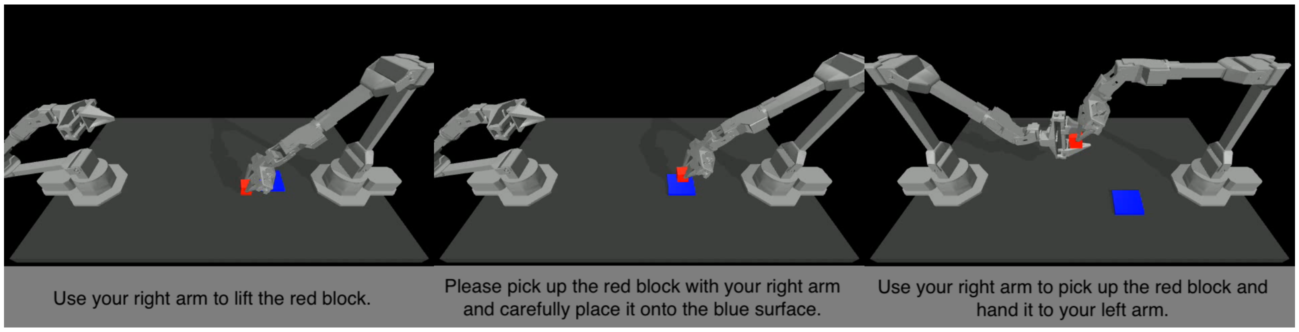
Integrating Natural Language Instructions into the Action Chunking Transformer for Multi-Task Robotic Manipulation

Kevin Rohling

kevin@kevinrohling.com
https://kevinrohling.com
https://github.com/krohling
https://www.linkedin.com/in/krohling/
The University of Texas at Austin, Masters in AI, Department of CS


Use your right arm to lift the red block.
Please pick up the red block with your right arm and carefully place it onto the blue surface.
Use your right arm to pick up the red block and hand it to your left arm.

Abstract

We address the challenge of enabling robots to perform multiple manipulation tasks based on natural language instructions by integrating text embeddings into the Action Chunking Transformer (ACT) model developed by Zhao et al. [2023]. Specifically, we modify the ACT architecture to accept embeddings of task instructions generated using the all-mpnet-base-v2 [Song et al., 2020] model, and integrate them into the transformer [Vaswani et al., 2023] encoder's input. To introduce generalization across task instruction phrasings, we generate a diverse set of paraphrased instructions for each task using GPT-4o [OpenAI et al., 2024] and use random instruction sampling during training to prevent overfitting trajectories to specific instructions.

Our experiments, conducted in a simulated Mujoco [Todorov et al., 2012] environment with a bimanual ViperX robot, focus on three tasks: object grasping, stacking, and transfer. We collect two datasets composed of 50 episodes per task with randomized object placements, one with noise applied to trajectories during data collection [Tangkaratt et al., 2021] and one without noise. For each task we use 25 task instruction phrasings during training and hold out 10 for evaluation to assess generalization. The modified ACT model achieves an overall success rate of 89.3% across the three tasks, demonstrating the ability to map unseen task instructions to robotic control sequences.

1 Introduction

Robots have a long history of integration into manufacturing and industrial applications, however, with recent AI advances, robotics is poised for deployment into more dynamic environments with complex and increasingly diverse tasks. Traditional methods of instructing robots often involve hard-coded instructions or fixed task specifications [Team, 2023], which are not scalable for the variety of tasks a robot may encounter in an environment such as a consumer household [Zhou, 2021, McKay, 2023]. Natural language offers a flexible and intuitive medium for task specification, allowing for more diverse applications and improving the end user experience [Covariant, 2024].

Additionally, as demonstrated in recent works like SayCan [Ahn et al., 2022] and LLM-Grounder [Yang et al., 2023], the rise of Large Language Models (LLMs) [Minaee et al., 2024] presents significant opportunities to bridge the gap between high-level reasoning and low-level robotic control [Shipps, 2024, Dickson, 2024]. LLMs excel at understanding human-like text and have shown significant gains in reasoning [Plaat et al., 2024] and problem-solving performance, making them suitable for interpreting complex instructions and generating structured strategies for task execution. Connecting LLMs with models capable of robotic control can enable the use of both global reasoning (handled by the LLM) and local action planning and execution (handled by the robotic manipulation models).

The Action Chunking Transformer (ACT) [Zhao et al., 2023, Osmulski, 2024], which uses an adapted Detection Transformer architecture [Carion et al., 2020, Potrimba, 2023], was chosen for this project due to its high performance on fine-grained manipulation tasks and it's ability to handle proprioceptive inputs, such as vision and joint positions. In real-world tasks, ACT has demonstrated impressive performance, achieving success rates of 92% on challenging real world tasks such as putting on a shoe and 96% on slotting a battery into a remote controller, outperforming other state-of-the-art models by significant margins [Zhao et al., 2023]. For example, in a comparative evaluation, ACT surpassed BeT [Shafiullah et al., 2022], RT-1 [Brohan et al., 2023], and VINN [Pari et al., 2021] by up to 59% in simulated tasks and consistently achieved higher success rates in real-world manipulation tasks, such as sliding a ziploc bag closed, where ACT achieved 88% success compared to 0% for all other baseline methods.

