ChatIoT: Zero-code Generation of Trigger-action Based IoT Programs

YI GAO, College of Computer Science and Technology, Zhejiang University, China
KAIJIE XIAO, College of Computer Science and Technology, Zhejiang University, China
FU LI, College of Computer Science and Technology, Zhejiang University, China
WEIFENG XU, School of Software Technology, Zhejiang University, China
JIAMING HUANG, College of Computer Science and Technology, Zhejiang University, China
WEI DONG*, College of Computer Science and Technology, Zhejiang University, China

Trigger-Action Program (TAP) is a simple but powerful format to realize intelligent IoT applications, especially in home automation scenarios. Existing trace-driven approaches and in-situ programming approaches depend on either customized interaction commands or well-labeled datasets, resulting in limited applicable scenarios. In this paper, we propose ChatIoT, a zero-code TAP generation system based on large language models (LLMs). With a novel context-aware compressive prompting scheme, ChatIoT is able to automatically generate TAPs from user requests in a token-efficient manner and deploy them to the TAP runtime. Further, for those TAP requests including unknown sensing abilities, ChatIoT can also generate new AI models with knowledge distillation by multimodal LLMs, with a novel model customization method based on deep reinforcement learning. We implemented ChatIoT and evaluated its performance extensively. Results show that ChatIoT can reduce token consumption by 26.1-84.9% and improve TAP generation accuracy by 4.2-65.5% compared to state-of-the-art approaches in multiple settings. We also conducted a real user study, and ChatIoT can achieve 91.57% TAP generation accuracy.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing**.

Additional Key Words and Phrases: Zero-code TAP generation, LLMs, Home automation, Internet of Things

ACM Reference Format:
Yi Gao, Kaijie Xiao, Fu Li, Weifeng Xu, Jiaming Huang, and Wei Dong. 2024. ChatIoT: Zero-code Generation of Trigger-action Based IoT Programs. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 3, Article 103 (September 2024), 29 pages. https://doi.org/10.1145/3678585

1 Introduction

The Internet of Things (IoT) has revolutionized the way we interact with the physical world. However, generating IoT programs is still a major challenge, due to the complex application requirements, the heterogeneous hardware and software, etc. Trigger-action programs (TAPs) [40, 60–62] are a special category of IoT programs, which is a set of rules with " IF *triggers*, THEN *actions*" syntax. For example, a typical TAP of the Home Assistant runtime [8] is "trigger: (platform: time, at '18:00:00') action: (service: light.turn_on, target: (entity_id:

*Wei Dong is the corresponding author.

Authors' Contact Information: Yi Gao, College of Computer Science and Technology, Zhejiang University, China, gaoyi@zju.edu.cn; Kaijie Xiao, College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, China, xiaokj@zju.edu.cn; Fu Li, College of Computer Science and Technology, Zhejiang University, China, lileaf@zju.edu.cn; Weifeng Xu, School of Software Technology, Zhejiang University, China, xuwf@zju.edu.cn; Jiaming Huang, College of Computer Science and Technology, Zhejiang University, China, huangjm@zju.edu.cn; Wei Dong, College of Computer Science and Technology, Zhejiang University, China, dongw@zju.edu.cn.

ACM 2474-9567/2024/9-ART103
https://doi.org/10.1145/3678585

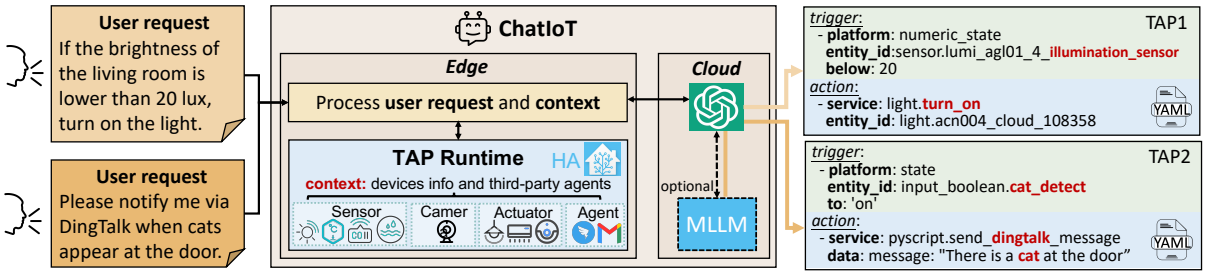


Fig. 1. ChatIoT usage. ChatIoT in edge will process the user request and home informantion and transport them to cloud. At the cloud, ChatIoT will interacts with the LLM and the multimodal LLM (if necessary) for TAP generation. Two examples and their corresponding workflows for generating corresponding TAPs in ChatIoT are shown in the figure where different requested workflows are marked with different colors.

light.lumi_acn004_b385_light))". With this simple syntax, TAPs can be surprisingly useful to express IoT applications [15, 42]. For example, the smart home is a typical IoT application scenario, which is already widely implemented by TAPs [9].

Due to the simplicity of TAPs, existing approaches have employed various techniques to automatically generate TAPs. To speed up the TAP generation process, in-situ interactive programming methods [40, 43] have been proposed recently. These methods enable users to generate TAPs with direct interactions with the nearby objects of interest. There are also trace-driven approaches [39, 60, 61] proposed to automatically generate TAPs based on traces of sensor readings and manual actuation of devices. However, these approaches depend on either customized interaction commands or well-labeled datasets, resulting in limited applicable scenarios.

With the recent advance of multimodal LLMs, it becomes possible to make the TAP generation process more intelligent. In this paper, we propose ChatIoT, a zero-code TAP generation system with multimodal LLMs for IoT applications, especially in smart home automation scenarios. Different from existing methods, ChatIoT enables a user to generate TAPs with natural language only. Figure 1 shows how ChatIoT works with two exemplary user requests. At the edge, ChatIoT will deploy the generated TAPs on the TAP runtime (e.g., Home Assistant [8]) which can interact with the devices or third-party agents (e.g., DingTalk, an instant messaging App) according to the TAPs. ChatIoT in edge will process the user input and context including all devices information and third-library agents and transport them to the cloud. At the cloud, ChatIoT will interact with the LLM and the multimodal LLM for TAP generation. The use of a multimodal LLM becomes essential when the capabilities requested by the user are not readily available. For instance, in response to a user request such as "Please notify me via DingTalk when there is a cat at the home entrance", if the service capable of detecting a cat is not available, it becomes necessary to generate this service automatically.

The idea of using LLMs to improve the intelligence of smart home is not new. The widespread adoption of large language models (LLMs), such as ChatGPT [4], has demonstrated significant potential for the integration of LLMs into daily life, as indicated by various studies [27, 28, 36, 41, 52, 55, 57, 64]. Wang et al. [55] reveal that LLM has the capability to assist or replace human efforts in producing vast amounts of high-quality, human-like content more rapidly and cost-effectively. In fact, after the release of ChatGPT in 2022, some of the earlier applications immediately integrated ChatGPT in various voice assistants for smart home, e.g., Amazon Alexa [1], Apple Siri [2], and Google Assistant [6]. These approaches can only enable intelligent interactions of devices using natural languages, while the zero-code TAP generation problem remains unsolved. There are two major challenges to enable zero-code TAP generation with LLMs.

First, to generate TAP from a request in natural language, the LLM needs to extract all the *triggers* and the *actions* in the format of the runtime (e.g., Home Assitant). This requires the LLM to have two kinds of contexts, device information of the home and the API information of the runtime. Since LLM cannot keep context across multiple conversations, it is challenging to token-efficiently prompt the LLM together with the context. In addition, there exist hallucination problems when LLM inference, which means redundant information will affect the accuracy of the final generated TAP. We propose a context-aware prompting method for automatic TAP generation in ChatIoT. To reduce the token cost during prompting, ChatIoT adopts a compressive prompting method, which removes the information of irrelevant devices and compresses the information of the relevant ones. Further, ChatIoT employs a four-step prompting scheme which divides the process of generating instructions into four steps: Preprocessing, Service Creation, TAP Generation and TAP Evaluation. This four-step prompting design combines the reasoning and acting of the LLM, enabling it to obtain extra information from the user in a context-aware manner.

Second, although a camera could be a powerful sensor (i.e., a *trigger* in a TAP), it is challenging to generate AI models for various user requirements, especially when these models should be deployed on edge devices with limited resources. For example, when the user needs to create a TAP with a request "turn on the light at the door when there is visitor at the door", a "visitor sensor" based on the camera at the door should be generated before the TAP generation. Therefore, how to automatically generate and manage multiple such AI models on edge devices with limited resources becomes another major challenge. In ChatIoT, we propose an on-demand model customization method based on deep reinforcement learning (DRL). When ChatIoT receives a new request of TAP generation in which the *trigger* should be a camera with an AI model, the proposed model customization method will first employ knowledge distillation to generate the AI model with the assistance of a multimodal LLM at the cloud. Then, due to the limited resource of the edge device, ChatIoT uses deep reinforcement learning to determine how to deploy the newly generated model, i.e., one of the following three actions, 1) direct deployment in case of sufficient remaining memory, 2) replacing one existing model, and 3) selecting two models from the existing models and the newly generated model and performing model merging.

We implemented ChatIoT and evaluated its performance extensively, mainly including the accuracy and token consumption of TAP generations. Results show that ChatIoT with the context-aware compressive prompting method can significantly reduce token consumption and improve TAP generation accuracy than existing approaches. For requests not related to camera operations, ChatIoT demonstrated a remarkable capability to enhance operational efficiency and accuracy. Compared with state-of-the-art approaches[59], ChatIoT can achieve a significant reduction in token consumption, ranging from 26.1% to 78.8% and improve TAP generation accuracy by 7.5% to 65.5% across different synthetic homes. For requests specifically related to camera operations, the performance of ChatIoT is even more impressive. The system is able to reduce token consumption by a staggering 52.5% to 84.9%, indicating an even greater efficiency in processing these particular types of requests. Additionally, ChatIoT can improve TAP generation accuracy by 4.2% to 35.3% for camera-related requests. In addition, we have evaluated the model customization method in ChatIoT which achieves a higher average model accuracy of 81.4%-95.2%, which reaches 1.12x-1.33x compared with two baselines. We further conducted a case study using a DIY smart home with multiple sensors, cameras, and actuators. We employ an end-to-end example to demonstrate the operational effectiveness of the entire system and its individual components, followed by an evaluation based on real users. Results confirm that ChatIoT can attain high TAP generation accuracy across a wide range of user requests which will significantly reduce users' learning costs.

We summarize the contributions of ChatIoT as follows:

- We propose a novel context-aware compressive prompting method to automatically generate TAPs from user requests in a token efficient manner.

- We propose a DRL-based model customization method to generate and deploy AI models on the edge device, with the assistance of multimodal LLMs.
- We implemented and evaluated ChatIoT extensively with synthetic homes equipped with different number of sensors and actuators. Results show that ChatIoT can significantly ease the TAP generation process in a token-efficient manner.

The rest of the paper is organized as follows. Section 2 presents the related work of this work. Section 3 describes the design methodologies of the context-aware compressive prompting in detail. Section 4 discusses the DRL-based on-demand model customization. Section 5 shows the evaluation results of ChatIoT under various conditions, compared with state-of-the-art TAP generation approaches. In Section 6, we conclude this paper and present possible future work.

2 Related Work

There are mainly three kinds of existing approaches which are closely related to ChatIoT, prompting schemes, TAP generating techniques and edge model deployment schemes. We will briefly describe these three kinds of related work and summarize their major differences from ChatIoT.

2.1 Prompting Schemes for LLMs

Various studies demonstrate the powerful ability of LLM to assist users in their daily lives [36, 55, 64]. LLMs today, such as GPT-4 [7], Claude [5], and LLaMA [50], are tuned to follow instructions and are trained on large amounts of data. Therefore, they are capable of performing some tasks in a *zero-shot* manner. Within the instructions, a user can inform the LLMs of the input, the desired format of the output, and the method to generate TAPs. In practice, however, these methods still face significant performance degradation in more complex tasks within the zero-shot setting [14, 65].

