JMST Advances
https://doi.org/10.1007/s42791-024-00085-x

Online ISSN 2524-7913
Print ISSN 2524-7905

LETTERS

Large language model based collaborative robot system for daily task assistance

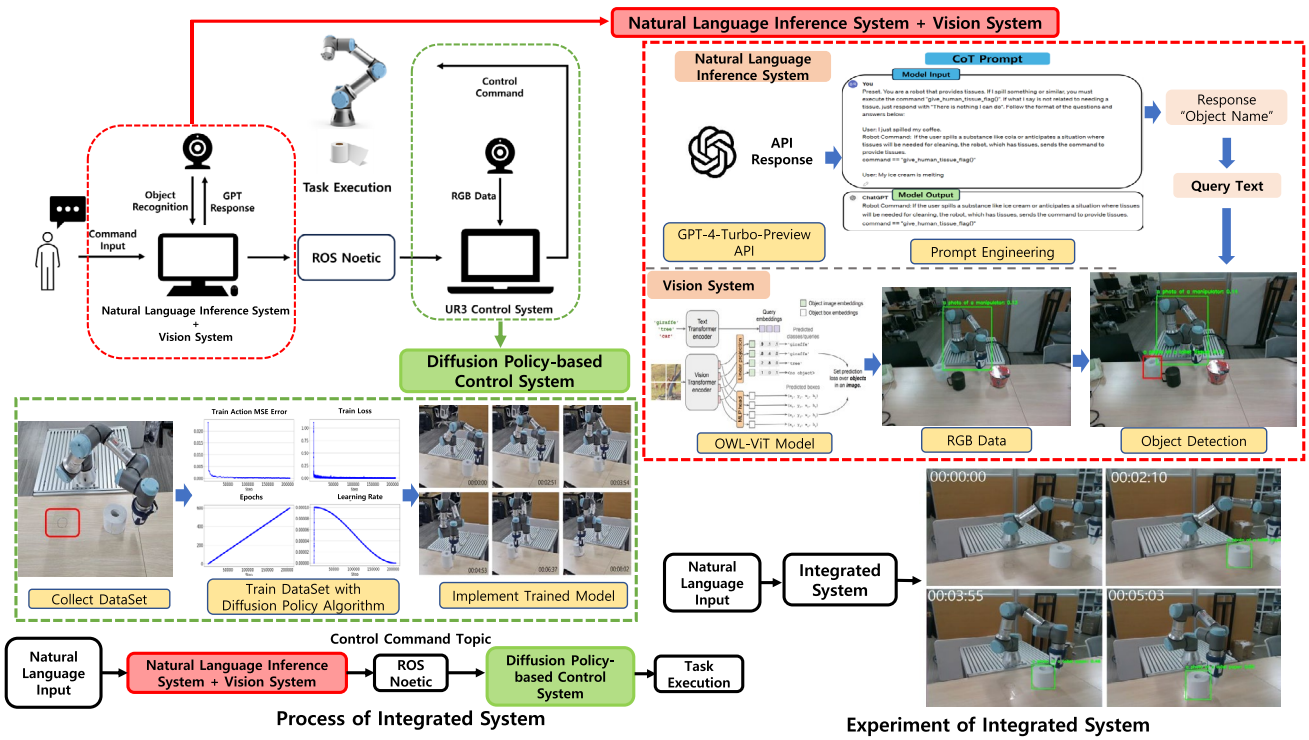
Seunguk Choi[1] · **David Kim**[1] · **Myeonggyun Ahn**[1] · **Dongil Choi**[1]

Received: 7 June 2024 / Revised: 21 July 2024 / Accepted: 22 July 2024

Abstract

The recent advancements in natural language processing models have spurred extensive research into robots that interact with and execute tasks based on human natural language commands. This study introduces a robot system that interprets human commands using a large language model (LLM) to understand the intent and relay it to a robot for task execution. The robot system comprises three key components: natural language command inference, vision, and learning-based control systems. The command inference system utilizes the GPT-4 model to interpret natural language commands and issue task execution orders to the robot. Based on the transformer model OWL-ViT, the vision system recognizes objects related to user commands and communicates the findings to the inference system. Upon understanding the user's commands, a UR3 robot executes the tasks using the Diffusion Policy. The robot system is integrated using ROS (robot operating system) and validated through experiments. Through the experiment, a success rate of 65% was achieved.

Graphical abstract



Keywords Large language model · Learning-based control · Vision transformer

Extended author information available on the last page of the article

Published online: 12 August 2024

List of symbols

Q	Query value
K	Key value
V	Value
W	Attention score
b	Weight value
A_k^t	Robot's action data
$\sigma^2 I$	Gaussian noise variance
O_t	Environmental data
k	Iteration step count
ε_θ	Noise prediction network
γ	Noise prediction network decay factor
α	Denoising decay factor

1 Introduction

1.1 Background of the study

The development of large language model (LLM) has been progressing rapidly, with increasing applications in the robotics field. LLM have evolved naturally due to advancements in deep learning and artificial neural networks, enabling the training on massive datasets. Notably, with the advent of transformers [1], the need for sequential processing as seen in traditional RNNs or LSTMs [2] has been eliminated, allowing for parallel processing of large datasets, exemplified by LLM like BERT [3] and the GPT series [4]. These models have made interpreting human natural language intents and inferring context feasible. However, generating user-specific responses remains challenging, potentially leading to responses that hinder robot task performance due to misinterpretation of user intent.

Recently, robots employing large language models, such as Tidybot [5], RT-2, and NICOL [6], have been introduced. These robots interpret human natural language commands through LLM and execute predefined actions. Tidybot was developed as a robot for household cleaning. Before starting the cleaning process, the robot is provided with several examples of specific objects, which are then fed into the LLM to categorize objects and infer their rearrangement. It performs two predefined actions, Pick and Place and Pick and Toss, to organize items according to user preferences.

However, since Tidybot focuses on the categorization capabilities of the LLM, there is an issue where the predefined actions appear unnatural when the robot performs complex tasks from a control perspective. Additionally, adding new actions to the robot requires a process of collecting new data and solving high-cost problems.

1.2 Purpose of the study

The robotic system proposed in this study aims to understand human natural language commands and perform tasks using the UR3 robot. The system integrates a natural language inference system to interpret user commands, a vision system to identify objects related to these commands using visual data, and a control system to execute the tasks when feasible. This integration is achieved through ROS (robot operating system), enabling seamless task execution.