To leverage the benefits of multi-task learning and natural language task specification, this study aims to implement support for text embeddings in the Action Chunking Transformer (ACT) architecture and introduce a corpus of varied task instructions into the training process. The benefits of this integration are two-fold: it allows the model to map unseen natural language instructions to specific robotic control tasks; and it enables multi-task learning by conditioning action generation on the task instructions. By extending the ACT model to handle natural language inputs in this way, we seek to enhance the robot's ability to understand and execute manipulation tasks based on unseen natural language instructions.

2 Related Work

Enabling robots to interpret and execute tasks based on natural language instructions has been a significant focus in robotics research. Stepputtis et al. [2020], for example, implemented a language-conditioned imitation learning [Lőrincz, 2019] framework using Faster R-CNN, proposed by Ren et al. [2016], that successfully integrated visual inputs, task instructions and joint positions into a single encoding. This approach achieved high performance on sequential tasks such as picking (94% success rate) and pouring (85% success rate).

CLIPort, introduced by Shridhar et al. [2021], combines the visual-semantic understanding of a pre-trained CLIP ResNet50 [Radford et al., 2021a,b] with the spatial precision of the Transporter network [Zeng et al., 2022]. This enables the model to interpret natural language instructions and locate relevant objects in complex scenes. CLIPort achieved high performance on manipulation tasks, with success rates on a box packing task of 98.2% for seen object colors and 71.5% on unseen object colors in simulation.

SayCan, introduced by Ahn et al. [2022], integrates large language models (LLMs) with learned value functions [P, 2023] to enable execution of high-level natural language instructions in real-world environments. By grounding the LLM's reasoning in the robot's capabilities and current state, SayCan combines task relevance (via task likelihoods extracted from the LLM) with feasibility likelihoods (via learned value functions) to determine the best action to take at each step. This approach achieved a planning success rate of 84% and an execution success rate of 74% on 101 diverse tasks in a real kitchen setting.

RoboAgent, proposed by Bharadhwaj et al. [2023], introduces the MT-ACT (Multi-Task Action Chunking Transformer) architecture which supports the specification of embeddings derived from natural language for task specification. Additionally, Bharadhwaj et al. [2023] present a novel semantic data augmentation method using SegmentAnything [Kirillov et al., 2023] and an in-painting model [Lugmayr et al., 2022] to fully automate the augmentation process. Their model demonstrated proficiency across 38 real-world tasks involving 12 unique manipulation tasks, such as wiping and sliding, in diverse kitchen environments. RoboAgent outperformed prior methods by over 40% on unseen scenarios, achieving an average success rate of 25% on unseen environments and up to 90% on seen environments for tasks such as Clean Up and Baking Prep.

Finally, our model architecture is most similar to the MT-ACT model proposed by Bharadhwaj et al. [2023], which also integrates natural language embeddings into the ACT model for multi-task learning. However, our model is trained on a set of text embeddings derived from a corpus of natural language instructions, enabling the model to map unseen task instructions to specific tasks. To the best of our knowledge, it does not appear that this capability was explored in the RoboAgent paper. Additionally, the MT-ACT architecture includes modifications to the visual backbone models that introduce FiLM [Perez et al., 2017] layers to condition the visual features on the instruction embeddings. Our implementation did not include this modification, but we believe it is a brilliant idea that would likely improve the model's performance. While we did not use the MT-ACT codebase for this project appropriate credit for the implementation of a similar and novel model architecture modification is due.

3 Methods

Action Chunking Transformer Overview The Action Chunking Transformer (ACT) [Zhao et al., 2023] is a transformer-based model designed to learn fine-grained bimanual manipulation skills for robotic systems. Leveraging the transformer [Vaswani et al., 2023] architecture's ability to model long-range dependencies and process sequential data effectively, ACT generates coherent action sequences necessary for complex manipulation tasks.

The ACT model is composed of several key components: a Visual Encoder based on a pre-trained ResNet model [He et al., 2015], transformer-based Action Encoder and Decoder, and a Variational Autoencoder (VAE) [Kingma and Welling, 2022]. The VAE is used during training to compress the ground truth action sequence and joint positions into a style variable, z, and is discarded for inference. The transformer encoder processes the visual features, current joint positions, and style variable, z, to generate a latent representation which is then used by the decoder to generate action sequences.

Instruction Embedding Model We use the `all-mpnet-base-v2` model [Song et al., 2020] to generate embeddings from natural language instructions. This model is selected for its strong benchmark performance as well as it's ability to generate full setence embeddings [Oguzoglu, 2021]. The `all-mpnet-base-v2` model combines masked language modeling and permuted language modeling during pre-training, resulting in embeddings that capture both global and local semantic information. $d_{embed} = 768$ is the dimensionality of the text embeddings generated by the model [sentence transformers, 2024].