Generating a chain of thought—a series of intermediate reasoning steps—significantly improves the ability of large language models to perform complex reasoning. Chain of thought [56] (CoT) reasoning facilitates the decomposition of complex, multi-step problems into constituent cognitive processes, enabling dynamic allocation of computational resources to more intricate reasoning tasks. Additionally, there have also been several works [13, 54, 58, 63] used to improve the performance of CoT. ToT [58] allows LLMs to perform deliberate decision making by considering multiple different reasoning paths like a tree. GoT [13] allows LLM to be freed from the combination of thoughts and then obtain the answer. However, when the inference requires additional information from outside the LLM, CoT cannot achieve satisfactory performance [54, 63], especially with complex user requests.

ReAct [59] prompts LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two. This tight synergy between "acting" and "reasoning" allows humans to learn new tasks quickly and perform robust decision making or reasoning, even under previously unseen circumstances or facing information uncertainties. In ChatIoT, we build the prompting module based on the ReAct approach with a number of improvements and customization for the automatic TAP generation scenario, e.g., compressive and context-aware prompting.

2.2 Home Automation Approaches

There are mainly two directions of recent attempts to ease home automation, in-situ interactive programming and trace-driven TAP generation. Table 1 shows the difference between existing home automation approaches and ChatIoT. ChatIoT implements a zero-code TAP generation technology which provides a simple interaction mechanism and model customization capability, enabling users to customize TAP rules.

Table 1. Difference between existing home automation approaches and ChatIoT

Method	User learning cost	User interaction	Model customization
In-situ programming [40, 43, 48]	High	✓	×
Trace-driven programming [19, 39, 60]	Low	×	×
ChatIoT (Ours)	Zero	✓	✓

ISP [40] is an in-situ programming paradigm that allows users to program smart home automation with in-situ contextual information and an interaction pattern. It significantly improves the user experience of generating home automation rules compared with traditional TAP generation methods based on GUI or voice assistants, e.g., Amazon Alexa [1], and Apple HomeKit [3]. In the literature, there are also similar in-situ programming approaches that use AR [43] or with different application scenarios [48]. Although in-situ programming is a promising technique for home automation, there are still practical challenges that may not be easily addressed in the near future, e.g., high user learning costs, privacy concerns, and high human sensing requirements. By utilizing the recent advances of LLMs, ChatIoT is purely based on natural language which requires zero learning cost for users. In addition, ChatIoT could be a natural extension of current deployed voice assistants, without introducing extra privacy concerns. In general, we believe that the in-situ programming paradigm and LLM-based TAP generation approaches like ChatIoT could be combined together and push the home automation intelligence to the next level.

Another important recent advance of TAP generation is trace-driven approaches [19, 39, 60, 60, 61]. For example, Trace2TAP [61] is able to automatically synthesize TAP rules from traces (time-stamped logs of sensor readings and manual actuation of devices). It can synthesize generalizable rules more comprehensively and fully handles nuances like out-of-order events, compared with prior approaches. RecipeGen [60] is another example, which obtains a model from TAP traces to map user requests to possible triggers or actions. However, due to the limited capabilities of the model, it cannot achieve fully automated TAP generation for deployment. These trace-driven approaches are actually orthogonal to ChatIoT, since they mainly focus on the TAP recommendation from historical traces, while ChatIoT focuses on automatically generating TAPs from user requests.

Furthermore, current solutions do not provide personalized model deployment, particularly in scenarios that necessitate camera detection events as trigger conditions. ChatIoT, on the other hand, is capable of utilizing multimodal large language models to facilitate customized model deployment.

2.3 Edge Model Deployment Scheme

In order to deliver highly-accurate query responses in real time, models that handle user needs have steadily migrated to the edge. As user needs increase, the number of models solving specific tasks grows which will cause an ever-worsening resource picture for edge model deployment.

Traditionally, techniques such as model quantization, model pruning, and knowledge distillation have been extensively employed to compress models, thereby reducing their computational demands and memory footprint [16, 18, 21]. Model quantization [25, 44, 66] reduces the precision of the numerical parameters, model pruning [31, 35, 38] eliminates redundant or less significant parameters, and knowledge distillation[17, 26, 53] transfers knowledge from a larger, more complex model to a smaller, more efficient one. To meet the diverse needs of users, ChatIoT uses MLLM as a teacher model and employs knowledge distillation to derive smaller models for edge deployment.

Recently, an innovative approach through model merging has been proposed, specifically addressing scenarios where multiple models are to be deployed on edge devices. This technique involves the amalgamation of several models into a single, more compact model that retains the collective functionality and knowledge of the individual

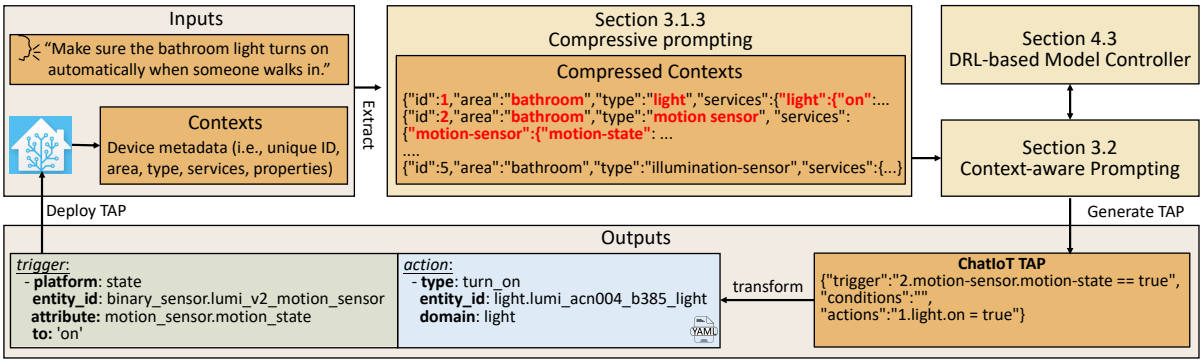


Fig. 2. Workflow of TAP generation. ChatIoT at the edge receives requests and contexts as input and then filter irrelevant information through compressive prompting. The compressed information, combined with the user requests, is employed for context-aware prompting to generate the corresponding ChatIoT format TAP, which is then transformed and deployed to Home Assistant. Furthermore, DRL-based Model Controller will be invoked when it needs to generate an AI model.

models. Model merging[12, 30, 34, 46, 49] is particularly suitable for situations where the number of models to be deployed is substantial, as it not only reduces the overall storage space required but also streamlines the inference process, potentially leading to improvements in both efficiency and speed. Git Re-Basin[12] generalizes further to models trained on the same data but with different initializations, though with a significant accuracy drop. Ziplt[49] uses a multi-head model for merging features within each model. Gemel[46] shares architecturally identical layers across the models for real-time video analytics at the edge.

The aforementioned studies have demonstrated the efficacy of alleviating resource limitations on the edge of networks through the implementation of model merging techniques. However, to attain optimal operational performance, it is imperative to devise an appropriate decision-making strategy. ChatIoT employs a model merging mindset, where the DRL decides on the appropriate model deployment strategy (including performing model merging) in order to achieve the deployment of the user's desired model on the edge side.

3 Context-aware Compressive Prompting

ChatIoT aims to enable users to generate TAPs utilizing natural language interfaces, rather than requiring traditional coding or configurations. In this section, we will present how ChatIoT enables users to generate TAPs in a token-efficient manner. We will first describe the overall workflow of TAP generation, including the inputs and outputs of ChatIoT. Then we will dive into the details of the context-aware prompting.

3.1 Overall Workflow of TAP Generation with Compressive Prompting

Figure 2 shows the overall workflow of TAP generation, mainly including the input, the output, the context-aware prompting module of ChatIoT, and the DRL-based model customization module which will be described in the next section.

3.1.1 Inputs. The input to ChatIoT includes two parts, a TAP generation request using natural language from the user, and additional information (i.e., the context) about the sensors and actuators in the home.

The prompting context is mainly device metadata, including unique *ID*, *area* (i.e. room where the device is located), *type*, *services*, and *properties* as shown on the left of Figure 2. Concretely, the *property* delineates the current state of a device, which is manipulated via defined access permissions, enabling interactive operations such as "read", "write", and "report". For instance, the binary state of a light (i.e. light on or light off) is encapsulated within

an "on" property, whereby writing a Boolean value (True/False) toggles the light's state. The *service* characterizes the suite of capabilities provided by the device, detailed through various "properties" that represent the specific features within the service. An example is a "light" service, which encompasses services like switch toggling, brightness adjustment, and chromatic temperature modulation, corresponding to three distinct properties: "on", "brightness", and "color-temperature".

3.1.2 Outputs. ChatIoT will generate the target TAP that aligns with the request of the user. Within some smart home frameworks, the utilization of TAPs in the format of "IF <triggers> WHILE <conditions> THEN <actions>" is widespread. This format stipulates that upon the occurrence of triggers and the satisfaction of conditions, a sequence of actions is executed. To generate TAP in the above format, ChatIoT needs to generate propositions for triggers, conditions, and actions based on user descriptions. For instance, when the user expresses "IF the motion sensor is activated WHILE the illumination is below 20 lux THEN turn on the light", the trigger-proposition will be "motion-sensor.motion-state == true", the condition-proposition will be "illumination-sensor.illumination < 20", and the action-proposition will be "light.on = true".

In ChatIoT, we define the ChatIoT-TAP format for TAPs as shown in Figure 2. It actually contains necessary information about the triggers, conditions, and actions. Then given a particular TAP runtime like Home Assistant, the TAP in ChatIoT-TAP format will be transformed into the format defined by the runtime. For example, the TAP shown at the bottom of Figure 2 is in the format defined by Home Assistant. Then deployment of such TAP to Home Assistant can enable this home automation rule.

3.1.3 Compressive prompting. Based on large-scale pre-training on massive text corpora and reinforcement learning from human feedback (RLHF), LLMs can produce superior capability in language understanding, generation, interaction, and reasoning. LLMs are bringing us closer to the goal of task-agnostic machine learning. Therefore, a natural idea is to ask the LLM to generate TAPs from the user request and the context. This approach is also referred to as zero-shot prompting, which fails to achieve satisfactory performance due to the following reasons.

- The context may include information about many devices, increasing the token cost significantly.
- In zero-shot prompting, the prompt include complex information including the user request, the context, and the output format. Processing such complex information leads to low TAP generation accuracy in practice.
- The initial user request may not include sufficient information for the requested TAP, e.g., the exact threshold of the room temperature.

Instead of using zero-shot prompting, ChatIoT uses a compressive prompting method to generate TAPs. This method can effectively solve the first two problems of zero-shot prompting. The solution to the third problem will be given by the context-aware prompting in the next subsection. In practice, when generating specific TAP rules, not all device information in the context is essential. To illustrate, consider a scenario where the user specifies a condition such as "Turn on the light when the motion sensor is triggered." For the generation of the corresponding TAP, the model only needs to focus on information pertinent to the motion sensor and the light. Information about other devices, such as air conditioners or televisions, is not necessary and can be disregarded.

To ascertain the relevance of a device to user requests, we extract key information from device metadata to generate descriptive narratives for each device. For instance, a description generated for a lamp in a bathroom may read, "area: bathroom, type: light, service: light, fan". The semantic similarity between the descriptions of a device and the user request often reflects the logical relevance of the device to the user's account. ChatIoT employs sentence-transformer [47] to convert these requests into vector representations within a semantic space. Then it derives the semantic similarity by computing the cosine similarity between vectors. Based on the similarities, ChatIoT can remove the information of many irrelevant devices to reduce the token cost significantly.

