GRAPE: Generalizing Robot Policy via Preference Alignment

Zijian Zhang[1,*], Kaiyuan Zheng[2,*], Zhaorun Chen[3,*], Joel Jang[2], Yi Li[2], Siwei Han[1], Chaoqi Wang[3]
Mingyu Ding[1], Dieter Fox[2], Huaxiu Yao[1]

***Abstract*— Despite the recent advancements of vision-language-action (VLA) models on a variety of robotics tasks, they suffer from critical issues such as poor generalizability to unseen tasks, due to their reliance on behavior cloning exclusively from successful rollouts. Furthermore, they are typically fine-tuned to replicate demonstrations collected by experts under different settings, thus introducing distribution bias and limiting their adaptability to diverse manipulation objectives, such as efficiency, safety, and task completion. To bridge this gap, we introduce GRAPE: Generalizing Robot Policy via Preference Alignment. Specifically, GRAPE aligns VLAs on a trajectory level and implicitly models reward from both successful and failure trials to boost generalizability to diverse tasks. Moreover, GRAPE breaks down complex manipulation tasks to independent stages and automatically guides preference modeling through customized spatiotemporal constraints with keypoints proposed by a large vision-language model. Notably, these constraints are flexible and can be customized to align the model with varying objectives, such as safety, efficiency, or task success. We evaluate GRAPE across a diverse array of tasks in both real-world and simulated environments. Experimental results demonstrate that GRAPE enhances the performance of state-of-the-art VLA models, increasing success rates on in-domain and unseen manipulation tasks by 51.79% and 58.20%, respectively. Additionally, GRAPE can be aligned with various objectives, such as safety and efficiency, reducing collision rates by 37.44% and rollout step-length by 11.15%, respectively.**

I. Introduction

The recent rapid proliferation of vision-language-action (VLA) models has streamlined general robotic manipulation tasks, demonstrating impressive capability across a range of tasks under controlled environmental variations [4], [6], [25], [42]. However, these models face several critical challenges such as poor generalizability across new environments, objects, tasks, and semantic contexts [25]. A significant factor contributing to this limitation is their reliance on *supervised fine-tuning* (SFT), where VLAs simply imitate actions from successful rollouts via behavior cloning while not developing a holistic understanding of the task goal or potential failure patterns [26]. While reinforcement learning (RL) algorithms such as PPO [40] have proved promising in enhancing their generalizability [51], the high cost of gathering sufficient online trajectories and explicitly defining reward make them impractical for training VLA [42].

Furthermore, training VLAs to solely replicate expert behaviors often results in *behavior collapse* [27] where the planned trajectories are often suboptimal [25]. This is because the SFT datasets are usually uncurated and consist of offline demonstrations collected from experts that embed implicitly different values (e.g. task completion, safety, and cost-efficiency) that are not clearly defined within the data [35], [43]. Simply imitating these behaviors via SFT can potentially confuse the model and result in suboptimal trajectories that deviate from the actual objective of the demonstrations. Some approaches attempt to address this challenge by explicitly defining a set of objectives and solving them hierarchically [22]. However, this approach incurs additional inference overhead and lacks scalability [29].

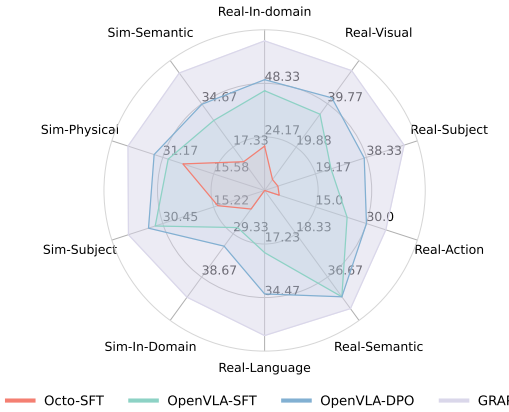

Fig. 1: Comparison of GRAPE with SOTA VLA models fine-tuned on the same data across a large variety of generalization and in-domain tasks in both real-world and simulated environments.