The research focuses on developing a system that infers human natural language, recognizes objects, and commands the robot accordingly. GPT-4 is employed as the large language model to infer user commands, and the Transformer-based OWL-ViT model is used to recognize objects related to the commands, with computations performed on a host PC. Both GPT-4 and OWL-ViT utilize pretrained models via APIs, with GPT-4 API incorporating CoT (chain of thought) [7] to understand user intentions.

The goal is to execute the inferred tasks using the UR3 robot with learning-based control, leveraging the commands processed by the large language model. The entire system is integrated through ROS to ensure cohesive and efficient task execution.

2 Natural language command inference system

2.1 Large language model GPT-4-Turbo

The GPT-4 model is a large language model released by OpenAI. Like most large language models, the GPT model is based on the transformer architecture. While the transformer architecture consists of an encoder–decoder structure, the GPT series models are constructed using only the decoder part. Through the decoder, GPT predicts the next word and understands the context of the input sequence by considering the relationships between words using the self-attention mechanism. In this study, we constructed a command inference system using the GPT-4-Turbo-preview model from the GPT series.

Detailed information about the training methods and the number of layers of the GPT-4 model is not disclosed, but it is known to operate similarly to GPT-3. The GPT-3 model consists of two main components: the multi-head self-attention mechanism and position-wise feedforward neural networks [8]. Each layer contains these two components, and GPT-3 has a total of 96 layers. The model needs the ability to understand contextual information regarding

the continuous context given as input. This capability is enabled by simultaneously considering the information from different positions in the sequence through the multi-head self-attention mechanism. When a sentence is input into the context text, the model splits the sentence into individual tokens and converts them into high-dimensional vectors. These converted vectors are multiplied by weight matrices and then re-converted into keys, queries, and values. Here, the query is the vector of the currently focused word, the key evaluates the relationship with other words, and the value is used to calculate the output based on the input. The mechanism calculates the attention score as shown in Eq. (1) to represent the relationships of each word in the context. Then, the attention score is applied to the position-wise feedforward neural network, as shown in Eq. (2). This process enhances the model's expressive power while processing each position independently.

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\beta_0 \tag{1}$$

$$\text{AFFN}(x) = \max\left(0, xW_1 + b_1\right)W_2 + b_2 \tag{2}$$

Here, $max\left(0, xW_1 + b_1\right)$ is the ReLU (rectified linear unit) activation function, which allows the learning of non-linear complex patterns and removes negative values. The final output vector is generated through these two main components, enabling the model to understand contextual information and identify essential patterns in natural language processing to infer the user's command intent and generate appropriate responses.

The GPT-4-Turbo-preview model can handle a maximum of 128,000 tokens and the training data include information up to December 2023. Tokens represent the number of characters the model can process at one time and a higher number allows the understanding and processing of more complex commands or questions. In this study, the command inference system was constructed using the GPT-4-Turbo-preview API to process user commands.

2.2 Prompt engineering

Prompt engineering is a technique that provides a large language model with the user's intent in advance to induce the generation of optimal responses. Using one of the prompt engineering techniques, chain-of-thought (CoT), the performance of the GPT-4 model was enhanced. CoT prompt technology explicitly provides intermediate reasoning steps to the language model to help solve complex problems. As shown in Fig. 1, the model's responses differ when using a general prompt versus CoT. The primary reasons for different responses to the same question are twofold: reducing errors in the reasoning process by learning problem-solving methods through providing reasoning steps to the model and enabling Few-Shot [9] learning to allow the model to infer correct answers.

In the command inference system of this paper, prompt such as Fig. 2 were structured into a Pre-Prompt.txt file and provided to the GPT-4 model in advance. This allows the system to infer user input and generate flags for task execution. A total of 20 examples were created, consisting of 10 scenarios where command execution is possible and 10 scenarios where it is not. Through these structured examples, the system generates responses and performs control commands via ROS.

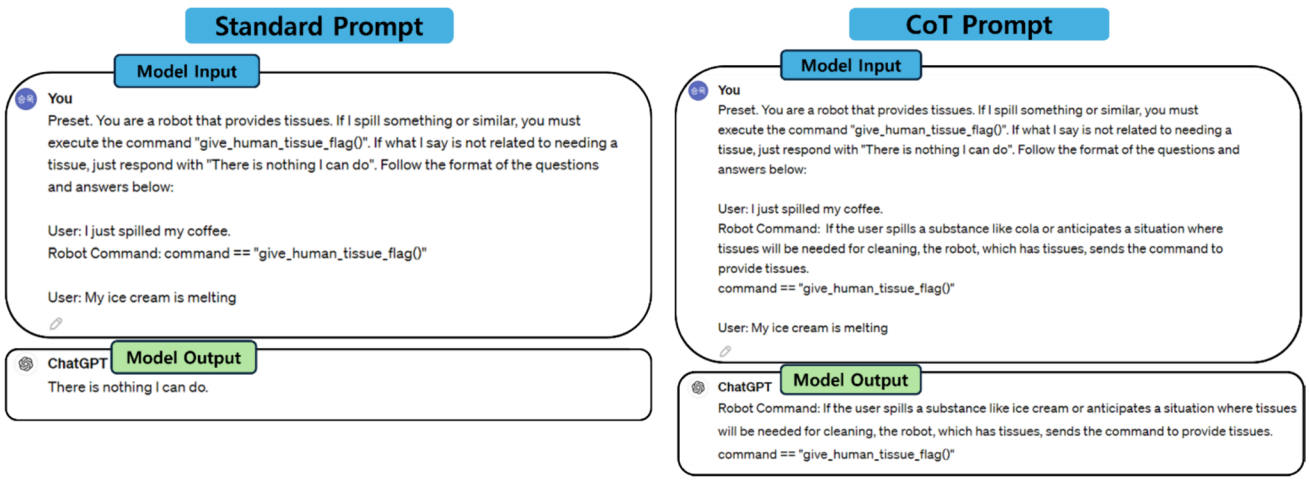


Fig. 1 Standard prompt (left), chain of thought prompt (right)

When a user mentions a situation that requires a tissue, the system should set the variable give_human_tissue_flag to 1, indicating that the robot should provide a tissue. Additionally, if the user input is empty or only one character, the system should not respond.

Chain of Thought:

1. Analyze the user's request to determine if a tissue is needed.
2. Detect keywords such as "spilled," "need something to clean," "need a tissue," etc.
3. If these keywords are detected, it means the user needs a tissue.
4. In this case, set the give_human_tissue_flag variable to 1.
5. Return the set value to instruct the robot to provide a tissue.

Example Situation 1:
User Input: "I spilled something and need something to clean it up."
Analysis:

1. The user mentioned "spilled something."
2. The user mentioned "need something to clean."
3. Both keywords are detected.
4. Therefore, the user needs a tissue.
5. Set give_human_tissue_flag to 1.