Algorithm 1: Compress context

Input: TAP generation request Q, Device list $D = [d_1, d_2, d_3, \ldots]$ in context
Output: Compressed device list $D_{\text{compressed}}$

Function `CompressContext(`Q, D`)`**:**
  $S \leftarrow [Q]$;
  foreach *device* d_i *in* D **do**
    Extract key information to generate string s_i;
    Append s_i to S;
  end
  Encode S by sentence-transformer;
  Calculate cosine similarity between Q and each device;
  Use DBSCAN to cluster devices based on cosine similarity;
  Filter out less relevant device clusters to get D_{temp};
  foreach *device* d_i *in* D_{temp} **do**
    foreach *service in* d_i **do**
      Encode Q and service name by sentence-transformer to get cosine similarity C;
      if $C <$ *threshold* **then**
        Remove details of this service;
      end
    end
  end
  return $D_{compressed}$;
return

Algorithm 1 shows the details of this process. Note that in the CompressContext algorithm, it further compress the "services" contained within the metadata of the remaining devices.

3.2 Context-aware Prompting in ChatIoT

As mentioned in the previous subsection, an important problem of zero-shot prompting is that the initial user request may not include sufficient information to generate the TAP. Therefore, it becomes inevitable to incorporate user feedback in practice.

In ChatIoT, we delineate the TAP generation task into four distinct sub-tasks: analyzing the TAP generation request, creating necessary services if required, generating the TAP, and evaluating the TAP. Correspondingly, we have meticulously designed four conversational *agents*: the Preprocessor, TAP Generator, Service Creator, and Evaluator. This division of tasks is structured in a way that directly mirrors the human approach to conceptualizing the generation of TAP based on a user request. These four agents are interconnected, enabling the accurate generation of TAP in a token-efficient manner.

For each agent, we have designed the system message shown in Figure 3. The system message mandates that LLMs act as an intelligent assistant within the domain of smart home technology, aiding users in accomplishing tasks. We define a specific task for each agent and outline the essential procedure required to complete it, with an emphasis on solving the task step-by-step. In this step-by-step reasoning, we have designed a series of actions that enable agents to interact with users.

In Figure 4, we illustrate the collaborative mechanism of the four agents, using a specific TAP generation request as an example. When a user issues a TAP generation request such as "When a person appears at the door, turn on the light", the *Preprocessor* agent initiates the process. It preprocesses the request based on the given context, identifying the trigger, condition, and action components, and associates them with the relevant devices, services, and properties. In this scenario, the agent identifies a camera at the door as the trigger and a light as the

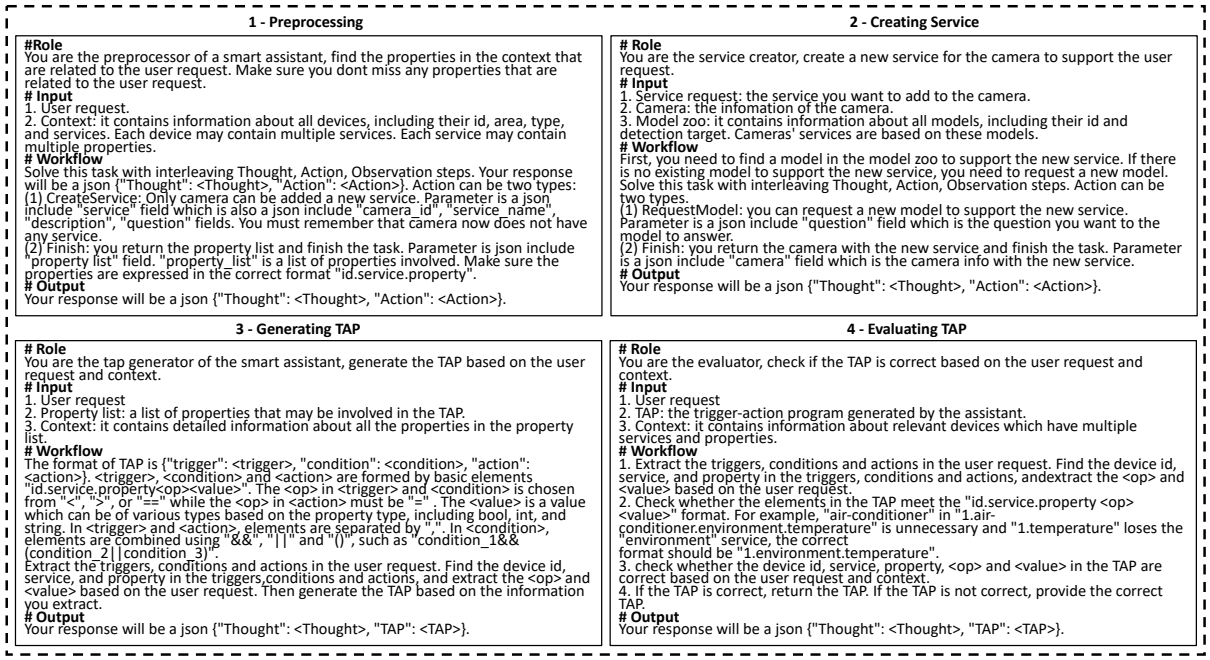


Fig. 3. System message design for each agent in ChatIoT. We configure each agent by defining its “Role” and its "Workflow" that instruct it to analyze inputs and generate corresponding outputs.

action. However, it notes that the camera does not offer a person detection service. To address this, the agent attempts to create this service by requesting the *Service Creator* agent via the action ‘CreateService’. The Service Creator agent then responds to the service request. In this case, it needs to create a person detection service for the camera, which requires a local model capable of person detection. Since there is no ready-made model in the model zoo, the agent requests the deployment of a suitable model via the action ‘RequestModel’ directed at the model controller. Details about the model controller will be described in the next section. In short, the model controller is in charge of generating the requested AI model for the camera and deploying it to the model zoo.

Once a person detection model is successfully deployed in the model zoo, it is integrated into the home assistant through the action ‘AddService’, thereby creating the required service for the camera. Upon completion, the Service Creator agent updates the camera information in the context and returns control to the preprocessor agent, which then finds the properties related to TAP and sends them to the TAP generator agent, concluding its task. The *TAP generator* agent organizes these properties into a ChatIoT format TAP and forwards it to the *Evaluator* agent. The evaluator agent assesses the TAP against the context involved, evaluating its format compliance and satisfaction of the user request. It has the capability to correct certain errors in the TAP, such as format discrepancies or irregular property values. In this example, the evaluator determines the TAP to be correct and outputs the finalized TAP, concluding the entire TAP generation task. It is important to note that, although not demonstrated in this example, any agent requiring user input can interact with the user through the action ‘AskUser’.

In summary, our approach divides the tasks generated by TAP into four sub-tasks, each handled by a specifically designed agent. These agents, in their respective reasoning processes, do not require the entirety of the context, allowing for a more focused approach to their individual sub-tasks, thus enhancing accuracy in task completion.

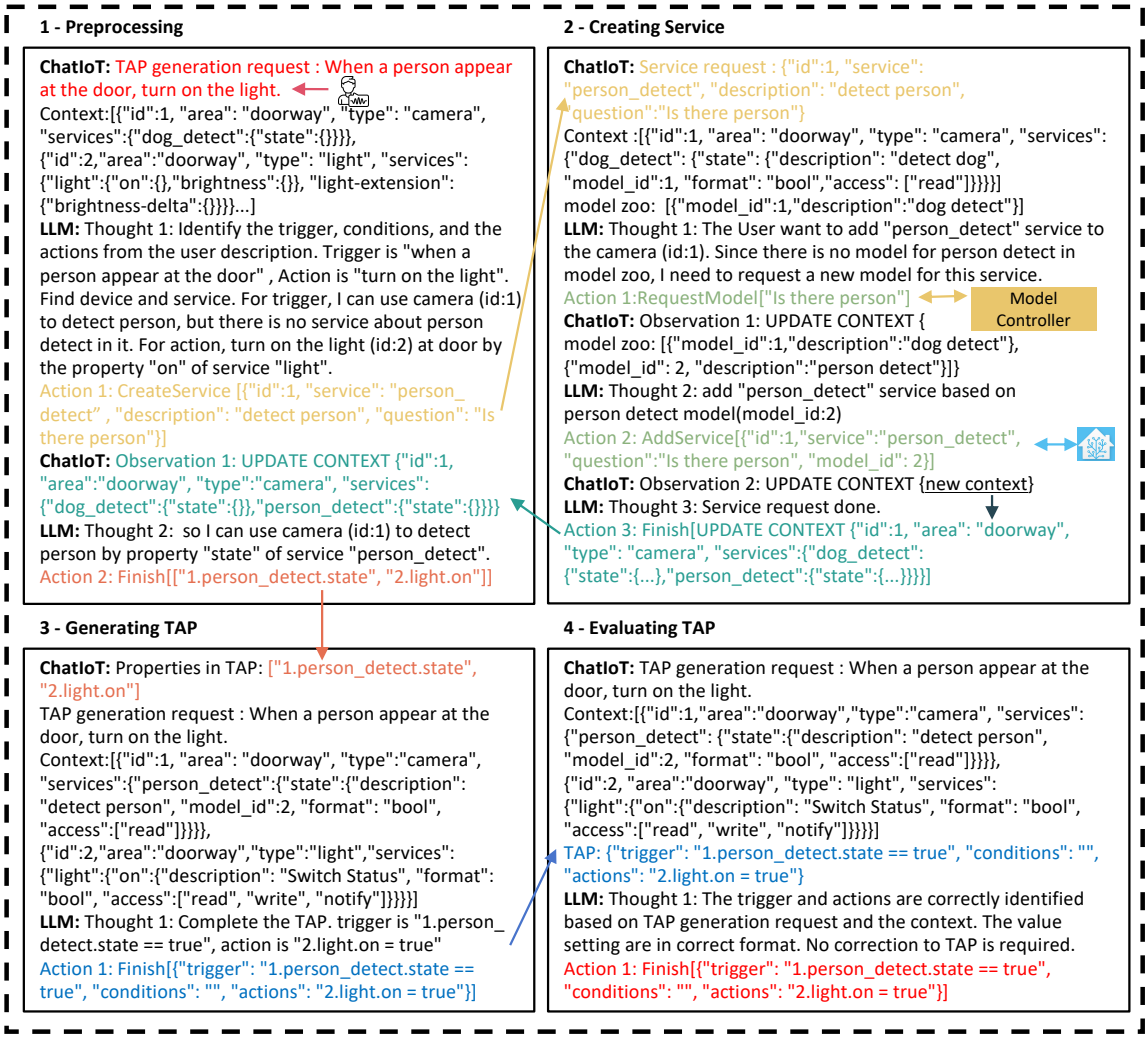


Fig. 4. An example of context-aware prompting, including four steps: preprocessing, service creation, TAP generation, and TAP evaluation. In the figure, each step is a conversation between the LLM and ChatIoT.

Given that LLMs function as stateless 'black boxes', agents must maintain the conversation, sending all previous historical information in each new round of reasoning. The reduced context size for each agent helps avoid token wastage. Through such a design, ChatIoT is capable of generating TAPs with both token efficiency and accuracy.

