but rejected during the day. Our analysis shows that such recommendations often fail when the recommended operations is inconsistent with a user's historical behavior within a specific time window.
- **Conflicting Operations**: Users are unlikely to perform contradictory actions in quick succession, leading to the rejection of conflicting recommendations. For example, suggesting to turn on the air conditioner moments after the user has manually switched it off would be considered intrusive and is typically rejected.

To enable our recommendation model to learn user preferences under these special scenarios, we refine it using Direct Preference Optimization (DPO) [19]. The core principle of DPO is to directly optimize the model on preference data. It uses pairs of preferred (positive) and dispreferred (negative) outputs to increase the likelihood of the positive samples while decreasing that of the negative ones. Compared to traditional reinforcement learning methods, the DPO sidesteps the need for an explicit reward model, resulting in a simpler and more stable training process.

DPO Sample Construction. For DPO training, we construct a dataset from a two-week sample of user data, comprising approximately 9,000 interactions. From this data, we build user preference

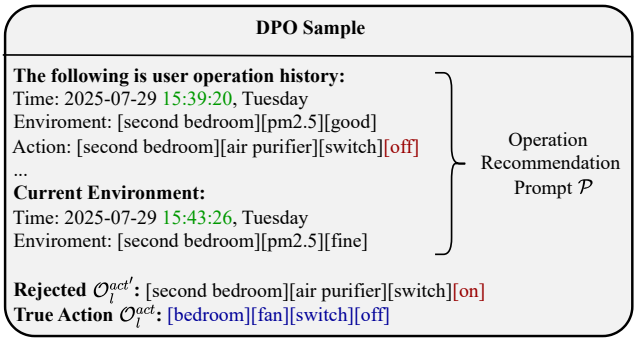


Figure 6: An example of a DPO sample. Note that the text in red represents conflicting operations, and text in green indicates similar timestamps.

pairs, each containing a positive sample S^{pos} and a negative sample S^{neg}. As shown in Figure 6, the positive sample S^{pos} consists of the operation recommendation prompt $\mathcal{P} = \{O_{<l}, O_l^{time}, O_l^{env}\}$ and the user's actual executed action O_l^{act}. The negative sample S^{neg} leverages the same prompt $\mathcal{P}$ but is paired with a dispreferred action $O_l^{act'}$. These pairs formally defined as:

$$S^{pos} = \{\mathcal{P}, O_l^{act}\}, \quad S^{neg} = \{\mathcal{P}, O_l^{act'}\} \tag{4}$$

Here, the dispreferred action is identified based on the two scenarios described above:

- **Time-sensitive Rejection.** An action is labeled as dispreferred ($O_l^{act'}$) if the user has no history of performing it within a ±1 hour window of the recommendation time, thus flagging it as temporally inappropriate.
- **Conflicting Rejection.** An action is considered dispreferred ($O_l^{act'}$) if it is the inverse of an action the user performed on the same device within the last 10 minutes. To avoid misclassifying common high-frequency interactions (e.g., toggling a light), this time window is shortened to two minutes for such operations.

DPO Training. We train the model by optimizing the log-probabilities of these sample pairs. The DPO objective is formalized as:

$$\mathcal{L}_{DPO} = -\log\sigma\left(\beta\log\frac{\mathrm{P}_\theta(O_l^{act}|\mathcal{P})}{\mathrm{P}_{ref}(O_l^{act}|\mathcal{P})} - \beta\log\frac{\mathrm{P}_\theta(O_l^{act'}|\mathcal{P})}{\mathrm{P}_{ref}(O_l^{act'}|\mathcal{P})}\right) \tag{5}$$

where P_θ denotes the probability computed by the current model, P_{ref} is the probability from the reference model (i.e., the fine-tuning stage), and β is a hyperparameter that controls the strength of the optimization. Compared to the reference model, this objective function encourages models to assign higher probabilities to preferred actions and lower probabilities to dispreferred actions.

3.3 Confidence-based Exposure Control

In operation recommendation tasks, users are sensitive to suboptimal suggestions, which severely degrades the user experience. This prompts us to ask a fundamental question: *How can we assess recommendation quality and control exposure accordingly?* To answer this question, we analyze rejected recommendations and find that users are significantly more likely to reject operations that the model predicts with low confidence. To improve the operation suggestion quality, we adopt a confidence-based exposure control mechanism. Specifically, we expose a suggestion only when the model's confidence in the prediction exceeds a certain threshold. This confidence is calculated as a weighted average of the model's output probabilities for the key attributes of the suggestion.