Model Modifications for Natural Language Integration To enable the ACT model to interpret and execute tasks based on natural language instructions, we introduce several modifications to the architecture. These changes allow the model to incorporate text embeddings derived from task instructions, conditioning action generation on natural language inputs.

First, it was necessary to convert the text embeddings into the input dimensionality of the transformer encoder, $d_{model} = 512$. This was achieved by projecting the text embeddings into the desired dimensionality using a linear projection layer.

$$e_{instr} \in R^{d_{embed}} \quad e_{proj} \in R^{d_{model}} \quad W_{proj} \in R^{d_{model} \times d_{embed}} \quad b_{proj} \in R^{d_{model}}$$

$$e_{proj} = W_{proj} \cdot e_{instr} + b_{proj}$$

where e_{instr} is the original embedding of the natural language instruction, and W_{proj} and b_{proj} are the weights and biases of the projection layer.

Additionally, it was necessary to update the model's positional embeddings to accommodate the new input. The ACT model applies positional information to the encoder inputs with one positional encoding for the visual features and another positional encoding for the joint positions. We introduced a third positional encoding for the text embeddings, ensuring that the model can differentiate between input types during processing.

$$e_{pos} \in R^{2 \times d_{model}} \rightarrow e_{pos} \in R^{3 \times d_{model}}$$

Finally, projected text embeddings are incorporated into the transformer encoder by concatenating them with the existing input sequence. Specifically, the text embedding is concatenated to the sequence of inputs that includes the latent variable from the VAE and the robot's proprioceptive data:

$$S_{input} = [e_{proj}, z, e_{visual}, j]$$

where z is the latent style variable generated by the VAE, e_{visual} represents the features generated by the Visual Encoder and j is the robot's current joint positions. The combined input sequence is then processed by the transformer encoder to generate a latent representation that is used by the decoder to generate action sequences.

By projecting the text embeddings to the appropriate dimensionality, appending them with the encoder input and applying the modified positional embeddings we were able to successfully integrate the natural language instructions into the ACT model. These modifications enable the model to learn to associate natural language instructions with corresponding tasks as well as condition the generated action sequences on the task instructions.

Codebase Our implementation builds upon the publicly available `act` repository provided by Zhao et al. [2023], accessible at `https://github.com/tonyzhaozh/act`. This codebase includes the initial implementation of the Action Chunking Transformer (ACT) and the simulation environment used for dataset generation and evaluation. By leveraging this codebase as a starting point, we were able to focus on extending the model to incorporate natural language instructions and support multi-task learning.

To support our experiments, we made a number of modifications to the original codebase:

- **Addition of New Tasks**: We introduced two new manipulation tasks—*Grasping* and *Stacking*—including their respective evaluation components and scripted policies for dataset generation. The *Transfer* task was retained from the original codebase.
- **Dataset Format Modification**: The dataset structure was modified to include a `task_id` attribute, enabling the model to differentiate between tasks during training.
- **Model Architecture Enhancement**: We adjusted the model architecture to accept text embeddings by applying the architecture modifications described in the previous section.
- **Dataset Class Modification**: The `EpisodicDataset` class was updated to support multi-task datasets and implement random instruction sampling process during training.
- **Simulation Environment Updates**: We introduced a blue square region into the simulation environment to serve as the target area for the stacking task. Object placements, including the red block and blue square, were randomized at the start of each episode to avoid trajectory memorization.
- **Data Recording Enhancements**: Modifications were made to support recording training episodes for multiple tasks, including updates to environment initialization and data collection procedures.
- **Integration with `wandb`**: We integrated Weights and Biases (`wandb`) into the training pipeline for experiment tracking and visualization.
- **Task Instruction Embeddings**: Code was implemented to pre-generate task instruction embeddings using `all-mpnet-base-v2` [Song et al., 2020].
- **Evaluation Script Update**: The evaluation script was modified to assess multi-task checkpoints and capture task-specific success rates.
- **Docker Container Implementation**: A Docker container was created to simplify deployment to cloud GPU services.

Simulation Environment We conducted our experiments in a simulated environment using MuJoCo and DeepMind Control Suite [Todorov et al., 2012, Tassa et al., 2018]. The environment features a bimanual ViperX robotic platform equipped with two 6-degree-of-freedom arms. The robot operates in a workspace containing two objects: a red cube intended for manipulation and a blue square region serving as the target area for the stacking task.