4 DRL-based On-demand Model Customization

In this section, we describe the on-demand model customization method in ChatIoT in detail, which is based on Deep Reinforcement Learning (DRL). When ChatIoT receives a TAP generation request in which the "trigger" should be a camera with a corresponding video analytical model, the model customization method described in this section will generate and deploy the model before the actual TAP generation. In the following, we will

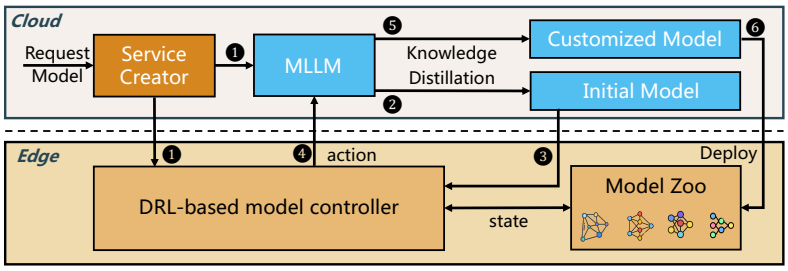


Fig. 5. Overview of Model Customization in ChatIoT. When the service creator receives a "Request Model" request, it will transport the request to DRL-based Model controller and MLLM. The DRL-based model controller deployed at the edge will determine appropriate deployment option based on the state of the Model Zoo if needed. The MLLM in the cloud will generate the customized model by knowledge distillation and deploy the model to Model Zoo using incremental update.

first give an overview of the overall framework of model customization, and then describe how to generate and deploy the newly generated model on resource-limited edge platforms.

4.1 Overview of DRL-based On-demand Model Customization

When ChatIoT receives a TAP request like "send me a message when there is a visitor at the door" while there is no "visitor sensor", ChatIoT will generate a *customized model* (i.e., the small model) from a *Multimodal LLM* (MLLM, i.e., the large model) by knowledge distillation. Then the customized model together with the camera can act as the "visitor sensor" and be used in the following TAP generation process.

Figure 5 shows an overview of this model customization method of ChatIoT. The core of model customization is the DRL-based model controller deployed at the edge, to preserve user privacy and improve the responsiveness. The controller uses DRL to determine how to deploy the newly generated model. Details about the DRL-based model controller will be described in the next subsection. At the edge, there is also the model zoo which stores and manages all the customized models. These customized models are generated at the cloud from a MLLM by knowledge distillation [26], and then deployed to the model zoo at the edge. The MLLM plays a pivotal role in two key scenarios including the generation of the initial model for users requests and the regeneration of the customized model for deployment. Since the multimodal capabilities of recent models have been significantly improved by the advance of LLMs, ChatIoT is able to generate various customized models from MLLMs for different user requests.

This approach is efficient in terms of resource utilization and scalable, facilitating the incremental introduction of new models based on demand, rather than deploying an extensive array of models initially. In the cloud-edge hybrid architecture, the cloud serves as a reservoir of knowledge and computational power, aiding edge devices in maintaining a streamlined yet effective model zoo.

4.2 On-demand Model Generation

4.2.1 Motivation of using Knowledge Distillation. The primary challenge is the generation of models tailored to user requirements. Real-world applications often necessitate models capable of processing diverse inputs, including visual data. Multimodal large language models (MLLMs), such as VisualGLM[22, 23], trained on extensive and varied datasets, possess the ability to comprehend and address these complex requests efficiently. Nonetheless, the computational demands of these comprehensive models render them impractical for deployment on edge devices with constrained resources. For example, VisualGLM necessitates an exorbitant 20GB of memory, which exceeds the capacity of many edge environments.

Techniques such as weight quantization have been introduced for the edge deployment of LLMs, yet they continue to require considerable computational resources and storage space. The MiniCPM-V [11], encompassing 2.8 billion parameters, demands 6.87GB of storage, a significant amount compared to a model like ResNet101[29], which, with only 44.5 million parameters, needs 170.5MB of storage. Furthermore, the extended inference times associated with large models can result in delayed responses amid a high volume of user requests. Consequently, knowledge distillation from MLLM is advocated to address the varied and dynamic nature of user requests, alongside the obstacle of limited labeled data.

4.2.2 MLLM-based Knowledge Distillation. Given the impracticality of preparing all labeled data or specific models to meet the diverse needs of users, we utilize a MLLM trained on a large-scale dataset as the teacher model to generate the required specific models in real time based on user needs. In ChatIoT, VisualGLM is employed as the teacher model in knowledge distillation. As illustrated in Figure 5, VisualGLM generates the customized model under the guidance of a DRL-based model controller.

Upon receiving requests, VisualGLM begins to collect unlabeled data from the internet, tapping into the vast information available online. With its advanced classification capabilities, VisualGLM annotates this data, transforming unstructured information into a structured, labeled dataset. These labeled data then become invaluable resources for training lightweight models at the edge, equipping them with high-quality, pre-labeled data. This strategy ensures that models operating with limited resources can achieve high accuracy and performance, effectively harmonizing the demands for efficiency and reliability in output.

4.3 DRL-based Model Controller

4.3.1 The functionalities of the controller. The core of model customization is the DRL-based model controller. As Figure 6 shows, the model controller needs to determine how to deploy the newly generated model. VisualGLM, our chosen multimodal LLM, plays a pivotal role in two key scenarios including the generation of the initial model for users requests and the regeneration of the customized model for deployment. For example, there are already two models deployed in the model zoo, i.e., model a and mode b. When a new model c (e.g., for the "visitor sensor") is required to be generated and deployed, ChatIoT will first generate the model c at the cloud by knowledge distillation from the MLLM. Then the controller needs to select one of the following three deployment options: 1) direct deploying model c to the model zoo in case of sufficient remaining memory at the edge; 2) replacing one existing model (i.e., model a or model b) with model c; and 3) selecting two models from the existing models (i.e., model a or model b) and the newly generated mode c and performing model merging.

When there is sufficient memory for the model zoo at the edge, the controller simply selects the first option, i.e., direct deployment of the newly generated model. Otherwise, ChatIoT uses a DRL-based decision making process to determine which model to replace or which two models to merge.

4.3.2 Motivation of using DDPG. The next question is which DRL algorithm to use. In our model customization problem, an issue arises from the uncertain number of existing models on the edge, resulting in an unknown dimension for the action space in the DRL process. In ChatIoT, we use the Deep Deterministic Policy Gradient (DDPG) algorithm [37]. While DDPG is usually suited for addressing problems with a continuous action space, it can be used to solve problems with an action space of unknown dimension. Some existing work[24, 51] has demonstrated the feasibility of solving discrete problems by using continuous action spaces with high performance. Therefore, by *mapping* the continuous space to the desired action space, we leverage the capabilities of the DDPG algorithm to effectively tackle this issue.

4.3.3 DDPG-based model customization. Figure 7 shows the overall training workflow of the process of the DDPG-based model customization approach at the cloud. In the DDPG-based model customization, the actor-critic networks collectively form the ChatIoT agent, while the environment encompasses the entire system in which

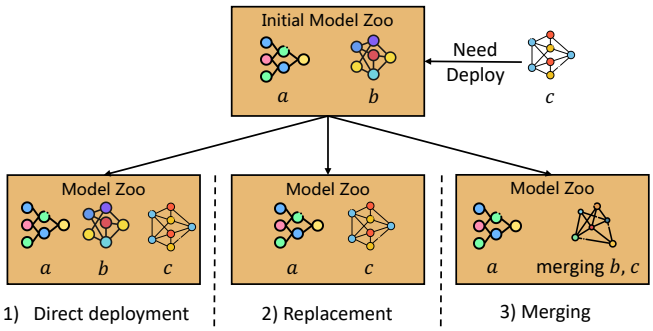


Fig. 6. Three deployment options determined by model controller. The initial model zoo has already deployed models a and b. When it is necessary to deploy model c for solving the newly arriving requirements, the model controller determine one of three options.

Table 2. Summary of notations

Notation	Description
$\mathcal{M}$	Set of AI models
C	Model average accuracy
v_i	Memory of model m_i
V_{max}	Max memory
$x_{i,j}$	Merge two models m_i and m_j
y_i	Replacement of model m_i
$\mathcal{S}$	State space
$\mathcal{A}$	Action Space
$\mathcal{N}$	Noise
r	Reward
d	The size of delta

the agent operates. Within the environment, there exists an efficient reward calculation process based on the current states and the actions provided by the agent. We will describe the details of this reward calculation in the next subsection. The MLLM at the cloud will receive the deployment decision from the agent, then deploy the merged model (in case of model merging) or the original generated model (in case of model replacement) to the model zoo.

We summarize the state and action spaces, reward function, the state transition policy that are used in our DDPG-based Model customization framework and how to deploy trained DRL networks at the edge in the following. Table 2 presents a list of the relevant notations and their corresponding descriptions.

State Space: The state space is primarily utilized to describe the environmental information, encompassing the model zoo, including the problem it addresses, accuracy, and memory usage. We employ the notation $\mathcal{S}$ to represent the state space, where each $s \in \mathcal{S}$ is denoted as a three-tuple $[\mathcal{M}, C, \mathcal{V}]$. Here, $m \in \mathcal{M}$ represents a model in the model zoo, $c \in C$ represents the accuracy of each model, and $v \in \mathcal{V}$ represents the memory required by each model for inference.

Action Space: The action space pertains to the actions performed by the policy module for model customization. Let $\mathcal{A}$ represent the action space and each $a \in \mathcal{A}$ is a decimal number in the range of -1 to 1. We use $x_{i,j}$ as a binary variable that indicates the merging of model m_i with model m_j, while y_i as a binary variable that indicates

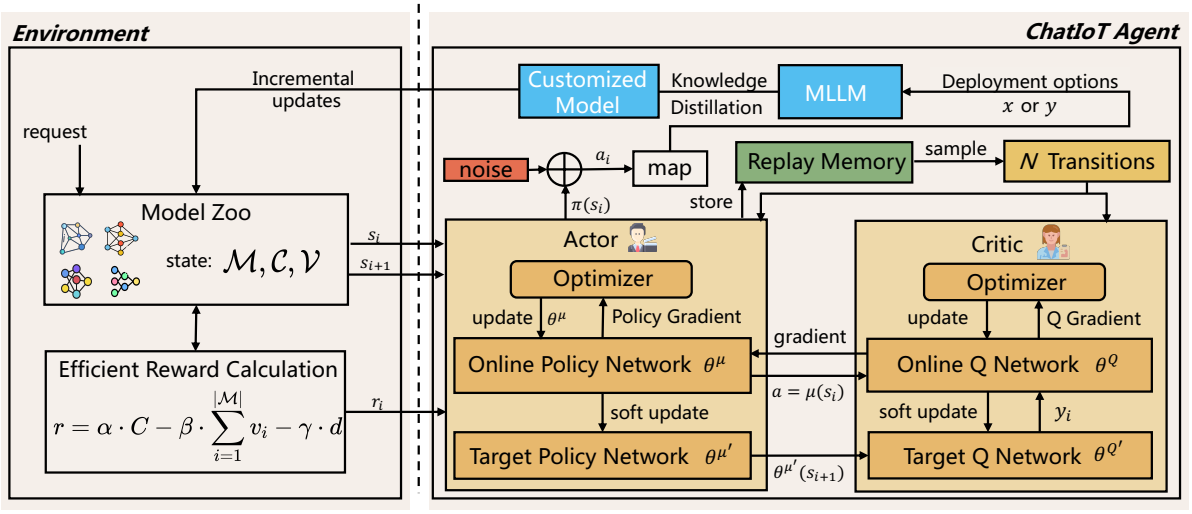


Fig. 7. Overall Workflow DDPG-based Model Customization.

the replacement of model m_i with the model needed for the current problem. As previously stated, the process of mapping the continuous action space of the DDPG algorithm to a discrete decision space involves the following steps.

$$\begin{cases} \text{model merging} & \text{if } a \in (-1, 0] \\ \text{model replacement} & \text{else } a \in (0, 1) \end{cases} \tag{1}$$

Initially, we assume that when the value of action a falls within the range of -1 to 0, we employ a model merging approach and i and j in $x_{i,j}$ can be obtained by using the following equations:

$$i = \lfloor (a+1) \cdot (|\mathcal{M}|+1) \rfloor \tag{2}$$

$$j = \lfloor [(a+1) \cdot (|\mathcal{M}|+1)^2] \bmod (|\mathcal{M}|+1) \rfloor \tag{3}$$

Conversely, when the value of action a falls within the range of 0 to 1, we utilize a model replacement approach and we have i in y_i can be obtained as

$$i = \lfloor \frac{a}{|\mathcal{M}|} \rfloor \tag{4}$$

By employing this binary variable approach, we can effectively handle the unknown dimension of the action space and enable the mapping of continuous actions from DDPG to the current required action space. The customization of models is subjective to the following constraints:

$$\sum_{i=1}^{|\mathcal{M}'|} v_i \leq V_{max} \tag{5}$$

where $\mathcal{M}'$ indicates the currently models after deployment require less memory than the memory limit.