Response:
give_human_tissue_flag=1

Example Situation 2
:User Input: "Water overflowed on the table. I need to clean it with something."
Analysis:

1. The user mentioned "water overflowed.
2. "The user mentioned "need to clean it with something."
3. Both keywords are detected.
4. Therefore, the user needs a tissue.
5. Set give_human_tissue_flag to 1.

Response:
give_human_tissue_flag=1

Fig. 2 Example of prompt for command inference

2.3 GPT-4-Turbo-Preview API

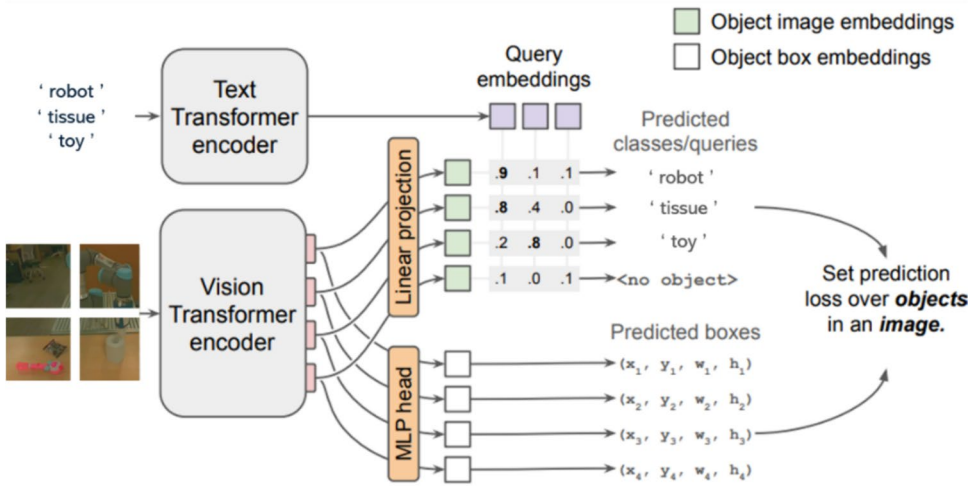
To use the GPT-Turbo-Preview model, a paid API key must be obtained from OpenAI. The method for using the API involves installing the OpenAI library, registering the API key, and paying based on the length of the used tokens. After connecting to the host server, the raw format response of the model is as follows. The API call returns various information, including the unique identifier ID, model information, and token count. The information we will use is the model's response, denoted as Text. In string format, this text will be input into the query of OWL-ViT, explained in the next section, and integrated with the vision system.

3 Vision transformer-based vision system

3.1 OWL-ViT model

In this study, we employed the OWL-ViT (open world localization using vision transformers) model, a type of Open-Vocabulary Object Detection (OVOD) model, for the vision system. OWL-ViT is based on the Vision Transformer architecture and is particularly strong in zero-shot learning [10]. OVOD models possess the capability to recognize and detect new objects that were not included in the training data. Most OVOD models, including OWL-ViT, rely on CLIP (Contrastive Language-Image Pre-Training), trained with text-image pairs, enabling them to understand complex textual descriptions and recognize corresponding objects.

Fig. 3 OWL-ViT architecture



The architecture of OWL-ViT, as shown in Fig. 3, consists of text and image encoders. Similar to how the GPT model processes text, the text data are divided into individual words using a tokenizer. These words are then converted into embedding vectors and processed through the transformer encoder. The input RGB data are divided into 16×16 patches for image encoding, then converted into one-dimensional vectors with positional encoding added. These vectors, including the positional information, are sent to the multilayer perceptron head. The similarity between the resulting text and image embedding vectors is calculated, and positional information is matched for object recognition.

3.2 Application of the OWL-ViT model

To apply OWL-ViT to the system, the model must be loaded through HuggingFace's Transformers library. After loading the model and its related configurations, the RGB data are converted into a tensor format for application. The objects to be recognized are set as text query values, tokenized, and then provided as input to the model. When the model was initially applied on a CPU, high computational demands resulted in low frame rates. The model was executed on a GPU to address this issue, significantly enhancing performance by approximately tenfold, as shown in Fig. 4. Although using a GPU increased frame rate, it also required substantial resources. Therefore, in the integrated system, the command inference system and vision system were run on a single PC, while the learning-based robot control algorithm was executed on a separate PC, with all components integrated using ROS.

3.3 Application of the vision system in the command inference system

Before the robot can execute tasks based on natural language commands from the user, it must verify the presence of necessary objects for task execution. This process involves sending up to ten object names related to the command from GPT-4 to the OWL-ViT as text query values for detection. OWL-ViT performs object recognition based on text queries, identifying the most accurate objects. Multithreading was employed to separate the threads and integrate both models into the system. When receiving responses from GPT-4, a queue operates to store the responses in the texts variable as queries. The list format values are updated each time an input from the GPT-4 model is received. The robot will recognize only itself if no relevant objects are identified in the response. As shown in Fig. 5, when the response from GPT-4 includes "Tissue," it is entered as a text query, and OWL-ViT detects and signals the object's presence. The right image with the red box indicates successful detection of "Tissue."

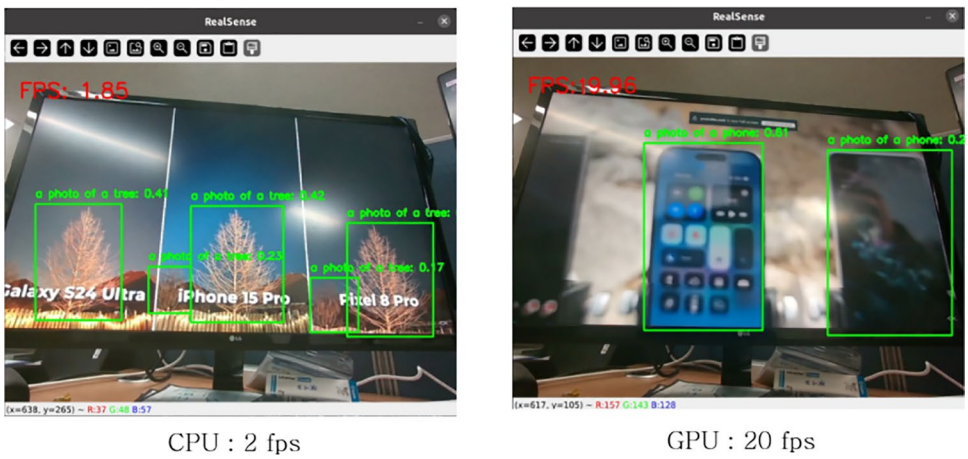


Fig. 4 Frame comparison using CPU (left) and GPU (right)