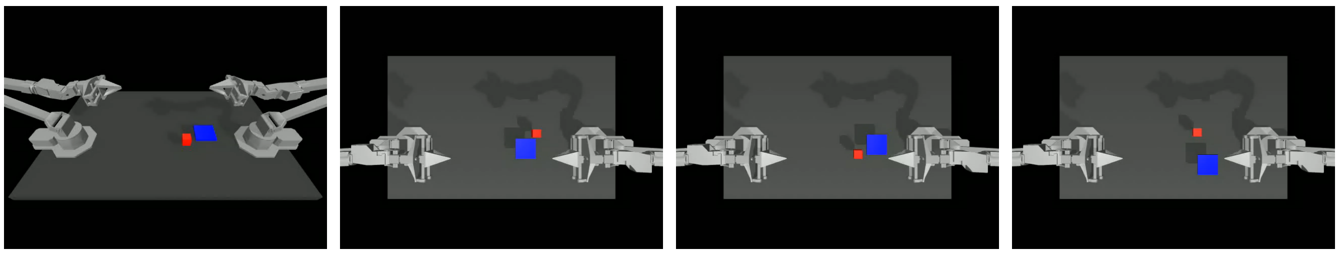


Figure 1: Simulation environment initialization was consistent across all task types to avoid providing environmental cues. Object placements were randomized at the beginning of each episode.

The simulation environment was designed to be consistent across all three tasks to ensure that the robot's behavior is conditioned exclusively on the natural language instructions rather than environmental cues. Object placements for the red cube and blue square region were randomized at the beginning of each episode to enhance the model's generalization capabilities.

Task Descriptions We trained the model to perform the following tasks:

1. **Grasping**: The robot's right arm is instructed to grasp the red cube and lift it off the table. Success is defined as the red cube being held by the right gripper and not in contact with the table.
2. **Stacking**: The right arm must pick up the red cube and place it onto the blue square region. Success requires the red cube to be on the blue square, not touching the table, and not touching the right gripper.
3. **Transfer**: The left arm moves to a receiving position while the right arm picks up the red cube and transfers it to the left arm. Success is achieved when the red cube is held by the left gripper.

Data Collection To generate diverse natural language instructions for each task, we used GPT-4o [OpenAI et al., 2024] to produce 35 paraphrased instructions per task. We provided initial task instructions and prompted the model to generate alternative phrasings. From these, 25 instructions per task were selected for training, and the remaining 10 were held out for evaluation. This approach was designed to promote generalization across different phrasings of the same task.

We collected datasets for each task by executing the scripted policies in the simulation environment. Each dataset comprised 50 successful episodes per task, resulting in 150 episodes in total. Two versions of the dataset were generated: one with noise applied to the robot arm's trajectories during data collection and one without noise. For the noisy dataset, the noise was sampled uniformly from $[-0.01, 0.01]$ meters and added to the trajectories to simulate slight inaccuracies and variations during task execution. The datasets were stored in HDF5 format, resulting in approximately 75 GB of data each and have been made available for download from the Hugging Face Datasets Hub:

- `https://huggingface.co/datasets/kevin510/act-grasp-stack-transfer`
- `https://huggingface.co/datasets/kevin510/act-grasp-stack-transfer-noisy`

Training Procedure During training, each episode was associated with a task instruction. To prevent fitting instructions to specific trajectories and improve the model's ability to generalize to new phrasings, we used a random sampling strategy for the instruction embeddings. For each training iteration, a task instruction was uniformly sampled from the pool of 25 training instructions corresponding to the episode's task. The corresponding pre-generated text embedding was retrieved and provided to the model.

Separate models were trained using the standard and noisy datasets to assess the impact of noise injection on performance. Each model was trained for 50,000 iterations, with checkpoints saved every 2,500 iterations for evaluation, which required approximately 45 hours of training time.

Model performance was evaluated using the success rate of task execution in the simulation environment. Success criteria were defined explicitly for each task, as outlined previously. During training, the model was evaluated every 2,500 iterations by performing 15 rollout episodes (five per task) using the 10 held-out instructions for each task. To improve assessment confidence, after training was completed, each model checkpoint was evaluated on 150 rollout episodes (50 per task) computing task-specific and overall success rates.

Training was conducted on cloud-based GPU servers provided by RunPod.io. Models were trained using the following hardware specifications:

Hardware Component	**Value**
GPU	NVIDIA A10 (48GB VRAM)
vCPUs	9
RAM	50GB

Table 1: Training hardware specifications.

Weights and Biases (`wandb`) was used for experiment tracking and to streamline deployment a Docker image containing both the simulation and training environment was pushed to DockerHub. The docker image is available for download at:

- `https://hub.docker.com/r/krohling/csml-final`

Hyperparameters were chosen based on recommendations from the original ACT paper [Zhao et al., 2023]. Preliminary experiments were run to determine the number of training iterations with the goal of ensuring convergence and capturing the full performance trajectory up to the point of saturation.

Hyperparameter	Value
Learning rate	1×10^{-5}
Transformer Hidden dimension	512
Feedforward dimension	3,200
KL divergence weight	10
Chunk size	100
Training Iterations	50,000

Table 2: Key hyperparameters used during training.

Dataset	Training Iteration	Overall (%)	Grasping (%)	Stacking (%)	Transfer (%)
With Noise Injection	22500	88.7	100.0	70.0	96.0
Without Noise Injection	40000	89.3	100.0	80.0	88.0

Table 3: Overall and Per-task success rates for the best checkpoints trained with and without noise injection. Each success rate is based on 50 evaluation episodes per task.