Reward Function: Generally, the DRL agent aims to maximize the reward in the training process. Therefore, we use average model accuracy C, memory usage $\sum_{i=1}^{|\mathcal{M}|} v_i$, and delta size d as the three features for measuring reward, and measure their importance by three coefficients α, β, γ. Note that ChatIoT uses an incremental update [33] method to transmit only the difference between the updated model and the existing model on the

edge device, to save the limited bandwidth between the cloud and the edge. Therefore the delta size d is included in the reward calculation. Then the reward function in our framework is measured as follows:

$$r = \alpha \cdot C - \beta \cdot \sum_{i=1}^{|\mathcal{M}|} v_i - \gamma \cdot d \tag{6}$$

In our reinforcement learning setup, once the action is to merge two models, we need to quickly obtain the reward including model accuracy, memory and delta size of the new model to speed up the training process of the DRL networks. Therefore, we need to use a predictive model, which can predict the reward efficiently in advance before the actual training of the model. We will discuss our prediction model in the next subsection.

State Transition Policy: DDPG can be divided into two major networks: policy network μ and value network Q. DDPG continues the idea of fixed target network with DQN[45], and each network is subdivided into target network and online network. Random initialization of the online Q network $Q(s, a|\theta^Q)$ and the online policy network $\mu(s|\theta^\mu)$ parameters of the θ^Q and θ^μ and with the parameters $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$ initialize their corresponding target networks Q' and μ'. The action is selected according to the current policy as follows:

$$a_i = \mu(s_i|\theta^\mu) + \mathcal{N}_i \tag{7}$$

where a_i is the action selected by the current policy network, s_i is the current observed state, and $\mathcal{N}_i$ is the exploration noise.

The decision process of the DDPG algorithm is to perform actions in the environment to obtain the reward r_i , the newly observed state s_{i+1}, the sample data (s_i, a_i, r_i, s_{i+1}), and store them in the experience replay buffer. The target Q value of the Critic is calculated as in Equation 8. The Critic is updated by minimizing TD deviation as shown in Equation 9.

$$y_i = r_i + \epsilon Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{8}$$

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{9}$$

The Actor's policy network updates the network parameters based on the Policy Gradient, as shown in Equation 10.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i (\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}) \tag{10}$$

where $\nabla_{\theta^\mu} \mu(*)$ is the Actor network gradient and $\nabla_a Q(*)$ is the Critic network gradient. $\nabla_{\theta^\mu} J$ allows the Actor to continuously adjust its network parameters in the maximum direction available reward θ^μ.

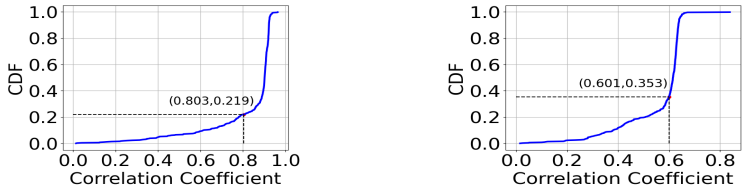
The Actor's online network is updated in real-time based on the Critic's online network as a guide, which is updated in real-time using its target network as a guide. Therefore, the parameters of the online network are up to date. In contrast, the target network parameters are delayed based on the online network parameters using soft updates, as shown in Equation 11.

$$\begin{cases} \theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{cases} \tag{11}$$

where $\theta^{Q'}$ is the target Q network parameter, $\theta^{\mu'}$ is the target policy network parameter, and τ is the soft update constant. As Figure 7 shows, the ChatIoT agent interacts with environment by selecting an action based on the current model state and given request. The selected action is executed in the environment, resulting in a transition to a new state. Through this ongoing interaction, our DDPG model will continue to apply its acquired knowledge to make informed decisions, adapt to the environment, and optimize its performance within the model merging and replacement.

Table 3. Summary of attributes for reward calculation

Attribute	Description
weight size	Storage size of the two model weights
model accuracy	Accuracy of models generated by knowledge distillation on the validation set
layer-wise model similarity	Similarity of each layer of the network obtained with the help of SSIM
model similarity	Weighted average of layer-wise model similarity



(a) Correlation with Accuracy (b) Correlation with Delta Size

Fig. 8. CDF of correlation coefficients between selected features and two variables. It can be seen that a large number of features are strongly correlated with the the accuracy and the delta size.

When the DRL networks are trained, ChatIoT will deploy the trained policy network in DDPG to the edge, forming part of the DRL-based model controller described above. Then during the actual TAP generation, the DRL-based model controller will calculate the deployment decision about the newly generated model, i.e., direct deployment, model replacement or model merging.

4.4 Efficient Reward Calculation

As mentioned earlier, during the training of the DRL networks, ChatIoT needs to efficiently calculate the reward (i.e., Equation 6) given an action to speed up the training process. More specifically, we consider the following three elements in the reward calculation, overall memory usage, the average model accuracy after model merging, and the delta size for incremental model update.

The memory usage of a model during inference is determined by the model structure and input samples. Since the model structures before and after model merging are predefined and the format of the input samples are also known, we can directly obtain the overall memory usage. Therefore, that leaves the following two elements for calculation: average model accuracy and the size of the delta.

In ChatIoT, the following attributes are selected as features to calculate the average model accuracy and the size of the delta, including weight size, model accuracy, layer-wise similarity and model similarity of the two models prior to merging. Table 3 describes the meanings of these features in detail. To estimate the similarity of two neural networks, we use SSIM [32] to calculate the layer-wise similarity of the two models. SSIM is an algorithm used to measure the similarity of two images. Since the parameters of each layer of the neural network can be considered as a two-dimensional matrix, we can calculate the layer-wise similarity of the two models with the help of SSIM. By incorporating these attributes, we aim to establish a robust regression model that enables accurate calculation of the reward.

We conducted experiments to unveil the correlations between the predicted outcomes and the underlying features. We use the Pearson coefficient $p(\cdot)$ to measure the correlation coefficient between two variables. Through the computation of correlation coefficients, it has been observed in Figure 8 that most of the independent variables

Table 4. Devices in different home environment

Home	Bathroom	Bedroom	Living room	Kitchen
Home 1	light, motion_sensor	light, air_conditioner	illumination_sensor	None
Home 2	light, motion_sensor, illumination_sensor	light, air_conditioner, illumination_sensor, temperature_humidity_sensor	light, motion_sensor, illumination_sensor	None
Home 3	light, motion_sensor, illumination_sensor	light0, air_conditioner, illumination_sensor, temperature_humidity_sensor, light1, curtain, humidifier	light0, motion_sensor, illumination_sensor, light1, light2, air_purifier, air_conditioner	light, range_hood, motion_sensor
Home 4	light, motion_sensor, illumination_sensor, water_heater	**Bedroom 1:** light0, air_conditioner, illumination_sensor, temperature_humidity_sensor, light1, curtain, humidifier, dehumidifier **Bedroom 2:** light0, air_conditioner, illumination_sensor, temperature_humidity_sensor, light1, curtain, humidifier	light0, motion_sensor, illumination_sensor, light1, light2, air_purifier, air_conditioner	light, range_hood, motion_sensor, ceiling_fan

as mentioned above exhibit a significant correlation with the accuracy and delta size of the merged model. For example, within a total of 524 features (i.e., weight size, accuracy, model similarity and layer-wise model similarity) in this experiment, 78% of the features exhibit a correlation coefficient with accuracy higher than 0.8 and 65% of the features exhibit a correlation coefficient with delta size higher than 0.6. To speed up the following MLP-based regression, we use PCA for dimension reduction. The resulting mean squared error (MSE) values are 0.0078 and 0.0033 which prove that our prediction algorithm can achieve high accuracy. After predicting the accuracy of the merged model as well as the size of the delta, we can quickly calculate the reward and send it to ChatIoT agent.

5 Evaluation

In this section, we will first introduce the setup, including the datasets, baselines, and evaluation metrics. Then we will evaluate the performance of ChatIoT through a comparative study in terms of multiple metrics. Finally, to show the effectiveness of ChatIoT, we report a case study with a DIY smart home including both DIY devices and Commercial-off-the-Shelf (i.e., COTS) devices.

5.1 Setup

5.1.1 Experiments Hardware and Models. We constructed a real-world prototype with edge devices (an Jetson Xavier NX with 8GB Memory) and a public Cloud (Ubuntu 20.04, 16 vCPU Intel(R) Xeon(R) Gold 6430, 120GB DDR4 RAM and RTX 4090 (24GB)) to evaluate the performance of ChatIoT. ChatIoT at edge employs sentence-transformer [47] to match user requests and home information. The matched information and requests are then forwarded to ChatGPT-3.5-turbo for the TAP generation. Referring to some previous works [54, 59], we set the temperature parameter to 0.7 and the maximum token to 512.

5.1.2 Home Setup. To evaluate the performance of ChatIoT, we utilized the *python-miio*[10] library to generate 65 devices for evaluation, including light sensors, motion sensors, lamps, air conditioners, and others. We deployed four instances of Home Assistant using Docker to emulate households with varying numbers of devices (Home 1 with 5 devices, Home 2 with 10 devices, Home 3 with 20 devices, and Home 4 with 30 devices). In addition, every home has one camera at the entrance. Table 4 shows the devices each home has in different rooms in detail.

5.1.3 Request Datasets. Due to the unavailability of publicly accessible datasets specifically designed for user requests on smart homes, we have generated a set of categorized requests with the assistance of GPT-3.5-turbo model for each household. Among these requests, there exist 20 TAP generation requests which have no relation to camera and 10 requests related to camera. In addition, we extend requests related to camera to 28 for evaluating the model customization module of ChatIoT. To automatically generate these models, such as identifying the presence of a cat in an image, we leveraged data from relevant categories available in ImageNet[20] and applied knowledge distillation. For categories that are not present in ImageNet, such as sports, we acquired them through web crawling. In the knowledge distillation process, the VisualGLM[22, 23] is utilized as the teacher model, while the student model employed is ResNet101[29].

5.1.4 Baselines and metrics. To evaluate the performance of context-aware prompting, we compare ChatIoT with three types of baselines.

- Zero-shot: Without providing any examples, LLMs are tasked to complete the TAP generation based on its internal knowledge. The TAP generation is executed within an agent, with the system message specifying only the required output format for TAP.
- CoT[56]: Based on zero-shot, three examples are added to show the process of TAP generation, and LLMs are required to think step by step in the system message.
- ReAct[59]: Based on CoT, allowing LLMs to use actions to interact with users and systems during reasoning.
- ChatIoTLite: Based on ChatIoT, but does not compress home information.

We set the device threshold of cosine similarity to 0.45, and the service threshold to 0.35 based on our additional evaluation. We report the token consumption and TAP generation accuracy of the context-aware compressive prompting used in ChatIoT. We manually generated the TAP rules corresponding to all requests as ground truth in advance. By traversing all generated TAPs by each algorithm, the TAP generation accuracy in each smart home setting can be easily calculated.