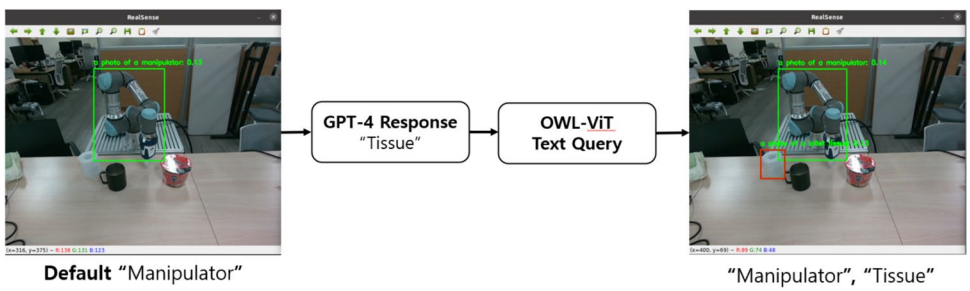


Fig. 5 GPT-4 response to OWL-ViT

4 Learning-based robot control system

4.1 Learning based control

Unlike model-based control, learning-based control involves adjusting and optimizing the system's operations using machine learning techniques. The primary methods used in learning-based control include reinforcement learning, supervised learning, and unsupervised learning. This paper focuses on applying supervised learning within the learning-based control framework. Supervised learning involves collecting data from the actual control system and training algorithms to control the system. The advantage of supervised learning is that, by leveraging extensive data, it can perform precise control even for complex or detailed tasks.

4.2 Diffusion policy algorithm

The supervised learning model employed in this study is the diffusion policy algorithm [11]. The diffusion policy algorithm is a supervised learning model developed by applying the approach of denoising diffusion probabilistic models (DDPMs) to learning policies. The basic concept of the diffusion model is to predict clean data x^0 from noisy data x^k. The process consists of the forward process, which creates x^k by adding Gaussian noise to the clean data x^0 at each of the K steps, and the reverse process, which predicts x^0 by reversing the noise addition process.

$$A_{k-1}^t = \alpha(A_k^t - \gamma\varepsilon_\theta\left(O_t, A_k^t, k\right) + N(0, \sigma^2 I)) \tag{3}$$

The policy equation for the diffusion policy algorithm is given in Eq. (3). The input data includes the Observation Space, containing the robot's information, and the output data consists of the robot's Action. At the current time step t, the robot's output data, Action, at the k^{th} step of the diffusion process is computed by subtracting the U-NET[12] based noise prediction network value, scaled by the network decay factor, and adding Gaussian noise to obtain the Action value at the $(\text{k}-1)^{\text{th}}$ step. Despite the process aiming to produce clean data, Gaussian noise is added to prevent model overfitting and ensure training stability.

The loss function is described by Eq. (4). It starts by randomly selecting data x_0 without added noise from the dataset. For each sample, a denoising step k is randomly chosen, and random noise ε_k with appropriate variance for that step is sampled. ε_k represents the noise added to the actual x_0. The predicted x_0 by the noise prediction network at step k is obtained from $\varepsilon_\theta(x_0 + \varepsilon_k, k)$. The Loss Function is defined using the Mean Squared Error between these values.

$$\text{Loss} = \text{MSE}(\varepsilon_k, \varepsilon_\theta(x_0 + \varepsilon_k, k)) \tag{4}$$

4.3 Virtual environment configuration for diffusion policy algorithm control

Data collection and training were conducted in a virtual environment for experimentation and validation to train the diffusion policy algorithm. The virtual environment was constructed using the Robosuite library, which employs the MuJoCo (multi-joint dynamics with contact) simulator. MuJoCo, an open-source software owned by DeepMind, is widely used for virtual environment experiments in robotics, reinforcement learning, and dynamic control. The virtual environment setup is shown in Fig. 6.

The manipulator used in the simulation environment was the Panda from Franka Robotics. The task was to move a can to the red box area on the right side of Fig. 6. The end-effector used was the Franka Hand from the same company.

To control the Panda manipulator in the simulation environment, a Spacemouse from 3Dconnexion (Fig. 7) was used. The Spacemouse is typically used with 3D modeling software such as CAD programs and can implement movement in 3D space with a 6-degree-of-freedom sensor. It provides control inputs to the manipulator with three translational movements (x, y, z) and three rotational movements (roll, pitch, yaw). The detailed control input values of the Spacemouse are summarized in Table 1.

Operational space control (OSC) is the control method for the manipulator using the Spacemouse in the simulation environment. Since the Robosuite library supports OSC, it is straightforward to use. OSC calculates the required joint torques to minimize the difference between the current and target poses with minimal kinetic energy. The control frame is defined in the space between the gripper fingers, and the joint torque commands are mapped by calculating the proportional-derivative gain vector between position and orientation differences.

4.4 Data collection method in virtual environment

In the virtual environment, the manipulator was directly controlled by a human using the Spacemouse to perform pick-and-place tasks. Data collection concluded when the manipulator successfully picked up the can and placed it in the designated red area shown in Fig. 6. During this demonstration, the robot's state and control inputs were collected as input–output data and stored in HDF5 format.

HDF5 (hierarchical data format version 5) is a file format designed for storing large amounts of data. The file consists of groups, attributes, datasets, and metadata, enabling a hierarchical structure within the file system. Specifically, attributes provide additional information about datasets and groups in key-value pairs. Groups contain datasets, where the collected data are stored, and metadata exists for the file, groups, and datasets to describe their respective information.