Confidence Score. We compute the confidence score $Conf(O_i)$ for each recommended operation. As an operation consists of a_i actions, the final score is an average across them. For each action $O_{i,j}^{act}$, we focus on the model's predicted probabilities for its key attributes, which are drawn from a vocabulary $\mathcal{V} = \{\text{device}, \text{field}, \text{value}\}$ (e.g., [AC][temperature][24] in the labeled action within Figure 4). To reflect their relative importance, the probability for each attribute type k, denoted as $\mathrm{P}(\mathrm{attr}_k(O_{i,j}^{act})|\mathcal{P})$, is scaled by a predefined weight, α_k. Then, the total confidence value is calculated by summing up these weighted probabilities of all attributes across all actions and normalizing the value to [0,1] by dividing it by the total number of actions. The process can be defined as:

$$Conf(O_i) = \frac{1}{a_i}\sum_{\mathrm{attr}_k\in\mathcal{V}}\sum_{j=1}^{a_i}\alpha_k \mathrm{P}(\mathrm{attr}_k(O_{i,j}^{act})|\mathcal{P}). \tag{6}$$

Adaptive Calculation Strategies. Given the complex scenarios in operation recommendations, relying on a static formula to calculate the confidence score is unreliable. Therefore, we introduce several adaptive strategies to refine this process: 1) The confidence threshold is dynamically adjusted according to task complexity. It is lowered by 10% for predictions with larger candidate pools or for numerical attributes (e.g., temperature) from a continuous range, accounting for the inherent difficulty of these tasks. 2) To enhance efficiency, we employ a cascading pruning mechanism. An entire recommendation is discarded if the confidence for a high-level attribute, such as the device, falls below its threshold. This hierarchical check prevents wasting computational resources on low-confidence candidates. 3) To create a more robust and reliable metric, the final confidence score is a fusion of the average confidence across all actions and the confidence of the first-generated token. This prevents scenarios where a high average score might mask low confidence in the foundational first token, which is often critical for the entire recommendation's validity.

4 Experimental Evaluations

4.1 Experiment Settings

Dataset. We extract environmental features, device lists, and historical operations from the Xiaomi Home app to construct a dataset for device operation recommendations. We sample 45,000 operation instances monthly, with this sampling process conducted continuously over three months for pre-training. Additionally, we manually annotate the original user logs to generate a high-quality test set consisting of 4,882 instances. Notably, the training and test sets are non-overlapping in time to prevent knowledge leakage into the model training phase. To enhance the diversity of the dataset, we divided it into multiple sub-datasets based on device numbers and categories. A total of 9 datasets are used for evaluation, comprising the 4 sub-datasets with the highest instance counts in each of two

Table 2: Accuracy results for operation recommendation. The best results are marked with bold. "1-5" denotes the number of devices per user in this sub-dataset ranges from 1 to 5. "Light" denotes all the operational devices in this sub-dataset are lights. The same applies to the rest sub-dataset. *Gain* is the average improvement of DevPiolt over the best-performing baseline.

Metric	Method	*Device Count*				*Device Category*					Whole	Gain (%)
		1-5	6-10	11-20	20+	Light	Switch	Camera	Warmer	AC		
EM-Acc	CGC	43.13	20.86	13.97	7.83	19.14	18.26	33.02	27.23	12.71	19.95	95.53
	DeepSeek	47.53	27.50	17.26	12.44	21.67	20.90	43.71	18.32	25.16	32.97	
	GPT-4o	48.03	29.99	18.33	15.57	20.02	21.47	47.03	19.51	25.90	33.43	
	DevPiolt	**68.66**	**51.69**	**36.48**	**27.62**	**38.56**	**44.72**	**77.67**	**53.00**	**35.90**	**44.33**	
LM-F1	CGC	49.48	22.70	16.10	7.29	13.90	15.05	33.85	32.13	17.39	17.72	58.49
	DeepSeek	56.38	38.19	38.10	33.23	41.66	37.46	45.27	26.42	27.37	44.68	
	GPT-4o	56.66	38.69	39.04	33.38	40.80	36.87	46.34	27.32	28.53	45.19	
	DevPiolt	**74.31**	**61.91**	**54.81**	**54.34**	**55.76**	**63.78**	**76.61**	**58.85**	**44.61**	**57.49**	
Rule	CGC	48.26	24.09	17.37	10.18	23.37	21.29	33.81	30.64	16.56	23.48	54.04
	DeepSeek	55.20	40.64	33.37	23.85	35.67	35.29	54.00	31.31	35.14	41.45	
	GPT-4o	55.61	39.14	34.26	24.04	34.96	33.94	55.19	33.26	37.29	42.32	
	DevPiolt	**74.44**	**59.32**	**46.23**	**38.38**	**55.76**	**63.78**	**79.72**	**60.3**	**45.18**	**53.03**	