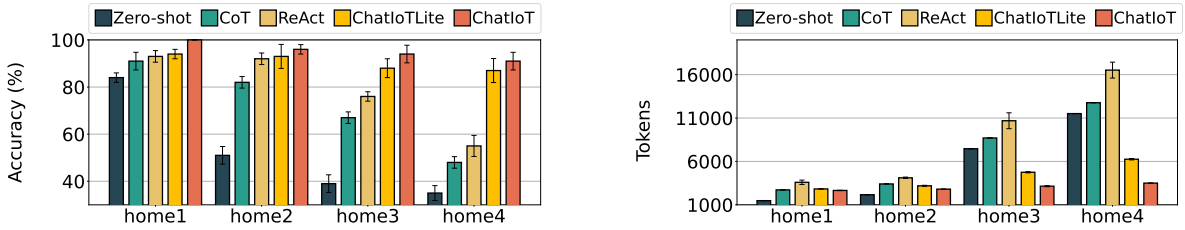
Similarly, to evaluate the performance of DRL-based model customization, we also evaluate following three types of baselines.

- Greedy Replacement (i.e., GR): A greedy algorithm for replacing a model according to a pre-defined greedy rule (i.e., Minimum loss of accuracy) when resources are not sufficient for direct placement of the generated model. Then the accuracy of the task been replaced becomes very low since it can only output the results by random guess.
- Random Merging (i.e., RM): A stochastic algorithm for randomly selecting two models from the existing models and the newly generated model and performing model merging when there is not enough memory to place it.
- Brute Force (i.e., BF): An exhaustive algorithm is used to exhaust all possible model deployment actions and select the one that achieves the maximum average model accuracy. Since the exhaustive algorithm is time-consuming (and not feasible in real scenarios), we report the results only when the execution is able to finish within a reasonable amount of time.

We report the average model accuracy and reasoning time of the DRL-based model customization used in ChatIoT. As for the parameter settings in DDPG, we set the training iteration steps to 300, with the learning rates for the actor at 0.001 and critic at 0.002. The soft update constant τ in Equ.11 is set to 0.01 and ϵ in Equ.8 is set to 0.9. In Equ.6, the three weights α, β, γ of the reward function are set to 0.9, 0.05, and 0.05, respectively.

5.2 Performance of TAP Generation

In this section, we evaluate the performance of ChatIoT in households with varying numbers of devices, using two metrics: TAP generation accuracy and token consumption. Figure 9 displays the average TAP generation



(a) TAP generation accuracy (b) Token consumption

Fig. 9. Performance of TAP generation for requests not related to cameras in terms of token consumption and accuracy.

Table 5. Detail results including TAP generation accuracy and token consumption over camera-related requests.

Home	Context	Zero-shot		CoT		ReAct		ChatIoTLite		ChatIoT	
		Acc. (%)	Token	Acc. (%)	Token	Acc. (%)	Token	Acc. (%)	Token	Acc. (%)	Token
home1	1118	0	1466.14	0	2685.48	80	9634.12	100	5109.3	**100.0**	4580.68
home2	1799	0	2155.64	0	3383.54	96	12353.62	98	5737.18	**100.0**	4888.06
home3	7098	0	7452.36	0	8675.08	68	28132.60	90	9073.28	**92.0**	5368.58
home4	11144	0	11500.7	0	12718.42	64	40438.04	78	11518.16	**86.0**	6067.10

accuracy and token consumption for requests not related to cameras. Furthermore, Table 5 presents detailed results for requests related to cameras.

TAP generation accuracy: Figure 9(a) demonstrates the average accuracy of TAP generation for requests not related to cameras using five different prompting methods across various household settings.

Results indicate that among various home settings, the zero-shot approach exhibits the lowest TAP generation accuracy, followed by CoT and ReAct methods. Employing the context-aware prompting, both ChatIoTLite and ChatIoT significantly enhance the accuracy of TAP generation. With the increase in the number of devices in homes, the density of key information pertinent to TAP generation requests within the context diminishes, resulting in a decrease in the accuracy of TAPs generated by all prompting methods. By implementing compression strategies to eliminate irrelevant information, ChatIoT consistently attains the highest TAP generation accuracy. Furthermore, in comparison to ChatIoTLite, which does not filter out redundant information, the higher TAP generation accuracy of ChatIoT underscores the efficacy of compressive prompting.

Results presented in Table 5 show the detailed results including TAP generation accuracy and token consumption over camera-related requests. Several studies [54, 63] indicated that when inference necessitates additional information external to the LLM, the Zero-shot and CoT methods fail to deliver satisfactory performance. Due to their inability to generate models relevant to cameras in the home, Zero-shot and CoT approaches achieve a 0% accuracy in generating camera-related TAPs. In contrast, ReAct, ChatIoTLite, and ChatIoT demonstrate the ability to determine the need for creating pertinent models based on user requests. As can be seen from Table 5, ChatIoT can achieve the highest TAP generation accuracy in all home settings while consuming the least token.

Token consumption: Figure 9(b) illustrates the token consumption for five prompting methods in various households. Notably, as the number of devices increases, the token consumption of the other four methods escalates dramatically, whereas the rise in token consumption for ChatIoT is comparatively gradual.

The Zero-shot approach utilizes a simple prompt, with its token usage primarily derived from device information. As indicated in Table 5, the representation of more devices necessitates a greater number of tokens, leading to

a substantial increase in token consumption of zero-shot approach as the number of devices grows. The CoT method, by incorporating examples to enhance accuracy, incurs additional token overhead. To broaden existing services, ReAct employs multiple rounds of reasoning and action, necessitating further token usage.

ChatIoTLite and ChatIoT implement a compressive context-aware prompting scheme, enabling LLMs to infer without requiring the full context information and to supplement context as needed during inference. As depicted in Table 5, within the Home 4 setup, despite the Zero-shot approach's inability to process requests necessitating additional services, it still consumes more than 10,000 tokens. ReAct, which can make decisions to request additional models based on user requests, requires more than 40,000 tokens and has an accuracy of 64%. In contrast, ChatIoT attains an accuracy of 86% while utilizing only 6,000 tokens.

5.3 System Insights

5.3.1 Parameter study for compressive prompting. As delineated in Algorithm 1, two thresholds are established for the elimination of irrelevant devices (line 10) and services (line 15), respectively. By establishing the thresholds that best maintain the integrity of necessary home information while maximizing compression, we can significantly improve the system's overall performance. When the threshold is set too small, the compression rate will be reduced; when the threshold is set too large, relevant information will be accidentally deleted, resulting in TAP generation failure.

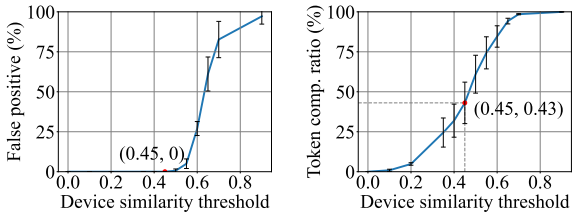
To assess the impact of varying thresholds on the compression of home information and the potential for erroneous data removal, we devised a comprehensive evaluation strategy. This strategy involved preparing a set of user requests that were distinct from those utilized in the performance evaluation of ChatIoT, as detailed in Section 5.2. Accompanying each user request, we manually obtained the minimal relevant home information(i.e., the related devices and services) for generating accurate TAP rules.

In the ablation studies, we investigated the correlation between "False Positive", the token compression ratio of home information, and the cosine similarity threshold. The ratio of mistakenly removed information is represented as "False Positive" which indicates the proportion of mistakenly eliminated information to the minimum necessary data. The token compression ratio of home information denotes the proportion of tokens in compressed home information relative to uncompressed information.

Initially, we assessed the performance of the compressive promoting method in ChatIoT under various device similarity thresholds between the user request and all devices. We conducted experiments on different numbers of devices and the results illustrated in Figure 10 reveal that the token compression ratio significantly increases as the similarity threshold increases, with a corresponding gradual rise in the "False Positive". Notably, at a similarity threshold of 0.45, relevant information is just not mistakenly removed, ensuring minimal token overhead.

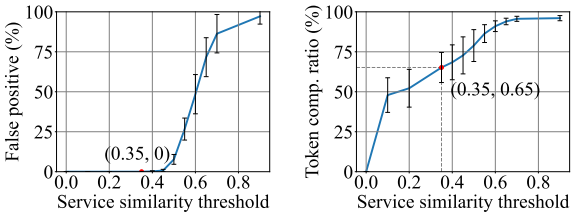
Subsequently, we evaluated the performance of ChatIoT under different service similarity thresholds, with the device similarity threshold set at 0.45. In alignment with Figure 10, Figure 11 demonstrates that as the similarity threshold increases, the token compression ratio and the "False Positive" gradually increase. To achieve an optimal compression ratio without mistakenly removing the relevant information, we determined the device similarity threshold at 0.45 and the service similarity threshold at 0.35. By compressive prompting, ChatIoT can effectively filter out unnecessary information from irrelevant devices and services. Under experimental settings with different numbers of devices, ChatIoT can achieve a compression ratio of 65% without mistakenly removing the relevant information.

5.3.2 Ablation study for TAP generation. We conducted an ablation study under the Home 4 setup to evaluate the performance of the preprocessor and the TAP evaluator in ChatIoT. Results depicted in Table 6 indicate that the inclusion of a preprocessor and TAP evaluator can enhance the performance of ChatIoT. By integrating both a preprocessor and an evaluator, ChatIoT can improve TAP generation accuracy to 86-91% over the direct approach.



(a) False Positive (b) Token compression ratio

Fig. 10. Performance of the compressive promoting method in ChatIoT under different device similarity thresholds.



(a) False Positive (b) Token compression ratio

Fig. 11. Performance of the compressive promoting method in ChatIoT under different service similarity thresholds.

Table 6. Impact of preprocessor and evaluator in ChatIoT

Pre.	Eva.	Camera	Accuracy (%)	Token consumption				
				Total	Preprocessor	Creator	Generator	Evaluator
×	×	×	$73.00_{\pm 4.83}$	$3921.84_{\pm 164.66}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$3921.84_{\pm 164.66}$	$0.00_{\pm 0.00}$
		✓	$70.00_{\pm 0.00}$	$7994.08_{\pm 3.57}$	$0.00_{\pm 0.00}$	$1210.36_{\pm 3.37}$	$6783.72_{\pm 1.54}$	$0.00_{\pm 0.00}$
×	✓	×	$86.00_{\pm 6.99}$	$7689.62_{\pm 290.42}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$3921.84_{\pm 164.66}$	$3755.66_{\pm 113.19}$
		✓	$82.00_{\pm 4.47}$	$11097.32_{\pm 11.78}$	$0.00_{\pm 0.00}$	$1210.36_{\pm 3.36}$	$6783.72_{\pm 1.54}$	$3103.24_{\pm 11.69}$
✓	×	×	$78.00_{\pm 7.89}$	$2478.65_{\pm 114.83}$	$1576.41_{\pm 57.47}$	$0.00_{\pm 0.00}$	$902.24_{\pm 65.28}$	$0.00_{\pm 0.00}$
		✓	$74.00_{\pm 5.48}$	$5165.10_{\pm 10.54}$	$3111.32_{\pm 3.90}$	$1210.70_{\pm 2.12}$	$843.08_{\pm 9.91}$	$0.00_{\pm 0.00}$
✓	✓	×	$91.00_{\pm 11.01}$	$3502.11_{\pm 196.78}$	$1576.41_{\pm 57.47}$	$0.00_{\pm 0.00}$	$902.24_{\pm 65.28}$	$1023.46_{\pm 84.58}$
		✓	$86.00_{\pm 5.48}$	$6067.10_{\pm 24.28}$	$3111.32_{\pm 3.90}$	$1210.70_{\pm 2.12}$	$843.08_{\pm 9.91}$	$902.00_{\pm 14.24}$

Compared to the conventional approach of direct TAP generation, the incorporation of a preprocessor into the workflow significantly enhances the system's efficiency and accuracy. Specifically, by employing a preprocessor, TAP generation accuracy can be markedly improved, achieving a 74-78% accuracy. This improvement is largely attributed to the preprocessor's ability to meticulously analyze and prepare the context before TAP generation, ensuring a more informed and accurate TAP creation process. Through its sophisticated preprocessing mechanism, the preprocessor effectively reduces the number of contexts required for generating TAP rules. Furthermore, the preprocessor enables rapid identification of gaps in the availability of relevant services and models allowing for timely development and integration of necessary services or models to fill these gaps.