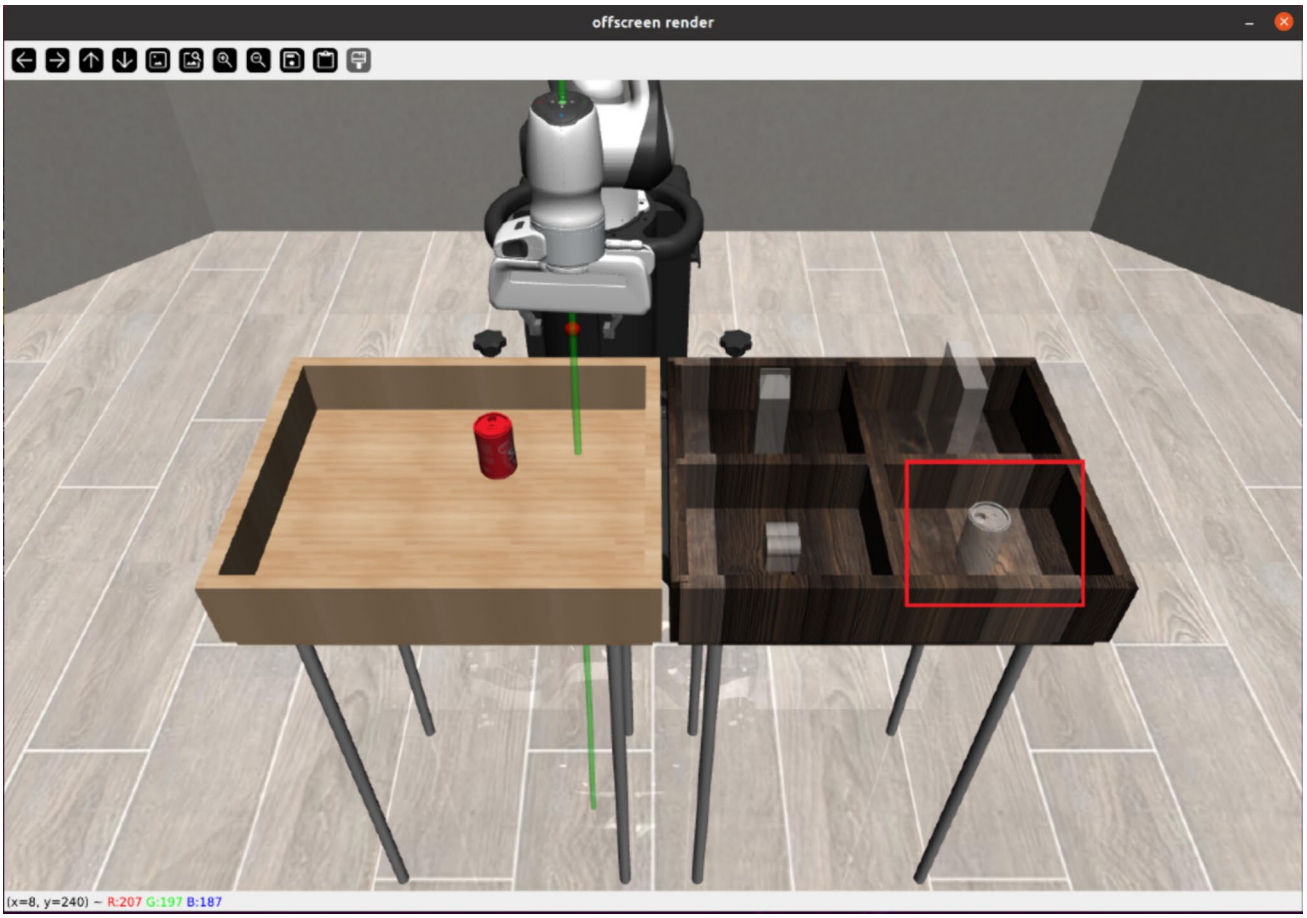


Fig. 6 MuJoCo simulator environment

Fig. 7 3Dconnexion Spacemouse wireless model

Table 1 Spacemouse controls to Panda robot and gripper

Control	Command
Right button	Reset simulation
Left button (hold)	Close gripper
Move mouse laterally	Move arm horizontally in x–y plane
Move mouse vertically	Move arm vertically

A total of 200 demonstrations were conducted and the collected data are summarized in Table 2.

4.5 Training data collection in virtual environments

The collected data was used to train the diffusion policy algorithm. The input data, robot state data, was fed into O_t and the output data, control commands from the Spacemouse, was fed into A^t for the denoising process. RGB data were processed through a noise scheduler and cropped to 76×76 pixels. A Resnet18 neural network, comprising 18 layers, was employed for image processing. Resnet18 uses

Table 2 Collected data lists

Data	Shape
Object with shape	(118, 14)
End effector pose	(118, 3)
End effector quaternion	(118, 4)
End effector velocity angle	(118, 3)
End effector velocity linear	(118, 3)
Gripper qpos	(118, 2)
Gripper qvel	(118, 2)
Joint pose	(118, 7)
Joint velocity	(118, 7)

Residual Learning with Skip Connections to address gradient vanishing issues. The noise prediction network employed a U-NET structure, which upscales data dimensions to 256, 512, and 1024, then downscales them again. This process has the advantage of minimizing data loss related to feature points by linking the previously lost information within the same dimension.

The training parameters were set as follows: 2000 epochs, AdamW optimizer, and a batch size 256. The WanB platform was used to monitor training progress and results. WanB supports various machine learning frameworks such as PyTorch, TensorFlow, and Keras. This feature provides intuitive results to the user by supporting the tracking of training progress and the training environment.

4.6 Training results

Figure 8 shows the training results. The training loss converged to zero, indicating proper training. Although the validation loss deviated slightly to around 0.1, it showed convergence, suggesting robustness to new data.

The trained model was then applied in the virtual environment, successfully completing the task of moving the can, as shown in Fig. 9. Out of 100 evaluations, 79 tasks were successfully completed. For comparison, the same dataset was used to train another algorithm, IBC (implicit behavior cloning), which learns behavior distribution using the maximum entropy principle. The IBC algorithm achieved a 40% success rate, demonstrating that the diffusion policy algorithm performed better.

5 System integration and experiment *in real* environment

5.1 Experimental setup and data training

The three systems described earlier were integrated and tested in a real-world environment. The hardware configuration used in the experiments is as follows: Intel's RealSense D435 was employed for the command inference system, and the vision system and the RealSense D455 were used for the learning-based control system. Thus, two cameras were utilized. The data format sent from the cameras to the system is RGB data. Detailed camera specifications are summarized in Table 3.

The manipulator used was the UR3 from Universal Robots. The UR3 is a robot with six degrees of freedom that is widely utilized in manufacturing, education, and other fields. Due to its flexibility and precision, the UR3 is particularly advantageous in small workspaces. Detailed specifications are summarized in Table 4.