above categories, along with the *whole* test set. Detailed dataset statistics are provided in the Appendix A.

Baselines. We compare DevPiolt with three baselines: 1) **CGC Model**. We adapt the customized gate control (CGC) model [23] to the operation recommendation by making detailed adjustments. Specifically, we use MLP layers to encode user and device features, and leverage a Transformer to capture operation dependencies in the historical sequence. The encoded features are then concatenated and fed together with the target operation into the CGC. The CGC dynamically controls the activation of different cross-features through learnable gating parameters, effectively filtering out the most relevant target operations with users. 2) **DeepSeek**. We concatenate the user's historical operation sequence, current environment, and list of operable devices, and then prompt the DeepSeek-V3 [14] model to directly output the operation recommendation. 3) **GPT-4o** [14]. Similar to the DeepSeek baseline, we use the GPT-4o model to provide operation suggestions. Note that current FRMs and LRMs are not designed for device operation recommendation, and thus cannot serve as appropriate baselines.

Implementation. We use Qwen2.5-14B [27] as the default model for training and inference. During the pre-training phase, we concatenate multiple historical operation sequences to fill the context window, setting the maximum length to 2048. The model is pre-trained for 1 epoch with a learning rate of 3e-5. In the fine-tuning phase, we set the LoRA rank to 16, LoRA alpha to 32, and LoRA dropout to 0.05. The LoRA [8] update matrices are applied to all projection layers. The model is fine-tuned for 2 epochs with a learning rate of 4e-4. Experiments are conducted on a server equipped with 8 NVIDIA H20 GPUs, each with 96GB of memory.

Operation Evaluation Criteria. A reasonable evaluation criterion is essential for the qualitative analysis of operation recommendations. We denote users' actual actions as ground truth labels $\mathcal{Y}$. The predicted results are represented as y. Next, we provide a detailed introduction to our three proposed metrics:

- **Exact Match Accuracy**. An exact matching indicates that the four items of the action quadruplets O^{act} in the prediction and the actual operations are identical.
- **Loose Match F1 Score**. An action may be deemed correct even if it does not exactly match the label, as users can employ various expressions to achieve the same operation intent. A loose match is defined as either the difference between the predicted and actual action value is below 20%, or the predicted and actual actions being functionally equivalent. For example, the predicted action "*Set bedroom AC to comfort mode*" is considered equivalent to the actual one "*Turn on the bedroom AC*". We provide detailed explanations in Appendix B. We calculate the loose match F1 score based on this loose matching criterion:

$$\text{LM-F1} = 2 \cdot \frac{lm\_precision \cdot lm\_recall}{lm\_precision + lm\_recall}.$$

- **Rule Score**. The rule score assesses the accuracy and comprehensiveness of the model's recommendation results in multi-device, multi-operation scenarios. We consider the single action quadruplet setting, which includes four metrics: *identical, intersecting, containing*, and *opposite*:

$$\varsigma = \begin{cases} 1, & \mathcal{Y} = y \\ 1/|E|, & E = \mathcal{Y} \cap y \\ 0.9, & \mathcal{Y} \subset y \\ -0.1, & y = -\mathcal{Y} \end{cases},$$

where $-\mathcal{Y}$ indicates the opposite operation. The total rule score is the sum of the scores for each action quadruple instance, divided by the number of predicted quadruples N_{quad}: Rule = $\left(\sum_{i=1}^{N_{quad}} \varsigma_i\right) / N_{quad}$. This metric is reasonable as it considers both

Table 3: Ablation study of fine-tuning, pre-training and recommendation refinement for DevPiolt. *Base* is the naive Qwen2.5-14B model.

Method	Whole		
	EM-Acc	LM-F1	Rule
Base	32.04	44.05	41.92
+ Fine-tuning	42.36	56.12	51.23
+ Pre-training	43.57	56.99	52.74
+ Recommendation refinement	44.33	57.49	53.03

the accuracy of individual operations and the overall accuracy across multiple operations.