The integration of a TAP evaluator into the system significantly bolsters the accuracy of TAP generation, elevating it to 82-86%. This enhancement is attributable to the evaluator's role in meticulously assessing each TAP rule against the user request and relevant contexts. The evaluator not only verifies the rule's compliance with the required format but also ensures that it effectively fulfills the user's request, thereby ensuring high accuracy and relevance in the actions triggered by the system. Furthermore, it has the capability to correct certain errors in the TAP, such as format discrepancies or irregular property values which ensure that they operate seamlessly within the system.

5.4 Performance of On-demand Model Customization

In this section, we report the performance of the DRL-based on-demand model customization framework against two baselines using two different evaluation metrics.

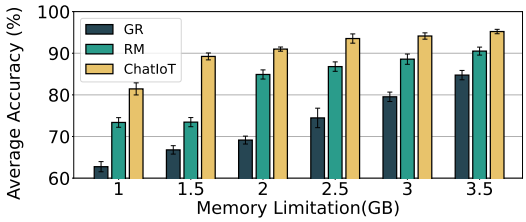
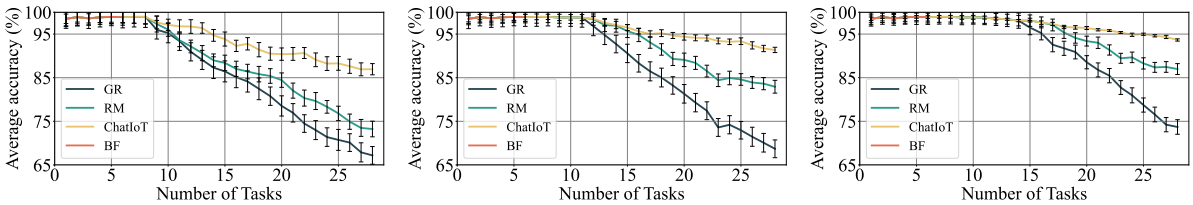
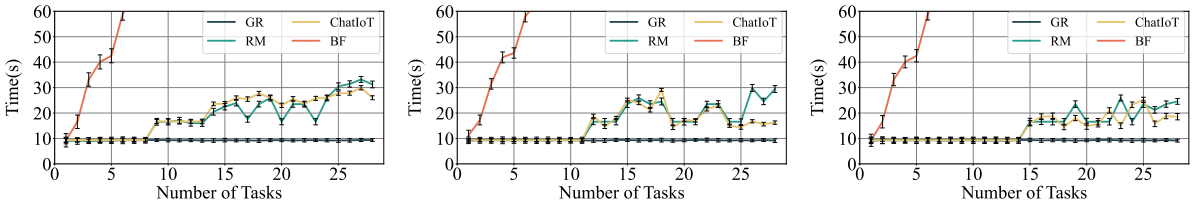


Fig. 12. Performance of three methods in terms of average model accuracy over different memory constraints. Results come from 50 different user request arrival sequences. ChatIoT achieves the highest accuracy over all memory constraints.



(a) memory=1.5GB (b) memory=2GB (c) memory=2.5GB

Fig. 13. Average model accuracy performance of the four algorithms over the number of tasks.



(a) memory=1.5GB (b) memory=2GB (c) memory=2.5GB

Fig. 14. Reasoning time of four algorithm over the number of tasks.

Average model accuracy: We show the average model accuracy results here. Figure 12 shows the accuracy of two baselines and our DRL-based model customization method in ChatIoT.

To begin with, we find that under diverse memory constraints, the *greedy replacement algorithm* always has the lowest accuracy due to its tendency to degrade the accuracy of a significant number of models. By utilizing model merging, the *random merging algorithm* outperforms the greedy replacement algorithm with different memory constraints. ChatIoT can outperform other baselines because of the DRL-based approach which considers both model replacement and merging options in an intelligent way. Additionally, it is observed that easing memory constraints leads to an increase in accuracy while the performance of ChatIoT improvement over the two baselines is more pronounced in scenarios with more severe resource constraints.

We also performed an analysis of the trend in average model accuracy with a number of tasks under specific memory constraints. Figure 13 provides an illustration of the results. The varying memory constraints correspond to the number of models that can be deployed individually. As the memory constraint increases, each algorithm

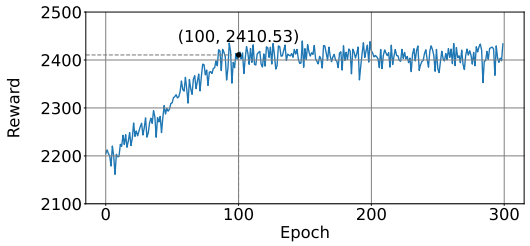


Fig. 15. Trends of reward. DDPG component can converge after 100 epochs (approximately 52.2 seconds).

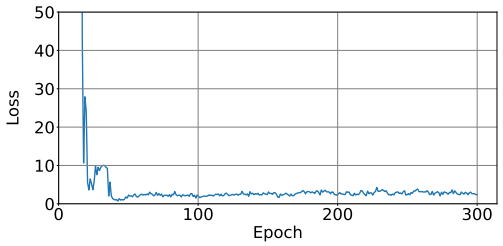


Fig. 16. Loss trends of Critic. The loss of the critic indicates a steady decline during training.

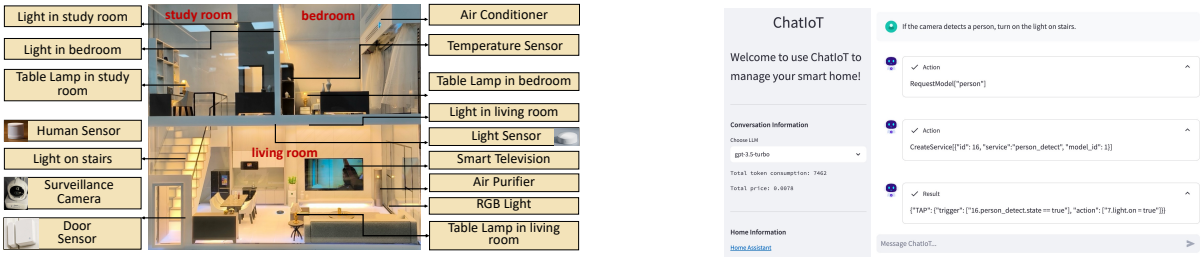
demonstrates a decrease in average accuracy only after accommodating a larger number of tasks. When the memory required for the desired model falls within the available memory capacity, direct deployment of the model yields high accuracy. However, when the newly arrived model lacks sufficient memory for deployment, performance differences among the three algorithms become evident. Under the same memory constraint, the *greedy replacement algorithm* replaces one of the existing tasks once the memory limit is reached, resulting in a rapid decline in average accuracy. Similarly, the *random merge algorithm* exhibits a subsequent rapid accuracy drop by merging two potentially unsuitable models. In contrast, our algorithm, leveraging the pre-trained DDPG model, selectively chooses the appropriate replacement or merging candidates based on the existing models and the current task requested. This minimizes the impact of the newly deployed model on the average accuracy, enabling it to remain high as the number of tasks fluctuates.

Reasoning time: We also analyzed the reasoning time of the algorithms including decision making, model generation and computation of deltas in Figure 14. The results indicate that as the number of tasks expands, the reasoning time for the exhaustive algorithm experiences a substantial increase. Considering the user experience, the exhaustive algorithm is not a viable option in practice. The *greedy replacement algorithm* only needs to decide which model to replace, and its main time consumption lies in the model generation, so that the total time is always minimal. The time consumption of the random merging algorithm and ChatIoT is similar since the time-consuming model generation process contributes to both of the two methods. However, when there are a large number of tasks, the random merging algorithm ends up with higher time consumption compared with that of ChatIoT. This is because the DRL-based model customization framework in ChatIoT will intelligently choose the model merging option or the model replacement option which cost less time.

Additionally, we also consider the deployment time of the model in real scenarios. With the incremental update strategy, the size of the delta is only about 210KB, while the size of the whole model is about 170MB. Therefore, incremental update can significantly reduce transmission latency and improve the real-time performance of ChatIoT.

Robustness and convergence of DDPG component: We subsequently assessed the robustness and convergence of the DDPG component as follows.

In the ChatIoT scenario, the environment is defined by the maximum memory capacity and the varying sequence of user requests. During the training of the DDPG component in the cloud, we set the maximum memory capacity to support only ten models. To accommodate users with diverse resource conditions at the edge, we conducted separate evaluations to assess the robustness of our pre-trained DDPG component. We generated request traces in multiple sequences to account for diverse sequences of user requests. As depicted in Figure 12, ChatIoT consistently attains the highest accuracy across various memory constraints, showcasing the resilience of our pre-trained DDPG component in different environments.



(a) DIY Home (b) ChatIoT Demo

Fig. 17. The DIY home includes sensors (i.e., temperature sensor) and actuators (i.e., light) deployed in the DIY home, and COTS devices (i.e., surveillance camera) deployed outside the DIY home. We also developed an web GUI to visualize the workflow of ChatIoT and shows that ChatIoT can customized models and generate TAPs based on user requests.

Moreover, we evaluated the convergence of the DDPG component. The trend of rewards during training indicates that the DDPG component can converge after 100 epochs (approximately 52.2 seconds), as shown in Figure 15. The decreasing trend in the loss of the critic, presented in Figure 16, also indicates a steady decline.

5.5 Case Study

We further conducted a case study using a DIY smart home with multiple sensors, cameras, and actuators. In addition, we also developed a web demo to facilitate user interaction and intuitively obtain the results of ChatIoT. Figure 17 shows our case study environment and the GUI for users. We started by introducing our DIY home and then exemplified the convenience of ChatIoT with an end-to-end example and user study.

5.5.1 Home detail. The living room area is furnished with a diverse array of devices, encompassing a smart television, an air purifier, a centralized light fixture illuminating the entire living space, an RGB light, and a door sensor for detecting the state of the door. In addition, we use the scene captured by the camera in the real scene to simulate the scene at the entrance of the DIY home. Transitioning to the staircase, a dedicated light source is installed to provide illumination specifically for the stairs. A COTS human sensor is deployed near the DIY home to simulate a human sensor on the stairs. Moving to the study room, a light and a table lamp are present to cater to the lighting needs in that area. In the bedroom, occupants can avail themselves of a light source, a table lamp, and an air conditioner for enhanced comfort. Lastly, a temperature sensor is equipped on the bookshelf to measure the temperature in the bedroom.

5.5.2 An end-to-end example of TAP generation. The case study commences with a focused examination of a single, detailed example encompassing the workflow of the TAP generation process, model generation and deployment, and the respective time metrics along with the related accuracy for each phase. The specified user request is as follows: "If the door is opened while the camera detects a person at the door, turn on the light on the stairs." The workflow of the ChatIoT system is illustrated in Figure 18. This end-to-end example, from the identification of a service need to the deployment of actionable TAP rules, exemplifies the system's agility and its ability to provide tailored and intelligent solutions that meet the dynamic needs of its users.