5.2 UR3 control dataset collection and training

New real-environment data had to be collected to apply the diffusion policy algorithm to the UR3 manipulator. The goal

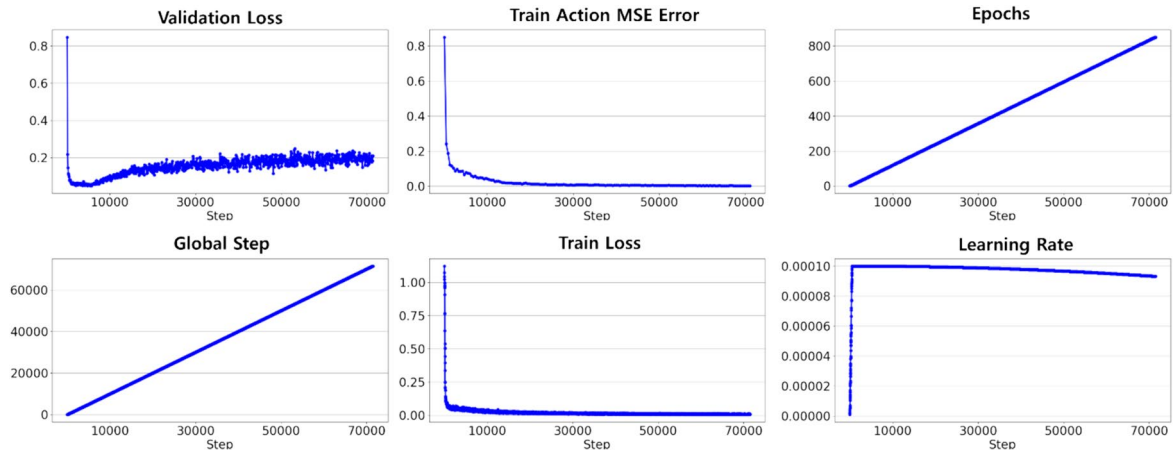


Fig. 8 Result graph of training

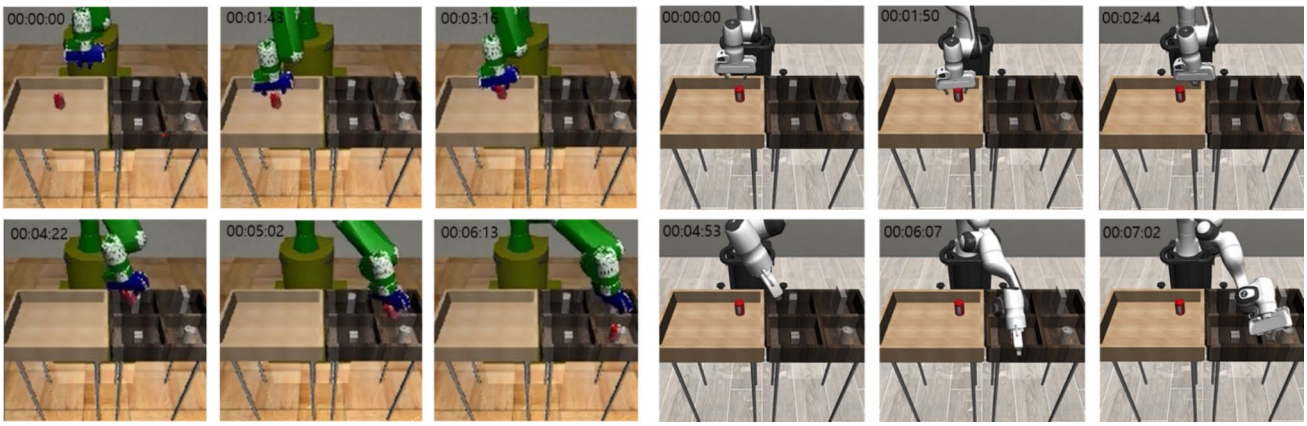


Fig. 9 Diffusion policy algorithm (left) and IBC algorithm (right)

Table 3 Intel Realsense D435 and D455 specification

Parameter	Camera sensor properties	
Camera name	D435	D455
Outer dimension	90 mm, 25 mm, 25 mm (W, L, H)	124 mm, 29 mm, 26 mm (W, L, H)
Image sensor	OmniVision Technologies OV9282	
Active pixels	1280 × 800	
Sensor aspect ratio	8:5	
Format	10-bit Raw RGB	

Table 4 Universal robots UR3 specification

Degree of freedom	6 DOF
Payload	3 kg
Maximum reach	500 mm
Joint position limits	A1, A2, A3, A4, A5: ± 360°, A6: infinite
Joint velocity limits	A1, A2, A3: ± 180°/s, A4, A5, A6: ± 360°/s
Power consumption	300 W
Moderate operating setting	100 W

was to move a tissue to a red area. During data collection, the manipulator was controlled using a Spacemouse. Movements were limited to the X–Y plane relative to the robot's end effector. A total of 200 demonstrations of moving the tissue to the target area were conducted, with the data collected in the same format as the simulation environment.

The training was conducted using a workstation. The parameters set for training were epochs = 600, batch size = 64, and weight decay = 1.0 e−6. The training was monitored using the WanB platform. As shown in Fig. 10, the results indicated that both the training loss and validation loss converged to zero, confirming that the training process was effective.

The trained model was saved as a pt file and applied to the actual UR3, where it performed tasks correctly. As shown in Fig. 11, the UR3 robot accurately moved the tissue to the target area, achieving 38 successes out of 40 trials. Additionally, the system demonstrated effective control even when the tissue was placed alongside other objects.

To verify the ability to move objects from various positions, an experiment was conducted where a tissue was placed at random locations and moved. The results are shown in Table 5.

5.3 Hardware configuration in the integrated system

The hardware configuration in the real-world environment is illustrated in Fig. 12. The setup includes a Host PC for the command inference and vision systems, which also issues task execution commands to the robot. A Client PC, used for robot control with the Diffusion Policy applied, was also included. Both PCs were connected to RealSense D435 and D455 cameras, respectively, to receive RGB data. Communication between the PCs was facilitated using ROS

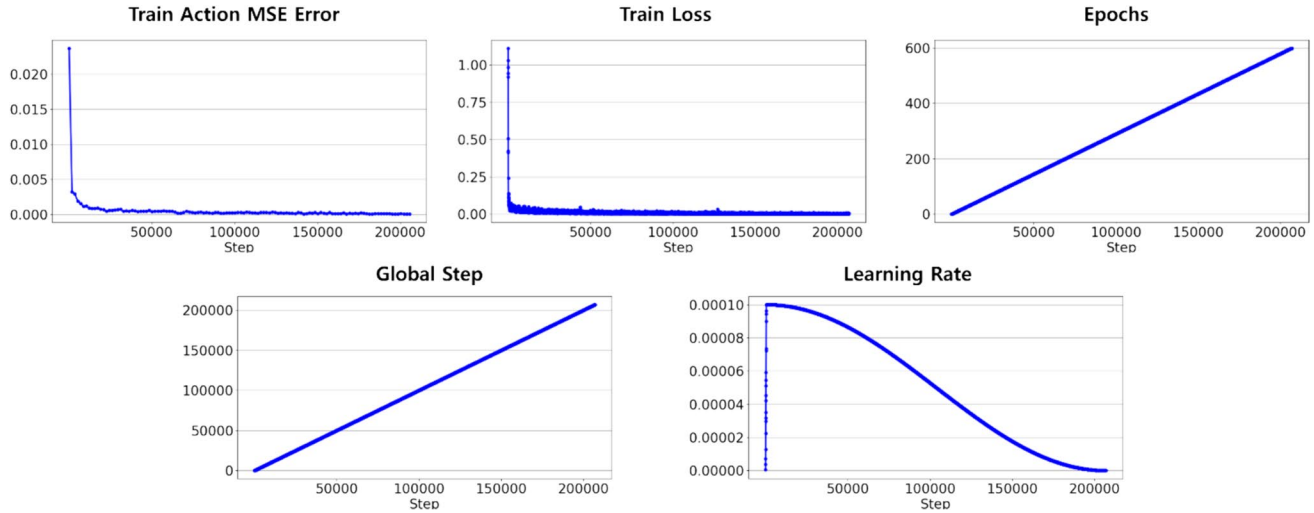


Fig. 10 Result graph of training (UR3)