4.2 Main Results

We compare DevPiolt with CGC model, DeepSeek and GPT-4o to evaluate the accuracy of recommendation results. Table 2 reports the experimental results on 9 sub-datasets and the *whole* dataset. We have following three observations: 1) DevPiolt outperforms the baselines on all three accuracy metrics across all datasets, demonstrating a maximum average of 95.53%, 58.49%, and 54.04% over the best-performing baseline. DevPiolt surpasses CGC model by leveraging the context understanding capability of the LLM to generate logically coherent operation combinations. Additionally, DevPiolt outperforms GPT-4o because the LLM learns specific operation patterns and user preferences through pre-training, fine-tuning, and DPO. 2) As the number of devices increases, the accuracy of all methods decreases, due to the increased complexity in making accurate predictions. 3) Our method achieves the highest accuracy on the Camera dataset but relatively lower accuracy on the Light dataset. This is because a camera has only two possible operations (i.e., on and off), while lights may have more complex operations, including lighting modes and brightness control.

4.3 Micro Experiments

Ablation Study. To study the impact of each component in DevPiolt, we gradually activate *fine-tuning*, *pre-training*, and *recommendation refinement* on the base LLM (i.e., Qwen2.5-14B) and evaluate them on the whole dataset. As shown in Table 3, each module significantly improves the model's accuracy, demonstrating the effectiveness of our techniques. Fine-tuning allows the LLM to learn fine-grained IoT device operation patterns, pre-training injects domain-specific knowledge by leveraging a large amount of historical operation sequences, and recommendation refinement ensures the LLM learns specific user preferences.

Evaluation of Exposure Control. To investigate the role of exposure control, we gradually adjust the confidence level from 0 to 1 and observe the changes in exposure rate and acceptance rate. Acceptance rate refers to the proportion of operation suggestions accepted by users out of the total number of operation suggestions. The exposure rate indicates the proportion of suggestions shown to users out of the total number of generated suggestions. As shown in Figure 7 (a), as confidence increases, the exposure rate initially decreases slowly and then more rapidly, while the acceptance rate

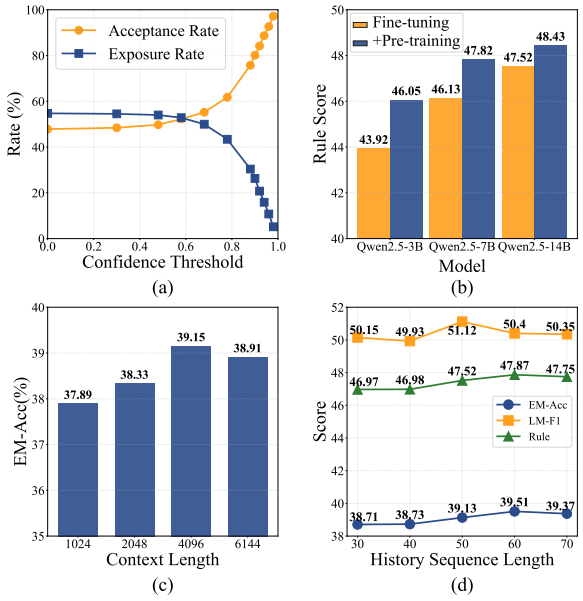


Figure 7: The comparison results of model accuracy with different parameters.

shows the opposite trend. Higher confidence indicates a more conservative recommendation strategy, thus reducing the likelihood of presenting prediction results to users. However, higher confidence also means the model is more certain in its predictions, leading to a higher acceptance rate. To balance the trade-off between exposure rate and acceptance rate, we set the confidence threshold to 0.7.

Action-First vs. Text-First. We evaluate the accuracy impact of two operation generation strategies on the whole test set: action-first and text-first. The comparison results are provided in Table 4. We observe that the action-first strategy outperforms the text-first strategy on all three metrics. This is because action quadruples provide a more fine-grained representation of operations compared to textual descriptions. The detailed action quadruple can help model to improve device recall. Conversely, if rough operation descriptions are generated first, the action quadruples may converge to the description due to the LLM's next token generation mechanism, thereby harming accuracy.