To address these issues, we propose **GRAPE**: **G**ene**ra**lizing Robot Policy via **Pre**ference Alignment to alleviate the high cost of training VLAs with RL objective, while offering flexibility for aligning towards customized manipulation objectives. As shown in Fig. 2, GRAPE introduces *trajectory-wise preference optimization* (TPO) to align VLA policies on a trajectory level by implicitly modeling reward from both successful and failure trials, boosting generalizability to diverse tasks. To further alleviate the difficulty in ranking trajectories and providing preferences towards arbitrary alignment objectives, GRAPE proposes to decompose the complex manipulation tasks into multiple independent stages and adopt a large vision model to propose keypoints for each stage, each associated with a spatial-temporal constraint. Notably, these constraints are flexible and can be customized to align the model with varying manipulation objectives, such as task completion, robot-interaction safety, and cost-efficiency. We evaluate GRAPE across a wide range of real-world tasks and two simulated environments. Experimental results show

*Equal contribution. [1]UNC-Chapel Hill, Chapel Hill, NC, USA, [2]University of Washington, Seattle, WA, USA, [3]University of Chicago, Chicago, IL, USA. Corresponding author: Huaxiu Yao huaxiu@cs.unc.edu

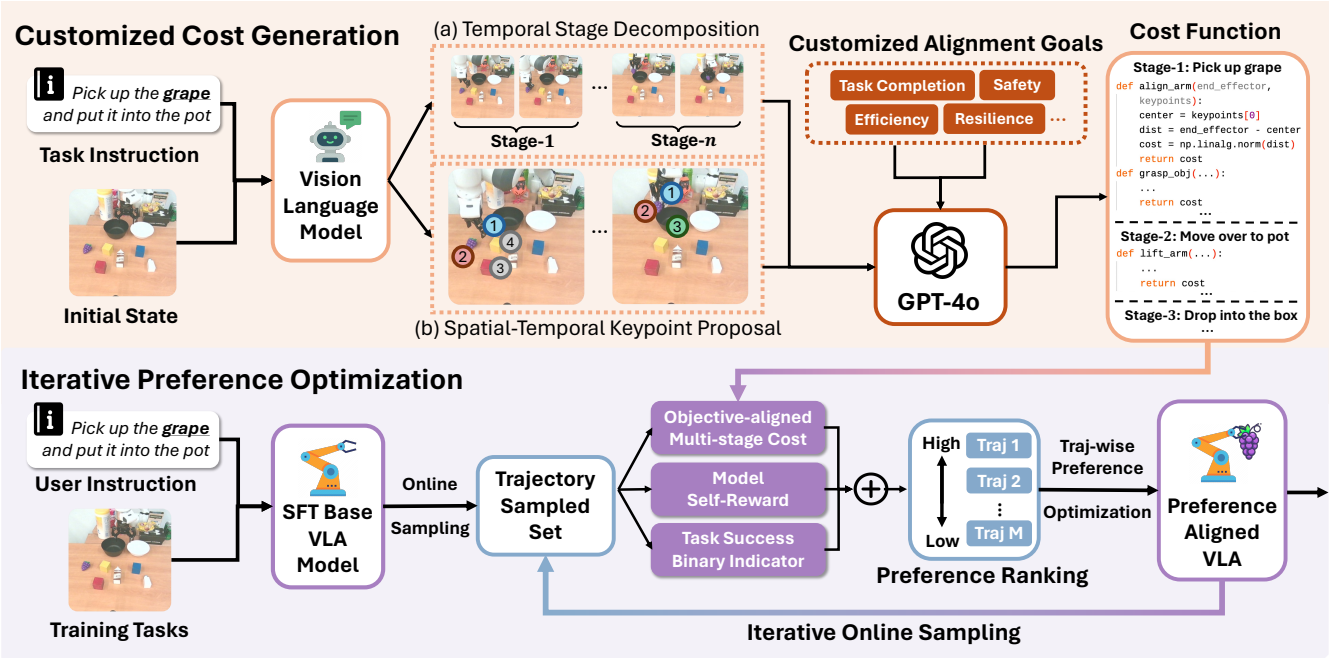

Fig. 2: **Overview of GRAPE.** GRAPE first uses a VLM to decompose a manipulation task (**top**) into temporal stages and identify key spatial points for each subtask. Given user-specified alignment goals, it prompts a VLM to generate cost functions for each stage. During iterative preference optimization (**bottom**), offline trajectories are sampled from the base VLA model, scored using multi-stage cost, self-evaluation and task success indicators, and ranked to form preferences. GRAPE then optimizes the VLA models iteratively until convergence.