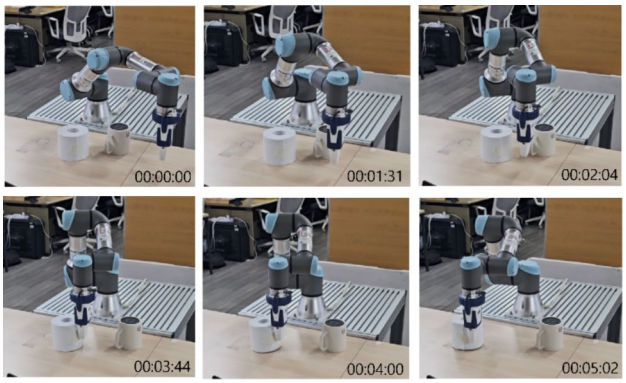


Fig. 11 Trained model applied on UR3

Table 5 Result of moving object from random location

Number of tests	40
Success	32
Fail	8

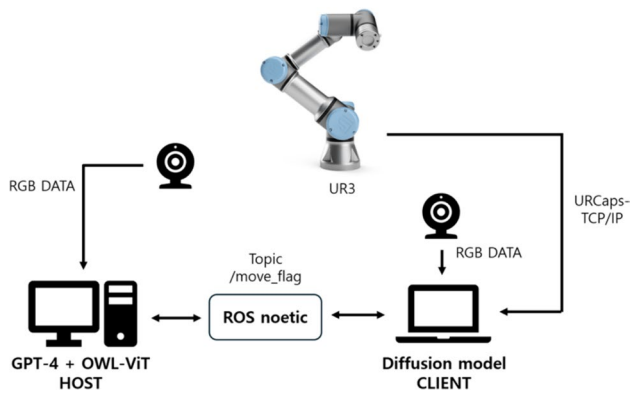


Fig. 12 Overview of Integrated System

noetic, which publishes topics to send control commands. For safety, a Spacemouse and an Emergency Stop button were prepared to allow manual control of the UR3 if necessary. The UR3 and the Client PC were connected via TCP/IP communication.

ROS (robot operating system) is an open-source framework for developing robot software across various platforms. The system configuration is as follows: when the user inputs a natural language command, the command inference system sends a query to the vision system to check for relevant objects. The vision system then relays the presence or absence of the objects back to the command inference system, which subsequently sends task execution commands to the robot control system. Based on these commands, the robot control system initiates the control of the UR3 robot.

5.4 Integrated system experiments

Experiments were conducted in a real-world environment with an integrated system. When the user issued the natural language command, “Oh no, I spilled coke,” the command inference system identified that the user needed something to clean up the spilled coke and relayed this query to the vision system. The vision system successfully performed object recognition based on the given query and communicated the results to the command inference system. The Host PC then published a topic to send the control command to the Client PC for task execution. The Client PC received the topic and issued control commands to the UR3 to move the tissue to the target area. This process is illustrated in Figs. 13 and 14, showing the Host PC screen and the recorded screen from a separate camera over time.

Additional experiments were conducted using various commands. A total of 20 different commands were given

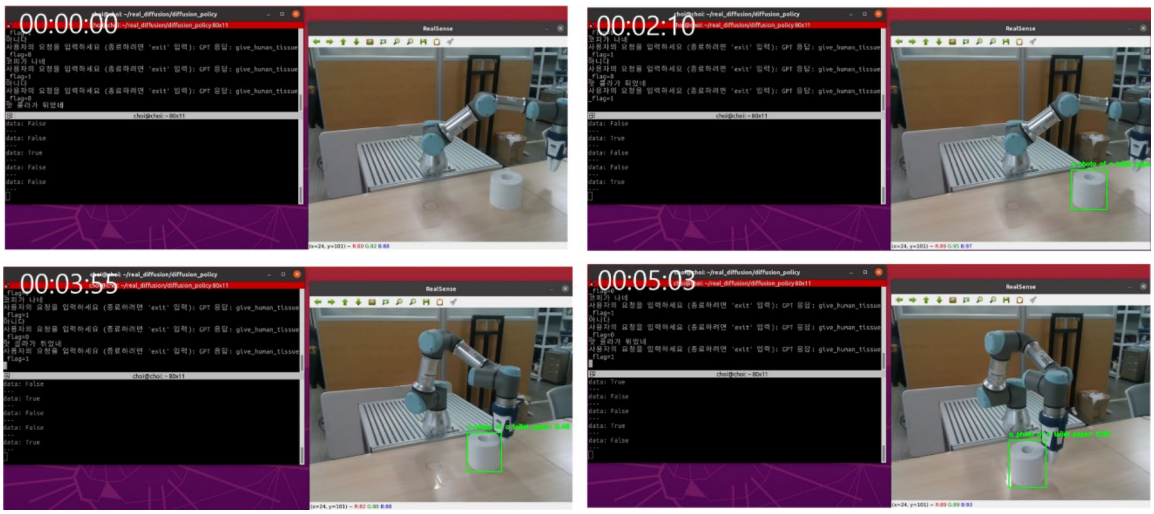


Fig. 13 System experiment, host PC screen

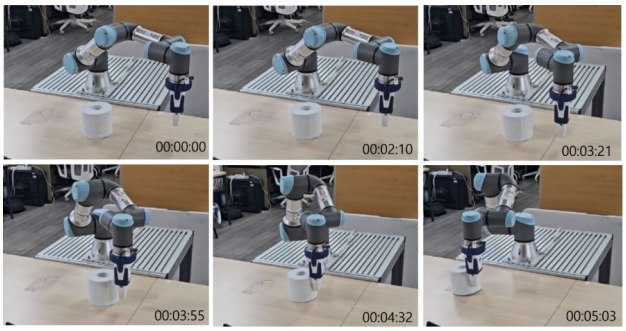


Fig. 14 System experiment, recorded video

and their success rates are summarized in Table 6. When the command inference system accurately inferred situations requiring tissue, the vision system recognized the tissue in all cases. However, when commands such as "The ice is melting" and "I caught a mosquito" were given, the command inference system did not recognize the need for cleaning. Hence, no task commands were sent to the robot. Additionally, in each experiment, the tissue started from a random position. The control success rate was lower compared to the virtual environment experiments due to insufficient training data in the experimental environment, indicating the need for further refinement.