Impact of Different Model Sizes. We evaluate the impact of different Qwen2.5 model sizes on accuracy across the whole test set. "Fine-tuning" refers to the fine-tuning to the original Qwen model, whereas "+Pre-training" indicates fine-tuning on a pre-trained Qwen model. As shown in Figure 7(b), we observe that accuracy increases with the model size, as larger models exhibit stronger expressive power and a better ability to model long historical operation sequences. Additionally, incorporating a pre-training stage into smaller models can achieve comparable accuracy to that of larger models. This suggests that pre-training learns domain-specific knowledge, and thus improving model accuracy.

Impact of Context Length and Historical Sequence Length. We study the impact of context length during training and the length of historical sequences during testing. The comparison results are presented in Figure 7(c), (d). We find that very short context

Table 4: The comparison of model accuracy between action-first and text-first strategies. "Action-first" indicates that actions are generated before text during fine-tuning, whereas "Text-first" follows the reverse order.

Method	Whole		
	EM-Acc	LM-F1	Rule
Action-first	44.33	57.49	53.03
Text-first	42.43	56.66	51.75

lengths limit the operational knowledge the model can learn, while excessively long context lengths cause the model to lose focus. A similar phenomenon is observed with the length of historical sequences. Therefore, a moderate context length and historical sequence length are crucial for optimal model accuracy.

Failure Analysis. We randomly sample 50 user-disliked operation suggestions over a week to conduct a failure analysis. We categorize the failure causes into six types:

- *Incomplete operations*. The generated suggestion fails to cover all user's actual operations.
- *Unclear description*. The generated operation descriptions are difficult for users to understand (e.g., a suggestion to "turn off the light" without specifying the room).
- *Imperceptible environment*. Inadequate environmental data leads to inaccurate operation suggestions (e.g., suggesting to turn off the lights at night when someone is in the room).
- *Inappropriate timing*. The suggested operations are often ill-timed (e.g., turning on all lights during the daytime).
- *Lack of historical operations*. The model's recommendations are inaccurate due to the limited historical operations. (Direct user control of devices results in missing operation records.)
- *Unreasonable recommendations*. The operation suggestions can not align with the user's needs (e.g., turning off the AC when the room temperature is high).

Figure 8 shows that over 45% of the failed cases are due to unreasonable recommendations and lack of historical operations. This highlights the importance of enriching historical operational data and using it to better understand user intentions, which is crucial for the accuracy of the recommendation model. The fewest failures are due to inappropriate timing, suggesting that our recommendation refinement technique can effectively mitigate such issues. The remaining three categories have roughly the same number of cases, indicating that the model still faces significant challenges in complex environments with multiple devices and operations.

5 Related Work

Recommendation Models. Existing recommendation models can be categorized into two types: feature-based recommendation models (FRMs) [5, 10, 17, 25, 34] and LLM-based recommendation models (LRMs) [7, 9, 12, 15, 35]. Specifically, for FRMs, DIN [36] captures users' interest features using an attention mechanism and deep neural networks, and employs MLP to model high-order interactions between features. DirectPred [24] distinguishes between short-term

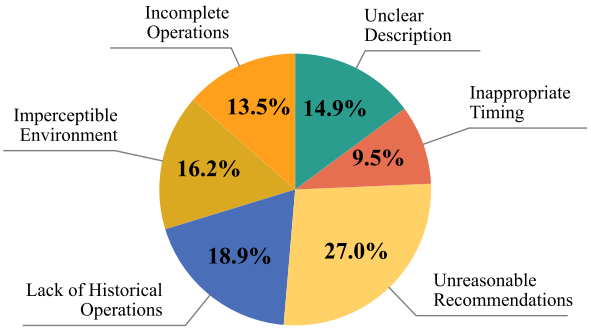


Figure 8: Case study for user-disliked operation suggestions.

behavioral preferences and long-term interest features through contrastive learning. HRNN [16] models the temporal dependencies of users' real-time behavior and contextual features using causal and dilated convolutions in a temporal convolutional network. For LRMs, CALRec [13] dynamically aligns users' behavior sequences with target item features in the latent space through fine-tuning and multi-task learning based on an LLM. Recformer [11] extracts item text features using a pre-trained language model and models user action sequences with a Transformer, achieving the mapping between user intent and item semantics using only textual modality. LRURec [33] achieves fast training and dynamic interest modeling with a simplified linear recurrent structure and parameter sharing, thus enhancing the sequential prediction capabilities while maintaining computational efficiency. However, none of these methods can be applied to operational recommendations, as they are unable to capture complex operational logic and patterns.