4 Results

The following figures present the results of our experiments evaluating the performance of our modified Action Chunking Transformer (ACT) model on the Grasping, Stacking, and Transfer tasks. The data was collected from both datasets, one with noise injected into the trajectories during data collection and one without noise.

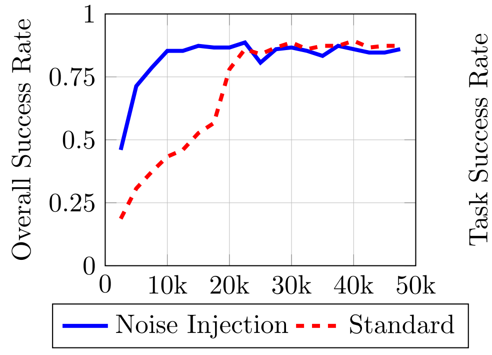


Figure 2: Overall Success Rates on datasets with and without noise injection

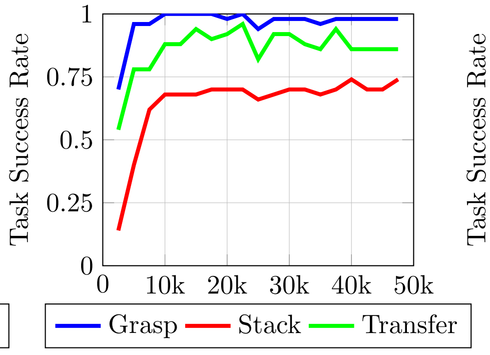


Figure 3: Individual Task Success Rates on the dataset with noise injection

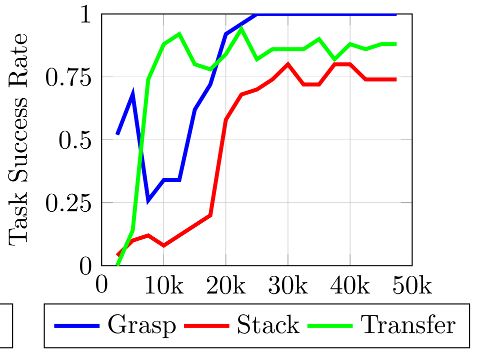


Figure 4: Individual Task Success Rates on the dataset without noise injection

Overall Performance The overall success rate of each model was computed by averaging the success rates across all three tasks. The model trained with noise injection achieved a maximum overall success rate of 88.7% at iteration 22,500, while the model trained without noise injection achieved a slightly higher overall success rate of 89.3% at iteration 40,000. Both models demonstrate high overall success rates, indicating the effectiveness of the modified ACT model in executing multiple tasks when given unseen natural language instructions.

Individual Task Performance

1. **Grasping Task**: Both models achieved perfect performance, 100%, on the Grasping task. This indicates that the Grasping task was relatively easy for the models to learn. This is likely due to the straightforward actions required for successful execution. Additionally, since the Stacking and Transfer tasks both required the robot to grasp the red cube, this skill would have been reinforced during training on those tasks as well.
2. **Stacking Task**: The success rate for the Stacking task was 70% and 80% for the datasets with and without noise injection respectively. This tasks's performance is notably lower than the Grasping task and the lowest of the three tasks. The Stacking task requires the most spatial precision and scene understanding given that

both the positions of the red block and the blue square region are stochastic waypoints. This likely contributed to the lower success rate and why it was the most difficult for the model to learn.

3. **Transfer Task**: The Transfer task success rate was 96% for the dataset with noise injection and 88% for the dataset without noise injection. While performance on this task was below the relatively simple Grasping task, it was still well above the performance of the Stacking task. This task also requires spatial precision, however, the trajectories for Transfer contain more deterministic waypoints. For example, the left and right arms always meet at the same location, which may have made learning easier as the only stochastic waypoint was the block location.

The Role of Noise Injection The inclusion of noise during data collection was intended to improve the model's robustness and generalization. Comparing the performance of the two models however reveals that both achieved comparable overall success rates, with the model trained without noise injection slightly outperforming the model trained with noise injection. This suggests that the models were able to generalize effectively without the need for additional noise in the training data.

However, of particular note, the model trained with noise injection achieved higher success rates significantly earlier in training compared to the model without noise injection. For example, at 10,000 iterations, the model trained with noise injection achieved an overall success rate of 85.3% compared to just 43.3% for the model without noise injection. This indicates that while noise injection did not significantly impact convergence it did dramatically speed up learning in the initial training stages, likely by exposing the model to a wider variety of states. However, as training progressed, the model without noise injection caught up and slightly surpassed the performance of the model with noise injection. This suggests that while eventual convergence is possible with or without noise injection, using a dataset with noise may facilitate faster learning. It is also likely that in more complex environments with more varied states and distractors noise injection would also result in convergence to a higher overall success rate.

Summary The experimental results demonstrate that the modified ACT [Zhao et al., 2023] model is capable of performing multiple manipulation tasks based on unseen natural language instructions with high success rates. The ability to generalize to unseen instructions and maintain performance across tasks highlights the effectiveness of integrating text embeddings into the ACT architecture.