6 Conclusion

This paper introduces a robot system capable of understanding and interpreting human natural language commands to perform tasks. The system comprises three key components: a natural language-based command inference system, a vision transformer-based vision system, and a learning-based control system. By applying the Chain of Thought technique to the GPT-4 model for natural language command inference, the system accurately captures user intent to generate appropriate responses. The inferred commands are translated into object queries for the OWL-ViT model, which recognizes the required objects and communicates their presence to the command inference system, confirming the feasibility of executing the natural language commands. The Diffusion Policy algorithm, trained via ROS, issues control commands to the UR3 robot to perform the tasks. To verify the integrated system's applicability, simulations were conducted in the MuJoCo environment, achieving a 79% success rate for control tasks using the diffusion policy algorithm. Furthermore, real-world experiments with the integrated system were conducted using 20 different natural language commands, achieving a 65% success rate. Future research plans include integrating the inference and vision systems to simplify the model, which is expected to reduce system costs. Additionally, the control success rate will be improved by constructing more diverse and complex datasets for diffusion policy training.

Table 6 Experiment results

Trial	Input (user command)	Command inference system	Vision system	Manipulator control system
1	"Oh no, coke splattered on me."	○	○	○
2	"I am crying."	○	○	○
3	"I spilled coke."	○	○	○
4	"I spilled soup on my clothes."	○	○	X
5	"I knocked over the kimchi container."	○	○	○
6	"Something is on the corner of my mouth."	○	○	○
7	"My phone is dirty."	○	○	X
8	"I caught a mosquito."	X	X	X
9	"My forehead is bleeding."	○	○	○
10	"The ice cream is melting."	○	○	X
11	"Crumbs are falling as I eat a hot dog."	○	○	○
12	"The water from the purifier is leaking."	○	○	○
13	"I painted, and now my clothes are stained."	○	○	○
14	"The ice is melting."	X	X	X
15	"I spat accidentally."	○	○	○
16	"My hands are sweaty."	○	○	○
17	"There is a lot of moisture on the windowsill."	○	○	X
18	"Why is water leaking from the ceiling?"	○	○	○
19	"Crumbs are falling as I eat bread."	○	○	○
20	"Ramen soup spilled on the floor."	○	○	○

Declarations

Conflict of interest The authors declared that there is no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

1. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, I. Polosukhin, Attention is all you need, in *Advances in neural information processing systems 30*. ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Curran Associates Inc, USA, 2017)
2. A. Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Phys. D **404**, 132306 (2020)
3. J.D.M.W.C. Kenton, L.K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in *Proceedings of NAACL-HLT*. (Association for Computational Linguistics, USA, 2019), pp.4171–4186
4. T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal et al., Language models are few-shot learners. Adv. Neural. Inf. Process. Syst. **33**, 1877–1901 (2020)
5. J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song et al., Tidybot: personalized robot assistance with large language models. Auton. Robot. **47**(8), 1087–1102 (2020)
6. M. Kerzel, P. Allgeuer, E. Strahl, N. Frick, J.G. Habekost, M. Eppe et al., Nicol: a neuro-inspired collaborative semi-humanoid robot that bridges social interaction and reliable manipulation. IEEE Access **11**, 123531–123542 (2023)
7. J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi et al., Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural. Inf. Process. Syst. **35**, 24824–24837 (2022)
8. G.I. Winata, A. Madotto, Z. Lin, R. Liu, J. Yosinski, P. Fung, Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer, in *2020 IEEE 33rd international system-on-chip conference (SOCC)*. (IEEE, USA, 2020), pp.84–89
9. S. Lu, M. Wang, S. Liang, J. Lin, Z. Wang, Language models are few-shot multilingual learners, in *2020 IEEE 33rd international system-on-chip conference (SOCC)*. (IEEE, USA, 2020), pp.84–89
10. M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy et al., Simple open-vocabulary object detection, in *European conference on computer vision*. (Springer Nature Switzerland, Cham, 2022), pp.728–755
11. C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, S. Song, Diffusion policy: visuomotor policy learning via action diffusion. arXiv (2020). https://doi.org/10.4855/arXiv.2303.04137
12. O. Ronneberger, P. Fischer, T. Brox, U-net: convolutional networks for biomedical image segmentation, in *Medical image computing and computer-assisted intervention—MICCAI 2015: 18th international conference, Munich, Germany, October 5–9, 2015, proceedings, part III 18*. (Springer International Publishing, Cham, 2015), pp.234–241

Seunguk Choi received his B.S. degree in mechanical engineering from Myongji University, Yongin-si, Gyeonggi-do, Republic of Korea, in 2021. He is currently a graduate student at Myongji University. His research interests include learning-based control, large language model, and reinforcement learning.

Myeonggyun Ahn received his B.S. degree in mechanical engineering from Myongji University, Yongin-si, Gyeonggi-do, Republic of Korea, in 2024. He is currently a graduate student at Myongji University. His research interests include design and control of unmanned aerial vehicle and real-time control.

David Kim received his B.S. and M.S. degrees in mechanical engineering from Myongji University, Yongin-si, Gyeonggi-do, Republic of Korea, in 2020 and 2022, respectively, and is currently a Ph.D. Candidate in the Dynamic Robot Control Laboratory, Department of Mechanical Engineering, Myongji University, Yongin-si, Gyeonggi-do, Republic of Korea. His research interests include design and control of autonomous robot, mobile manipulator, and motion control.

Dongil Choi received his B.S., M.S., and Ph.D. degrees in mechanical engineering from Korea Advanced Institute of Science and Technology (KAIST), in 2005, 2007, and 2012, respectively. He is currently a professor at Myongji University. His research interests include design and control of autonomous robot, biped humanoid robot, and mobile manipulator.

Authors and Affiliations

Seunguk Choi[1] · David Kim[1] · Myeonggyun Ahn[1] · Dongil Choi[1]

✉ Dongil Choi
dongilc@mju.ac.kr

[1] Dept. of Mechanical Engineering, Myongji University, Yongin, South Korea