Leveraging user input and the comprehensive data from existing home devices, the compressor employs cosine similarity to meticulously filter out redundant data. This sophisticated approach enables the compression of context tokens from an initial count of 5736 to a mere 329 and takes 1.93s. As depicted in Figure 18, the

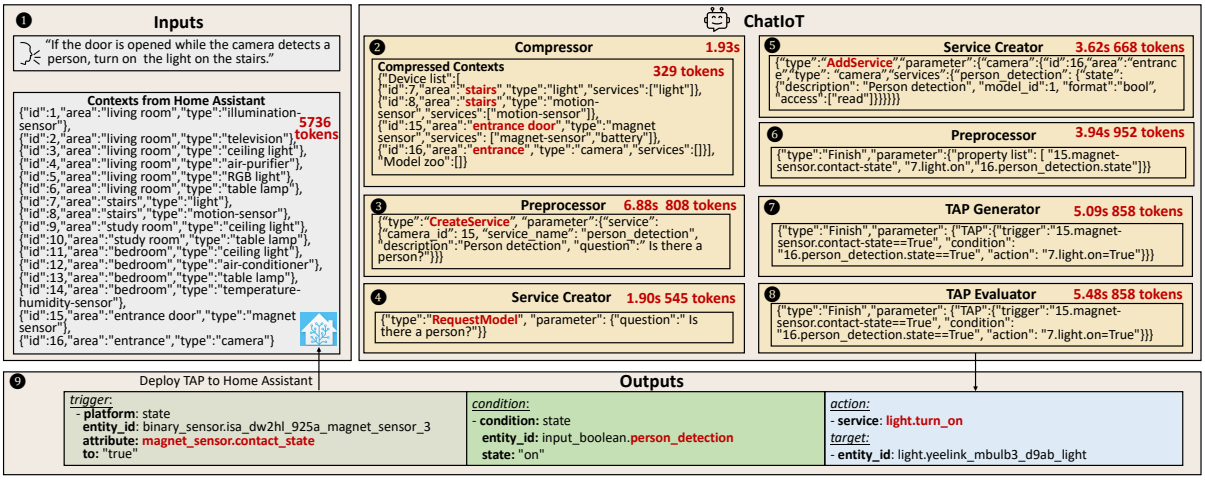


Fig. 18. An end-to-end example workflow of TAP generation.

compression algorithm selectively retains only the most pertinent information concerning specific home devices, such as the "Magnet sensor", "Surveillance Camera", and "Light on stairs".

Upon receiving the user's request alongside the compressed contexts, ChatIoT will transport the inputs to its preprocessor. Utilizing the "Thought" mechanism, the preprocessor swiftly identifies a gap in the availability of relevant models for person detection and it submits a request to the service creator. This process of analysis is efficient, consuming 6.88 seconds and 808 tokens. In the subsequent phase of the process, the service creator embarks on an initial analysis to precisely identify the requirements for the requested model which takes 1.90 seconds and consumes 545 tokens. Following this analysis, the service creator proceeds to leverage a MLLM to procure the person detection model through knowledge distillation. The generation of this model was accomplished in a span of 10 seconds, culminating in a model whose accuracy astonishingly reaches 99%. With the model successfully created, the next step executed by the service creator involves adding this new service into the current contexts which takes an additional 3.62 seconds and consumed 668 tokens. Subsequently, the preprocessor embarks on the task of generating a related property list, which takes 3.94 seconds and 946 tokens. Upon the successful generation of the necessary services and the related property list (15.magnet-sensor.contact-state, 7.light.on, 16.person_detection.state), it takes 5.09 seconds and 858 tokens to employ a LLM to generate corresponding TAP rules ("trigger": "15.magnet-sensor.contact-state==True","condition":"16.person_detection.state==True","action":"7.light.on=True").

In the final stage of the workflow, the evaluator plays a pivotal role in assessing the quality and efficacy of the generated TAP rules and it takes 5.48 seconds and requires 858 tokens. Upon successful verification, these TAP rules are deployed to the HA system, marking the culmination of a process that not only bridges the gap between user needs and system capabilities but also significantly enhances the system's functionality and user experience. Considering all processes, ChatIoT can generate a TAP that requires a customized model based on user requests after consuming 4689 tokens (about 0.0028865$ using GPT-3.5-turbo) in about 40 seconds. When the inference speed of LLM is further accelerated in the future, ChatIoT can complete TAP generation in a shorter time.

5.5.3 User study. To enhance the accessibility and user engagement with the ChatIoT, we developed a web page dedicated to visualizing the system's workflow. A user study was conducted involving 15 real users, each of whom utilized a GUI to submit TAP generation requests tailored to our DIY Home setting. Upon receiving input from the users, ChatIoT employed a series of technical measures we proposed, leading to the generation

of the corresponding TAP rules. Through this process, a total of 60 requests were collected from the 15 real users. Alongside the system-generated TAP rules, we also manually crafted accurate TAP rules based on the user requests to serve as a benchmark for evaluating the system's performance. Results derived from these requests were highly encouraging, demonstrating that ChatIoT is capable of achieving a TAP generation accuracy of 91.57%. Additionally, the average token consumption for generating one TAP rule is about 4093 tokens (0.0026$ using GPT-3.5-turbo).

6 Conclusion and Future Work

In this paper, we present ChatIoT, a zero-code TAP generation system based on LLMs. Major contributions of ChatIoT include context-aware compressive prompting and DRL-based model customization. With ChatIoT, a user can request a new TAP using purely natural language, and leave the TAP generation and deployment to ChatIoT. Further, when the user request includes the requirements of new sensing abilities such as a new "visitor sensor" based on the camera, ChatIoT can efficiently generate the required AI model by knowledge distillation from the MLLM, achieving out-of-box TAP generation. ChatIoT is implemented with ChatGPT, VisualGLM, and Home Assistant. Extensive evaluations show that ChatIoT achieves high TAP generation accuracy in a token-efficient manner in various settings.

There are multiple possible directions for future work. First, besides Home Assistant, it is interesting to extend ChatIoT to more TAP runtimes, which may also enables more application scenarios beyond home automation. Second, as mentioned in the related work section, combining trace-driven TAP recommendation approaches and ChatIoT could further improve the user experience. Third, since some actuators could be controlled in a more fine-grained way (e.g., opening speed of a smart curtain), extending the abilities of fine-grained control is another future direction of ChatIoT.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 62272407 and 62072396, the "Pioneer" and "Leading Goose" R&D Program of Zhejiang under grant No. 2023C01033, and the National Youth Talent Support Program. The work is also supported by Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.

References

[1] 2023. Amazon Alexa. https://alexa.amazon.com.
[2] 2023. Apple Siri. https://www.apple.com/siri/.
[3] 2023. AppleHomeKit. https://www.apple.com/home-app/.
[4] 2023. ChatGPT. https://openai.com/chatgpt.
[5] 2023. Claude. https://claude.ai/.
[6] 2023. Google Assistant. https://assistant.google.com.
[7] 2023. GPT4. https://openai.com/research/gpt-4.
[8] 2023. Home Assistant. https://www.home-assistant.io/docs/automation/.
[9] 2023. Mijia. https://home.mi.com.
[10] 2023. python-mimo. https://github.com/rytilahti/python-miio.
[11] 2024. MiniCPM-V. https://github.com/OpenBMB/MiniCPM-V.
[12] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. 2022. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836* (2022).
[13] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17682–17690.
[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33

(2020), 1877–1901.
[15] Xuyang Chen, Xiaolu Zhang, Michael Elliot, Xiaoyin Wang, and Feng Wang. 2022. Fix the leaking tap: A survey of Trigger-Action Programming (TAP) security issues, detection techniques and solutions. *Computers & Security* 120 (2022), 102812.
[16] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).
[17] Jang Hyun Cho and Bharath Hariharan. 2019. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*. 4794–4802.
[18] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review* 53 (2020), 5113–5155.
[19] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2021. From users' intentions to if-then rules in the internet of things. *ACM Transactions on Information Systems (TOIS)* 39, 4 (2021), 1–33.
[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[21] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.
[22] Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. 2021. Cogview: Mastering text-to-image generation via transformers. *Advances in Neural Information Processing Systems* 34 (2021), 19822–19835.
[23] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 320–335.
[24] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
[25] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320* (2020).
[26] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129 (2021), 1789–1819.
[27] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints* (2023).
[28] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints* (2023).
[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[30] Xiaoxi He, Zimu Zhou, and Lothar Thiele. 2018. Multi-task zipping via layer-wise neuron sharing. *Advances in Neural Information Processing Systems* 31 (2018).
[31] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*. 1389–1397.
[32] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE, 2366–2369.
[33] Zhenbo Hu, Xiangyu Zou, Wen Xia, Sian Jin, Dingwen Tao, Yang Liu, Weizhe Zhang, and Zheng Zhang. 2020. Delta-DNN: Efficiently compressing deep neural networks via exploiting floats similarity. In *Proceedings of the 49th International Conference on Parallel Processing*. 1–12.
[34] Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. 2022. Repair: Renormalizing permuted activations for interpolation repair. *arXiv preprint arXiv:2211.08403* (2022).
[35] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).
[36] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459* (2024).
[37] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
[38] Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. 2020. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253* (2020).

[39] Liwei Liu, Wei Chen, Tao Wang, Wei Wang, Guoquan Wu, and Jun Wei. 2023. Generating Scenario-Centric TAP Rules for Smart Homes by Mining Historical Event Logs. In *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, 21–27.
[40] Xiaoyi Liu, Yingtian Shi, Chun Yu, Cheng Gao, Tianao Yang, Chen Liang, and Yuanchun Shi. 2023. Understanding In-Situ Programming for Smart Home Automation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 7, 2 (2023), 1–31.
[41] Amama Mahmood, Junxiang Wang, Bingsheng Yao, Dakuo Wang, and Chien-Ming Huang. 2023. LLM-Powered Conversational Voice Assistants: Interaction Patterns, Opportunities, Challenges, and Design Guidelines. *arXiv preprint arXiv:2309.13879* (2023).
[42] Marco Manca, Fabio Paternò, Carmen Santoro, and Luca Corcella. 2019. Supporting end-user debugging of trigger-action rules for IoT applications. *International Journal of Human-Computer Studies* 123 (2019), 56–69.
[43] Andrea Mattioli and Fabio Paternò. 2023. A Mobile Augmented Reality App for Creating, Controlling, Recommending Automations in Smart Homes. *Proceedings of the ACM on Human-Computer Interaction* 7, MHCI (2023), 1–22.
[44] Mark D McDonnell. 2018. Training wide residual networks for deployment using a single bit for each weight. *arXiv preprint arXiv:1802.08530* (2018).
[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[46] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. 2023. Gemel: Model Merging for {Memory-Efficient},{Real-Time} Video Analytics at the Edge. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 973–994.
[47] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084
[48] Emmanuel Senft, Michael Hagenow, Robert Radwin, Michael Zinn, Michael Gleicher, and Bilge Mutlu. 2021. Situated live programming for human-robot collaboration. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 613–625.
[49] George Stoica, Daniel Bolya, Jakob Bjorner, Taylor Hearn, and Judy Hoffman. 2023. Zipit! merging models from different tasks without training. *arXiv preprint arXiv:2305.03053* (2023).
[50] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]
[51] Hado Van Hasselt and Marco A Wiering. 2009. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*. IEEE, 1149–1156.
[52] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 1–26.
[53] Lin Wang and Kuk-Jin Yoon. 2021. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE transactions on pattern analysis and machine intelligence* 44, 6 (2021), 3048–3068.
[54] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).
[55] Yuntao Wang, Yanghe Pan, Miao Yan, Zhou Su, and Tom H Luan. 2023. A survey on ChatGPT: AI-generated contents, challenges, and solutions. *IEEE Open Journal of the Computer Society* (2023).
[56] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
[57] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).
[58] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2024).
[59] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).
[60] Imam Nur Bani Yusuf, Lingxiao Jiang, and David Lo. 2022. Accurate generation of trigger-action programs with domain-adapted sequence-to-sequence learning. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 99–110.
[61] Lefan Zhang, Weijia He, Olivia Morkved, Valerie Zhao, Michael L Littman, Shan Lu, and Blase Ur. 2020. Trace2tap: Synthesizing trigger-action programs from traces of behavior. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3 (2020), 1–26.
[62] Lefan Zhang, Cyrus Zhou, Michael L Littman, Blase Ur, and Shan Lu. 2023. Helping Users Debug Trigger-Action Programs. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 4 (2023), 1–32.
[63] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493* (2022).

[64] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[65] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625* (2022).

[66] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. 2018. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.