5 Conclusion

In this work, we extended the Action Chunking Transformer (ACT) to enable multi-task robotic manipulation based on natural language instructions. By integrating text embeddings into the ACT architecture, we allowed the model to interpret and execute multiple manipulation tasks conditioned on diverse and unseen natural language instructions. Our experiments demonstrated that the modified ACT model could successfully perform Grasping, Stacking, and Transfer tasks with high success rates, effectively mapping unseen instructions to robotic control sequences.

We achieved an overall success rate of 89.3% across the three tasks, with perfect performance on the Grasping task and strong results on the Stacking and Transfer tasks. The model demonstrated the ability to generalize to unseen task instructions, highlighting the effectiveness of incorporating natural language embeddings into the ACT architecture. Additionally, our results indicated that while noise injection during data collection accelerated early training, both models—with and without noise—converged to similar performance levels.

Limitations Despite the promising results, our study has a numer of limitations we believe are important to address. One limitation is the focus on only three specific manipulation tasks. While these tasks effectively demonstrate the model's capabilities, a broader range of tasks would more comprehensively evaluate the ability to generalize across different types of manipulation. Additionally, the manipulated object in all tasks was always the red cube, and we did not explore generalization to different objects, which is important for real-world environments where robots must interact with a diverse collection of scene elements. The simulation environment lacked diverse scene types and distractors, limiting the relevance of the results to simplified scenarios. Furthermore, we did not incorporate Feature-wise Linear Modulation (FiLM) [Perez et al., 2017] layers, as proposed by Bharadhwaj et al. [2023] [Perez et al., 2017], into the visual backbone of the model. Incorporating this modification would likely enhance the model's ability to attend to task-relevant visual features, useful for tasks that require greater scene understand, such as the Stacking task. Lastly, we used the default hyperparameters used in the original ACT paper without any attempt to explore alternatives. It is likely that hyperparameter optimization could further improve the model's performance.

Future Work Building upon our findings, several avenues for future research have been identified. Expanding the model's capability to generalize across a wider variety of tasks, including manipulation of different object types, would demonstrate it's usefulness to more complex environments. We believe that incorporating semantic augmentation techniques, as detailed by Bharadhwaj et al. [2023], is a very promising direction to explore for enabling broader generalization in a data efficient manner. Further exploration into the integration of large language models (LLMs) for high-level reasoning, as described in the SayCan framework [Ahn et al., 2022], could enable the combination of global planning and local control, leveraging natural language as an interface between the two. Collecting a larger and more diverse set of task instruction phrasings, including adversarial augmentations, as well as introducing varied scene configurations and distractors, would likely improve the model's robustness. Using sim-to-real strategies [Simsangcheol, 2023], such as the one described by Ho et al. [2021], could facilitate transferring the learned policies from simulation to real robotic platforms. Finally, collecting real-world datasets or leveraging existing ones would help in validating the model's applicability beyond simulation.

Closing Remarks Our work demonstrates the potential of integrating natural language instructions into ACT models for robotic manipulation. By enabling robots to interpret and execute tasks based on a variety of natural language commands, we move closer to the goal of intuitive human-robot interaction. Future research that addresses the identified limitations and explores the suggested directions has the potential to significantly advance the field of natural language-conditioned robotic control, paving the way for more adaptable and intelligent robotic systems in real-world applications.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking, 2023. URL https://arxiv.org/abs/2309.01918.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023. URL https://arxiv.org/abs/2212.06817.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020. URL https://arxiv.org/abs/2005.12872.

Covariant. Rfm-1: Allowing robots and people to communicate in natural language. https://covariant.ai/insights/rfm-1-allowing-robots-and-people-to-communicate-in-natural-language/, 2024. Accessed: 2024-11-30.

Ben Dickson. How llms are ushering in a new era of robotics. https://venturebeat.com/ai/how-llms-are-ushering-in-a-new-era-of-robotics/, 2024. Accessed: 2024-11-30.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Daniel Ho, Kanishka Rao, Zhuo Xu, Eric Jang, Mohi Khansari, and Yunfei Bai. Retinagan: An object-aware approach to sim-to-real transfer, 2021. URL https://arxiv.org/abs/2011.03148.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL https://arxiv.org/abs/1312.6114.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023. URL https://arxiv.org/abs/2304.02643.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models, 2022. URL https://arxiv.org/abs/2201.09865.

Zoltán Lőrincz. A brief overview of imitation learning. https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c, 2019. Accessed: 2024-11-30.

Chris McKay. Google researchers teach robots new skills with just natural language. https://www.maginative.com/article/google-researchers-can-teach-robots-new-skills-with-just-natural-language/, 2023. Accessed: 2024-11-30.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey, 2024. URL https://arxiv.org/abs/2402.06196.