that GRAPE outperforms state-of-the-art VLA models, improving success rates on both in-domain and unseen manipulation tasks by 51.79% and 58.20%, respectively. Moreover, GRAPE can be aligned to diverse objectives such as safety and efficiency, to further reduce collision rate by 37.44% and rollout step-length by 11.15%, respectively.

II. Generalizing Robot Policy via Preference Alignment

A. Preliminaries

During inference, a VLA typically initializes with an task instruction q, and at each timestep t, it takes an environment observation o_t (usually an image) and outputs an action a_t, where we can denote $\pi_\theta(a_i|(o_i, q))$ as the action policy of a VLA parameterized by θ. To complete the task, VLA iteratively interacts with the environment and obtains a trajectory $\zeta = \{o_1, a_1, \cdots, o_T, a_T|q\}$ of length T. Typically, VLAs are fine-tuned to imitate expert behaviors via SFT:

$$\mathcal{L}_{\text{SFT}} = -\sum_{(\zeta,q)\in\mathcal{D}}\sum_{t=1}^{T}\log p(a_t|o_t, q; \pi_\theta), \tag{1}$$

where $\mathcal{D} = \{(\zeta_1, q_1), \ldots, (\zeta_N, q_N)\}$ denotes the training set containing N expert trajectories. Specifically, $\mathcal{L}_{\text{SFT}}$ enforces VLA to memorize the action associated with each observation sampled from a distribution $\mathbb{P}_\mathcal{D}$, resulting in poor generalizability to new task settings. It is worth to note that while we follow [6], [35] and consider the step-wise policy based on the Markov decision process (MDP) assumption [41], our approach can be easily adapted to both non-MDP case which takes past interaction histories (usually a video or a series of images) as state [9] and diffusion policy [16] which generates multiple future steps all at once [42].

B. TPO: Trajectory-wise Preference Optimization

To improve generalization, we follow [3], [40] and further fine-tune VLA policies via RL objective. Let r_ϕ denote a reward function parameterized by ϕ, we have

$$\max_{\pi_\theta} \mathbb{E}_{\zeta\sim\pi_\theta}\left[r_\phi(\zeta)\right] - \beta D_{\text{KL}}\left[\pi_\theta(\zeta) \,\|\, \pi_{\text{ref}}(\zeta)\right], \tag{2}$$

where β controls the deviation from the base reference policy π_{ref} trained via SFT in Eq. (1) and $\pi(\zeta, q)$ is the likelihood of policy π generating the entire trajectory ζ under instruction q. Then we follow [38] and derive the analytical reparameterization of the trajectory reward $r(\zeta)$ as:

$$r(\zeta, q) = \beta\log\frac{\pi_\theta(\zeta \mid q)}{\pi_{\text{ref}}(\zeta \mid q)} + \beta\log Z(\zeta). \tag{3}$$

Similar to [38], we adopt the Bradley-Terry (BT) [5] model and model r_ϕ from a set of trajectories ranked with preferences. Specifically, let ζ_w and ζ_l denotes the chosen and rejected trajectory starting from the same initial state, we can formulate the trajectory-wise reward modeling objective as:

$$P\left(\zeta_w \succ \zeta_l\right) = \frac{\exp\left(r(\zeta_w), q\right)}{\exp\left(r(\zeta_w), q\right) + \exp\left(r(\zeta_l), q\right)}. \tag{4}$$