Management for IoT Devices. IoT device management [1, 2, 6, 18, 31] encompasses the configuration and scheduling of IoT devices to ensure their efficient and reliable operation. AutoIOT [20] achieves automated AIoT application management by using LLM-driven natural language programming, which generates executable configuration instructions for devices through semantic parsing. qToggle [21] designs a low-power communication protocol and an embedded sensor network to enable interconnectivity and autonomous decision-making among IoT devices. Cristina et al. [22] employs a cost-effective Wi-Fi module as the core controller and integrates physical switches to unify local and remote control in smart homes. However, these methods are limited to basic device control and can not offer fine-grained operation suggestions based on the current environment.

6 Conclusion

We introduce DevPiolt, an LLM-based operation recommendation for IoT devices. Specifically, we pre-train and fine-tune LLMs to understand the basic logic of device operations. We then use direct preference optimization to tailor the suggestions to individual user preferences. Finally, we implement a confidence-based exposure control technique to prevent the presentation of suboptimal suggestions. Extensive experiments demonstrate that our approach outperforms all baselines, paving the way for community research in operation recommendations.

References

[1] Pragya Bajpayi, Smita Sharma, and Madhu Sharma Gaur. 2024. AI driven IoT healthcare devices security vulnerability management. In *2024 2nd International Conference on Disruptive Technologies*. IEEE, 366–373.

[2] Anders Eivind Braten, Frank Alexander Kraemer, and David Palma. 2020. Autonomous IoT device management systems: structured review and generalized cognitive model. *IEEE Internet of Things Journal* 8, 6 (2020), 4275–4290.

[3] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. 2024. Sharegpt4v: Improving large multi-modal models with better captions. In *European Conference on Computer Vision*. Springer, 370–387.

[4] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[6] Ehtisham Ul Haque, Adil Shah, Jawaid Iqbal, Syed Sajid Ullah, Roobaea Alroobaea, and Saddam Hussain. 2024. A scalable blockchain based framework for efficient IoT data management using lightweight consensus. *Scientific Reports* 14, 1 (2024), 7841.

[7] Jesse Harte, Wouter Zorgdrager, Panos Louridas, Asterios Katsifodimos, Dietmar Jannach, and Marios Fragkoulis. 2023. Leveraging large language models for sequential recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 1096–1102.

[8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.

[9] Jun Hu, Wenwen Xia, Xiaolu Zhang, Chilin Fu, Weichang Wu, Zhaoxin Huan, Ang Li, Zuoli Tang, and Jun Zhou. 2024. Enhancing sequential recommendation via llm-based semantic embedding learning. In *Companion Proceedings of the ACM Web Conference 2024*. 103–111.

[10] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining*. IEEE, 197–206.

[11] Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. 2023. Text is all you need: Learning language representations for sequential recommendation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1258–1267.

[12] Lei Li, Yongfeng Zhang, and Li Chen. 2023. Prompt distillation for efficient llm-based recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 1348–1357.

[13] Yaoyiran Li, Xiang Zhai, Moustafa Alzantot, Keyi Yu, Ivan Vulić, Anna Korhonen, and Mohamed Hammad. 2024. Calrec: Contrastive alignment of generative llms for sequential recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems*. 422–432.

[14] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[15] Qidong Liu, Xian Wu, Yejing Wang, Zijian Zhang, Feng Tian, Yefeng Zheng, and Xiangyu Zhao. 2024. Llm-esr: Large language models enhancement for long-tailed sequential recommendation. *Advances in Neural Information Processing Systems* 37 (2024), 26701–26727.

[16] Yifei Ma, Balakrishnan Narayanaswamy, Haibin Lin, and Hao Ding. 2020. Temporal-contextual recommendation in real-time. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2291–2299.

[17] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[18] Thinagaran Perumal, Soumya Kanti Datta, and Christian Bonnet. 2015. IoT device management framework for smart home scenarios. In *2015 IEEE 4th Global Conference on Consumer Electronics*. IEEE, 54–55.

[19] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems* 36 (2023), 53728–53741.

[20] Leming Shen, Qiang Yang, Yuanqing Zheng, and Mo Li. 2025. Autoiot: Llm-driven automated natural language programming for aiot applications. *arXiv preprint arXiv:2503.05346* (2025).