Busra Oguzoglu. Sentence embedding methods— a survey. *Medium*, December 2021. URL https://medium.com/@busra.oguzoglu/sentence-embedding-methods-a-survey-7c62857f7b43. Accessed: 2024-11-30.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL `https://arxiv.org/abs/2303.08774`.

Radek Osmulski. An introduction to the action chunking transformer. `https://radekosmulski.com/how-to-train-your-robot-with-a-transformer/`, 2024. Accessed: 2024-11-30.

Prema P. Reinforcement learning value function: A detailed analysis. `https://www.linkedin.com/pulse/reinforcement-learning-value-function-detailed-analysis-prema-p/`, 2023. Accessed: 2024-11-30.

Jyothish Pari, Nur Muhammad Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation, 2021. URL `https://arxiv.org/abs/2112.01511`.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017. URL `https://arxiv.org/abs/1709.07871`.

Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Back. Reasoning with large language models, a survey, 2024. URL `https://arxiv.org/abs/2407.11511`.

Petru Potrimba. What is detr (detection transformers)? `https://blog.roboflow.com/what-is-detr/`, 2023. Accessed: 2024-11-30.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021a. URL `https://arxiv.org/abs/2103.00020`.

Alec Radford, Ilya Sutskever, Jong Wook Kim, Gretchen Krueger, and Sandhini Agarwal. Clip: Connecting text and images. `https://openai.com/index/clip/`, 2021b. Accessed: 2024-11-30.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. URL `https://arxiv.org/abs/1506.01497`.

sentence transformers. all-mpnet-base-v2. `https://www.aimodels.fyi/models/huggingFace/all-mpnet-base-v2-sentence-transformers`, 2024. Accessed: 2024-11-30.

Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone, 2022. URL `https://arxiv.org/abs/2206.11251`.

Alex Shipps. Natural language boosts llm performance in coding, planning, and robotics. `https://news.mit.edu/2024/natural-language-boosts-llm-performance-coding-planning-robotics-0501`, 2024. Accessed: 2024-11-30.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation, 2021. URL `https://arxiv.org/abs/2109.12098`.

Simsangcheol. What is sim2real? `https://medium.com/@sim30217/sim2real-fa835321342a`, 2023. Accessed: 2024-11-30.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding, 2020. URL `https://arxiv.org/abs/2004.09297`.

Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks, 2020. URL `https://arxiv.org/abs/2010.12083`.

Voot Tangkaratt, Nontawat Charoenphakdee, and Masashi Sugiyama. Robust imitation learning from noisy demonstrations, 2021. URL `https://arxiv.org/abs/2010.10181`.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018. URL `https://arxiv.org/abs/1801.00690`.

The RoboCat Team. Robocat: A self-improving robotic agent. `https://deepmind.google/discover/blog/robocat-a-self-improving-robotic-agent/`, 2023. Accessed: 2024-11-30.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

Jed Yang, Xuweiyi Chen, Shengyi Qian, Madhavan Iyengar, David Fouhey, and Joyce Chai. When llms and robots meet. `https://cse.engin.umich.edu/stories/when-llms-and-robots-meet`, 2023. Accessed: 2024-11-30.

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Ayzaan Wahid, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation, 2022. URL `https://arxiv.org/abs/2010.14406`.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL `https://arxiv.org/abs/2304.13705`.

Li Zhou. Teaching robots to respond to natural-language commands. `https://www.amazon.science/blog/teaching-robots-to-respond-to-natural-language-commands`, 2021. Accessed: 2024-11-30.

A Resources

Datasets: The datasets used in this study are available for download from the Hugging Face Datasets Hub:

  - `https://huggingface.co/datasets/kevin510/act-grasp-stack-transfer`
  - `https://huggingface.co/datasets/kevin510/act-grasp-stack-transfer-noisy`

- **Docker Image**: A Docker image containing the simulation and training environment is available for download from DockerHub:
  - `https://hub.docker.com/r/krohling/csml-final`
- **Modified ACT Codebase**: *Coming Soon*

B Task Instructions Corpus

B.1 Grasping Task Instructions

```
Right arm, grasp the red block.
Use your right arm to lift the red block.
Right arm, retrieve the red block from the table.
Right arm, execute pickup of the red block.
With your right arm, secure the red block.
Right arm, extend and capture the red block.
Activate right arm to pick up the red block.
Right arm, initiate retrieval of the red block.
Right arm, maneuver to grasp the red block.
Right arm, approach and lift the red block.
Use your right arm to clasp onto the red block.
Right arm, perform the action of picking up the red block.
Right arm, engage and lift the red block.
Right arm, execute the grab of the red block.
Right arm, deploy to pick up the red block.
Right arm, proceed to secure the red block.
Command the right arm to take hold of the red block.
Right arm, execute a pickup maneuver for the red block.
Right arm, perform the task of lifting the red block.
Direct the right arm to retrieve the red block.
Right arm, apply grip to the red block.
Right arm, activate and lift the red block.
Enable right arm to acquire the red block.
Instruct right arm to grasp and lift the red block.
Right arm, carry out the pickup of the red block.
Right arm, take control of the red block.
Command the right arm to seize the red block.
Right arm, accomplish the pickup of the red block.
Right arm, enact the lifting of the red block.
Right arm, proceed with the grasping of the red block.
Right arm, mobilize to secure the red block.
Right arm, undertake the retrieval of the red block.
Right arm, handle the task of picking up the red block.
Right arm, go ahead and grab the red block.
Right arm, complete the action of picking up the red block.
```