Then, we follow [38] and substitute Eq. (3) into Eq. (4) and obtain the following *trajectory-wise preference optimization* (TPO) loss $\mathcal{L}_{\text{TPO}}$ equivalent to Eq. (2):

$$\mathcal{L}_{\text{TPO}} = -\mathbb{E}_{(\zeta_w,\zeta_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\left(\log\frac{\pi_\theta(\zeta_w)}{\pi_{\text{ref}}(\zeta_w)} - \log\frac{\pi_\theta(\zeta_l)}{\pi_{\text{ref}}(\zeta_l)}\right)\right)\right], \tag{5}$$

where we can further draw from MDP and decompose the likelihood of a trajectory ζ into individual state-action pairs, i.e., $\pi(\zeta, q) = \prod_{i=1}^{T} \pi(a_i \mid (o_i, q))$ and further obtain

$$\log \frac{\pi_\theta(\zeta, q)}{\pi_{\text{ref}}(\zeta, q)} = \sum_{t=1}^{T} \log \frac{\pi_\theta(a_i \mid (o_i, q))}{\pi_{\text{ref}}(a_i \mid (o_i, q))}. \tag{6}$$

Then we can substitute Eq. (6) into Eq. (5) to obtain the TPO loss $\mathcal{L}_{\text{TPO}}$ in terms of step-wise state-action pairs. Our TPO loss Eq. (6) is beneficial as it: (1) aligns policy π_θ globally towards human preferences on a trajectory level while simply using step-wise rollouts collected by VLAs; (2) it stabilizes the policy and steers it towards the final goal by backpropagating the gradients throughout all the state-action pairs along the trajectory; (3) it significantly boosts generalizability by learning from both successful and failed trajectories via a RL objective. Although [20] indicates that expanding the size of the sampled trajectory can reduce the bias in reward modeling, it also increases the training costs. Thus while our method can be easily scaled up, we keep our discussion to the binary case where only one chosen/rejected trajectory is present.

C. Guided-Cost Preference Generation

While given the TPO objective Eq. (5) we can align the policy towards arbitrary objectives defined through trajectories ranked by the corresponding preference, it incurs high costs as it requires human expertise and lengthy manual annotation. Thus to better scale up the preference synthesis towards arbitrary alignment objectives (e.g. task completion, safety, efficiency), we propose *Guided-Cost Preference Generation (GCPG)* to automatically curate such preferences that integrate different alignment objectives.

1) Multi-Stage Temporal Keypoint Constraints

Building on insights from [22], we address the complexity of specifying precise trajectory preferences for complex manipulation tasks by decomposing trajectories into temporal stages and assigning costs to quantify performance at each stage. Then, we aggregate these stage-specific costs to obtain a holistic evaluation for each trajectory. Specifically, we adopt a VLM-based stage decomposer $\mathcal{M}_D$ (detailed in Appendix VIII), to partition a trajectory ζ into a sequence of $\mathbf{S}$ consecutive stages, formulated as

$$\{\zeta^1, \ldots, \zeta^S\} = \mathcal{M}_D(\zeta, q), \quad \zeta^i = \{(o_t^i, a_t^i)\}_{t=1}^{T_i}, \tag{7}$$

where ζ^i represents the i^{th} stage of trajectory ζ.

After obtaining the stage decomposition, we further employ a vision-language model (e.g. DINOv2 [36]) to identify keypoints that serve as reference metrics across each stage. Then we prompt a powerful LLM [1] to propose cost functions (see examples in Appendix XII-B.) for each stage that corresponds with the alignment objective, where lower cost indicates better objective compliance. Specifically, the cost $C^{S_i}(\{\kappa_{S_i}\})$ at stage S_i is calculated using its corresponding keypoints $\{\kappa_{S_i}\}$.

Then to aggregate the costs for the entire trajectory, instead of summing each stage linearly, we apply an exponential decay to capture the casual dependencies of each temporal stage (e.g. if a trajectory incurs high costs in preceding stages it is not expected to perform well subsequently), defined as the *external reward*:

$$R_{\text{ext}}(\zeta) = \prod_{i=1}^{\mathbf{S}} e^{-C^{S_i}(\{\kappa_{S_i}\})} \tag{8}$$

where Eq. (8) aggregates the individual costs and sub-objectives from each stage to tackle the curse of dimensionality and effectively adhere to the customized alignment.