[21] K Srinivasrao, S Ramana Kumar Joga, Athava Praveen Kumar, Orupula Puja Hemanth, Kanumareddy Leela Varaha Lavanya, and Dasari Yaswanth. 2024. User Experience design for IoT-Driven Smart Home Automation interfaces. In *2024 3rd International Conference on Artificial Intelligence For Internet of Things*. IEEE, 1–6.

[22] Cristina Stolojescu-Crisan, Calin Crisan, and Bogdan-Petru Butunoi. 2021. An IoT-based smart home automation system. *Sensors* 21, 11 (2021), 3784.

[23] Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. 2020. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 269–278.

[24] Yuandong Tian, Xinlei Chen, and Surya Ganguli. 2021. Understanding self-supervised learning dynamics without contrastive pairs. In *International Conference on Machine Learning*. PMLR, 10268–10278.

[25] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD*. 1–7.

[26] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. 2022. A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4425–4445.

[27] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[28] Shenghao Yang, Weizhi Ma, Peijie Sun, Qingyao Ai, Yiqun Liu, Mingchen Cai, and Min Zhang. 2024. Sequential recommendation with latent relations based on large language model. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 335–344.

[29] Yuyang Ye, Zhi Zheng, Yishan Shen, Tianshu Wang, Hengruo Zhang, Peijun Zhu, Runlong Yu, Kai Zhang, and Hui Xiong. 2025. Harnessing multimodal large language models for multimodal sequential recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 13069–13077.

[30] Mingjia Yin, Hao Wang, Wei Guo, Yong Liu, Suojuan Zhang, Sirui Zhao, Defu Lian, and Enhong Chen. 2024. Dataset regeneration for sequential recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3954–3965.

[31] Liang Yu, Weiwei Xie, Di Xie, Yulong Zou, Dengyin Zhang, Zhixin Sun, Linghua Zhang, Yue Zhang, and Tao Jiang. 2019. Deep reinforcement learning for smart home energy management. *IEEE Internet of Things Journal* 7, 4 (2019), 2751–2762.

[32] Sha Yuan, Hanyu Zhao, Zhengxiao Du, Ming Ding, Xiao Liu, Yukuo Cen, Xu Zou, Zhilin Yang, and Jie Tang. 2021. Wudaocorpora: A super large-scale chinese corpora for pre-training language models. *AI Open* 2 (2021), 65–68.

[33] Zhenrui Yue, Yueqi Wang, Zhankui He, Huimin Zeng, Julian McAuley, and Dong Wang. 2024. Linear recurrent units for sequential recommendation. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 930–938.

[34] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, et al. 2024. Actions speak louder than words: trillion-parameter sequential transducers for generative recommendations. In *Proceedings of the 41st International Conference on Machine Learning*. 58484–58509.

[35] Junjie Zhang, Ruobing Xie, Yupeng Hou, Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2025. Recommendation as instruction following: A large language model empowered recommendation approach. *ACM Transactions on Information Systems* 43, 5 (2025), 1–37.

[36] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1059–1068.

A Dataset

We provide the statistical data of the test set in Table 5. "DC 1-5" indicates that the number of user devices in this sub-dataset is between 1 and 5, and "Light" indicates that all the objects being operated on in this sub-dataset are lights. The remaining datasets follow a similar pattern. *Whole* represents the entire dataset. The numbers in the table indicate the number of samples included in each sub-dataset.

B Details for Loose Match Criterion

The loose matching criterion comprises two necessary conditions. First, the difference between the predicted and actual continuous parameter values must be within 20%. Specifically, if the operation value is numerical (e.g., temperature setting or device power level),

Table 5: Test Set Statistics. #Samples denote the number of samples.

Dataset	DC 1-5	DC 6-10	DC 11-20	DC 20+	Light	Switch	Camera	Warmer	AC	Whole
#Samples	970	1,037	1,650	1,225	1,211	1,599	318	200	390	4,882

we calculate the relative error using the formula:

$$\epsilon = \frac{|y_p - y_a|}{y_a},$$

where y_p is the predicted value and y_a is the actual value. The condition for loose matching is satisfied when $\epsilon \leq 0.2$. Second, the operational intent must be functionally equivalent. For instance, if the actual operation is "Turn on bedroom AC", a model prediction of "Set bedroom AC to 26" is also considered correct. Similarly, "Activate living room light" and "Adjust living room ceiling light to 50% brightness" are both deemed to achieve equivalent lighting effects.