B.2 Stacking Task Instructions

```
Robot, grasp the red block and set it down on the blue surface.
Please pick up the red block with your right arm and carefully place it onto the blue surface.
```

```
Using your right arm, retrieve the red block and deposit it on the blue surface.
Take the red block with your right arm and relocate it to the blue surface.
Your task is to lift the red block with your right arm and then rest it on the blue surface.
Maneuver your right arm to grab the red block and gently position it on the blue surface.
With your right arm, seize the red block and place it carefully on the blue surface.
Use your right arm to collect the red block and move it to rest on the blue surface.
Command your right arm to fetch the red block and deposit it atop the blue surface.
Engage your right arm to capture the red block and position it on the blue surface.
Robot, your objective is to transfer the red block using your right arm to the blue surface.
Initiate a sequence to lift the red block with your right arm and settle it on the blue surface.
Employ your right arm to pick up the red block and arrange it on the blue surface.
Direct your right arm to clasp the red block and station it on the blue surface.
Task your right arm to hoist the red block and nestle it onto the blue surface.
Operate your right arm to secure the red block and place it on the designated blue surface.
Instruct your right arm to snatch the red block and lay it down on the blue surface.
Commandeer the red block with your right arm and transfer it to the blue surface.
Have your right arm take the red block and allocate it to the blue surface.
Engage your right arm to elevate the red block and deposit it on the blue surface.
Robot, use your right arm to pick up the red block and position it onto the blue surface.
Activate your right arm to get a hold of the red block and place it gently on the blue surface.
Direct your right arm to grasp the red block and perch it on the blue surface.
Right arm, proceed to lift the red block and set it on the blue surface.
Your mission is to use the right arm to retrieve the red block and place it on the blue surface.
Use your right arm to acquire the red block and carefully position it on the blue surface.
Utilize your right arm to pick up the red block and establish it on the blue surface.
Command your right arm to take up the red block and rest it upon the blue surface.
Task your right arm to procure the red block and relocate it to the blue surface.
Let your right arm fetch the red block and place it on the blue surface.
Employ the right arm to grasp the red block and move it to the blue surface.
Have your right arm secure the red block and deposit it onto the blue surface.
Use the right arm to pick the red block and then lay it on the blue surface.
Tell the right arm to capture the red block and set it down on the blue surface.
Command the right arm to pick up the red block and position it atop the blue surface.
```

B.3 Transfer Task Instructions

```
Grasp the red block with your right hand and pass it to your left hand.
Use your right arm to pick up the red block and hand it to your left arm.
Transfer the red block from your right arm to your left arm.
With your right arm, retrieve the red block and deliver it to your left arm.
Employ your right arm to lift the red block and shift it to your left arm.
Command your right arm to secure the red block and relocate it to your left arm.
Activate your right arm to take hold of the red block and place it into your left arm.
Instruct your right arm to acquire the red block and confer it to your left arm.
Direct your right arm to grasp the red block and transfer custody to your left arm.
Engage your right arm to pick up the red block and hand it over to your left arm.
Utilize your right arm to capture the red block and deposit it to your left arm.
Initiate the pickup of the red block with your right arm and hand it to your left arm.
Enable your right arm to lift the red block and pass it along to your left arm.
Tell your right arm to pick up the red block and transition it to your left arm.
Have your right arm grab the red block and move it to your left arm.
Commandeer the red block with your right arm and allocate it to your left arm.
Assign your right arm to pick up the red block and relinquish it to your left arm.
Instruct your right arm to seize the red block and transfer it over to your left arm.
Guide your right arm to clutch the red block and yield it to your left arm.
Order your right arm to fetch the red block and confer it onto your left arm.
Mobilize your right arm to secure the red block and consign it to your left arm.
```

```
Signal your right arm to take the red block and present it to your left arm.
Get your right arm to pick up the red block and switch it to your left arm.
Instruct your right arm to hoist the red block and allocate it to your left arm.
Prompt your right arm to catch the red block and entrust it to your left arm.
Set your right arm to collect the red block and bestow it upon your left arm.
Deploy your right arm to grasp the red block and surrender it to your left arm.
Enable your right arm to snag the red block and transfer it to your left arm.
Engage your right arm to lift the red block and hand it over to your left arm.
Have your right arm take the red block and pass it to your left arm.
Direct your right arm to take possession of the red block and transfer it to your left arm.
Command your right arm to fetch the red block and hand it to your left arm.
Orchestrate a transfer of the red block from your right arm to your left arm.
Tell your right arm to retrieve the red block and deliver it to your left arm.
Charge your right arm to secure the red block and convey it to your left arm.
```