2) Guided-Cost Preference Generation

To further improve the stability and optimality of the preference synthesis, we draw inspirations from self-rewarding [53] and determine that *a more optimal trajectory should be confirmed by both the external judge (as in Eq. (8)) and the model itself*. Thus we incorporate two additional rewards and obtain the GCPG reward:

$$R_{\text{GCPG}}(\zeta) = \lambda_1 R_{\text{self}}(\zeta) + \lambda_2 R_{\text{ext}}(\zeta) + \lambda_3 I_{\text{success}}(\zeta) \tag{9}$$

where $R_{\text{self}}(\zeta)$ is the self-evaluated score provided by π, which equals the log-likelihood of generating trajectory ζ:

$$R_{\text{self}}(\zeta) = \log(\pi(\zeta, q)) = \log(\prod_{i=1}^{T} \pi(a_i \mid (o_i, q))) \tag{10}$$

and $I_{\text{success}}(\zeta)$ is a binary indicator function that indicates whether the trajectory ζ successfully completes the task:

$$I_{\text{success}}(\zeta) = \begin{cases} 1, & \text{if } \zeta \text{ is successful,} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

where λ are the weight parameters that adjust the importance of each reward. Intuitively, Eq. (10) can be seen as a dense approximation of the sparse signal provided by Eq. (11), which are further calibrated by Eq. (8) to obtain a holistic evaluation of the trajectory that accounts for both its optimality and degree of alignment to a customized objective specified through the external reward in Eq. (8).

D. Iterative Preference Optimization

After generating the preference, we then discuss our iterative preference optimization strategy. Inspired by the practices of on-policy RL [40] which often yield more optimal policy than off-policy training, we iteratively fine-tune the SFT VLA model via TPO with trajectories collected online. For example, during the k^{th} iteration, we (1) first sample numerous trajectories for a variety of tasks and obtain $\mathcal{D}^k$; (2) then we calculate the costs for each trajectory using Eq. (9) and rank these trajectories accordingly per task; (3) we pair the top-m and bottom-m trajectories with each other for each task, and obtain m^2 chosen-rejected trajectory pairs; (4) then we fine-tune the same sampling policy with TPO via Eq. (5) and obtain an updated policy. We iterate this process for K times and obtain the final model aligned with the target objective. We detail the GRAPE iterative preference optimization procedure in Algorithm 1.

III. Experiment

In this section, we evaluate GRAPE's performance in both real and simulated environments, addressing four key questions: (1) Does GRAPE improve the VLA model's performance relative to SFT-based baseline models? (2) How

effective are guided-cost preference selection and iterative preference optimization in enhancing the model's performance? (3) What is the individual contribution of each reward component to overall model performance? (4) Can GRAPE support flexible alignment with different alignment objectives? The experiment results and additional analysis can be found in Appendix VI.

A. Experimental Setups

Implementation Details. We employ OpenVLA [25] as the backbone model, using LoRA fine-tuning with the AdamW optimizer for both supervised and preference fine-tuning. In the supervised fine-tuning stage, we use a learning rate of 4×10^{-5} with a batch size of 16. For preference fine-tuning, we apply a learning rate of 2×10^{-5} with the same batch size. Further details on the training process and datasets are available in Appendices VIII and IX.

Baseline Models. We first compare GRAPE with two leading robot learning models known for their strong performance in robot control tasks. The first model, Octo [42], is a large transformer-based policy model. The second, OpenVLA [25], is a 7B VLA model. Both models were supervised fine-tuned using the same dataset sampled from corresponding environments. We denote the supervised fine-tuned models as Octo-SFT and OpenVLA-SFT, respectively. In addition, we compare GRAPE, which utilizes TPO, with the original step-wise direct preference optimization, denoted as OpenVLA-DPO, which is directly trained to optimize preferences defined at each step.

B. Evaluation in Simulation Environment

Evaluation Setup. Follow [25], we evaluate GRAPE's performance in two robot simulation environments: Simpler-Env [28] and LIBERO [32]. In Simpler-Env, we evaluate the model's in-domain performance as well as its generalization across three aspects: subject (generalize to unseen objects), physical (generalize to unseen object sizes/shapes), and semantic (generalize to unseen instructions) generalization. In LIBERO, we test our model on four tasks: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. All tasks are in-domain tasks. Additional details about the experimental setup are provided in Appendix X-B.

Results. We use the success rate across all tasks in Simple-Env and LIBERO as our primary evaluation metric, while we also record the grasping rate in Simpler-Env. The results of Simple-Env and LIBERO are reported in Fig. 5 and Fig. 6, respectively. According to the results, GRAPE outperforms Octo-SFT and OpenVLA-SFT in Simpler-Env by an average of 131.72% and 46.10%, respectively, and in LIBERO by an average of 8.53% and 7.36%, respectively. Additional results are provided in Appendix XI. This outcome aligns with our expectations, as learning from preference comparisons enhances alignment with trajectory completion, thereby improving performance. Moreover, while GRAPE significantly boosts in-domain performance, it also enhances the generalizability of VLA policies on OOD tasks by aligning task completion at the trajectory level. Furthermore, GRAPE outperforms OpenVLA-DPO in both environments, achieving an average improvement of 33.14%, demonstrating the effectiveness of trajectory-wise preference optimization due to learning from both success and failure from a global trajectory level without low-level step-wise noises.

C. Evaluation in Real-World Robot Environment

Evaluation Setup. We conducted 300 real-world experiments across 30 tasks to evaluate the generalization capabilities of GRAPE. The evaluation focus on in-distribution evaluation and five out-of-distribution generalization types: visual, subject, action, semantic, and language grounding generalizations. Here, visual generalization assesses the ability to adapt to new visual environments; subject generalization evaluates the recognition and handling of unfamiliar objects; action generalization measures performance across diverse actions; semantic generalization evaluates responses to prompts with similar meanings; and language grounding generalization gauges comprehension of spatial directions. Detailed experimental setup are provided in Appendix X-A and illustrated in Figure 3.

Results. In the real-world experiment, GRAPE significantly outperforms other models across a variety of tasks. Notably, in in-domain tasks, GRAPE achieves a success rate of 67.5%, which is a 17.5% improvement over OpenVLA-DPO's 50%, OpenVLA-SFT's 45% and substantially higher than Octo-SFT's 20%. Additionally, in visual generalization tasks, GRAPE demonstrates higher adaptability with a success rate of 56%. In the more challenging action generalization tasks, although OpenVLA-SFT shows modest performance, GRAPE still outperforms OpenVLA-SFT, indicating its potential in understanding various actions and executing commands based on language. Considering tasks across all categories, GRAPE's total average success rate is 50.3%, marking a 11% improvement over OpenVLA-DPO's 39.3%, OpenVLA-SFT's 32.3% and significantly ahead of Octo-SFT's 5.7%. This performance highlights (1) GRAPE's effectiveness and adaptability in handling complex and variable task environments and (2) validates the effectiveness of trajectory-wise preference optimization in learning from global success and failure patterns when compared to OpenVLA-DPO.

The rest of the experiment results and additional analysis can be found in Appendix VI.

IV. Conclusion

In this work, we addressed the critical challenges faced by vision-language-action (VLA) models, including limited generalizability and adaptability to diverse manipulation objectives. We proposed GRAPE, which aligns VLA policies on a trajectory level. GRAPE enhances generalizability by learning from both successful and failed trials, offering flexibility in aligning with objectives such as safety, efficiency, and task success through customized spatiotemporal constraints. Experimental results demonstrated significant improvements, with GRAPE enhancing success rates on both in-domain and unseen tasks while enabling flexible alignment on different objectives. Moreover, we have demonstrated the potential of GRAPE to align VLA with customized objectives, effectively resulting in an improvement of lower collision rate and average step lengths.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B Tenenbaum, Tommi S Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*, 2023.

[3] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[4] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[5] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[8] Lucian Buşoniu, Tim De Bruin, Domagoj Tolić, Jens Kober, and Ivana Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 46:8–28, 2018.

[9] Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, et al. Gr-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024.

[10] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[11] Yuanpei Chen, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen McAleer, Hao Dong, Song-Chun Zhu, and Yaodong Yang. Towards human-level bimanual dexterous manipulation with reinforcement learning. *Advances in Neural Information Processing Systems*, 35:5150–5163, 2022.

[12] Zhaorun Chen, Yichao Du, Zichen Wen, Yiyang Zhou, Chenhang Cui, Zhenzhen Weng, Haoqin Tu, Chaoqi Wang, Zhengwei Tong, Qinglan Huang, et al. Mj-bench: Is your multimodal reward model really a good judge for text-to-image generation? *arXiv preprint arXiv:2407.04842*, 2024.

[13] Zhaorun Chen, Francesco Pinto, Minzhou Pan, and Bo Li. Safewatch: An efficient safety-policy following video guardrail model with transparent explanations. *arXiv preprint arXiv:2412.06878*, 2024.

[14] Zhaorun Chen, Zhuokai Zhao, Tairan He, Binhao Chen, Xuhao Zhao, Liang Gong, and Chengliang Liu. Safe reinforcement learning via hierarchical adaptive chance-constraint safeguards. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[15] Zhaorun Chen, Zhuokai Zhao, Zhihong Zhu, Ruiqi Zhang, Xiang Li, Bhiksha Raj, and Huaxiu Yao. Autoprm: Automating procedural supervision for multi-step reasoning via controllable question decomposition. *arXiv preprint arXiv:2402.11452*, 2024.

[16] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[17] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[18] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[19] Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[20] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016.

[21] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[22] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

[23] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[24] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pages 9902–9915. PMLR, 2022.

[25] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[26] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*, 2021.

[27] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

[28] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[29] Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: Hierarchical action models for open-world robot manipulation. In *1st Workshop on X-Embodiment Robot Learning*, 2024.

[30] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[31] Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. In *International Conference on Machine Learning*, pages 20725–20745. PMLR, 2023.

[32] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[33] Yao Mu, Junting Chen, Qing-Long Zhang, Shoufa Chen, Qiaojun Yu, GE Chongjian, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. Robocodex: Multimodal code generation for robotic behavior synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

[34] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36, 2024.

[35] Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[36] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[37] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[38] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[39] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[41] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

[42] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[43] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[44] Chaoqi Wang, Zhuokai Zhao, Yibo Jiang, Zhaorun Chen, Chen Zhu, Yuxin Chen, Jiayi Liu, Lizhu Zhang, Xiangjun Fan, Hao Ma, et al. Beyond reward hacking: Causal rewards for large language model alignment. *arXiv preprint arXiv:2501.09620*, 2025.

[45] Chaoqi Wang, Zhuokai Zhao, Chen Zhu, Karthik Abinav Sankararaman, Michal Valko, Xuefei Cao, Zhaorun Chen, Madian Khabsa, Yuxin Chen, Hao Ma, et al. Preference optimization with multi-sample comparisons. *arXiv preprint arXiv:2410.12138*, 2024.

[46] Siyue Wang, Zhaorun Chen, Zhuokai Zhao, Chaoli Mao, Yiyang Zhou, Jiayu He, and Albert Sibo Hu. EscIRL: Evolving self-contrastive IRL for trajectory prediction in autonomous driving. In *8th Annual Conference on Robot Learning*, 2024.

[47] Tianhao Wu, Yunchong Gan, Mingdong Wu, Jingbo Cheng, Yaodong Yang, Yixin Zhu, and Hao Dong. Unidexfpm: Universal dexterous functional pre-grasp manipulation via diffusion policy. *arXiv preprint arXiv:2403.12421*, 2024.

[48] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *7th Annual Conference on Robot Learning*, 2023.

[49] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

[50] Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Sejune Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, et al. Latent action pretraining from videos. *arXiv preprint arXiv:2410.11758*, 2024.

[51] Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *arXiv preprint arXiv:2405.10292*, 2024.

[52] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. *arXiv preprint arXiv:2408.11039*, 2024.

[53] Yiyang Zhou, Zhiyuan Fan, Dongjie Cheng, Sihan Yang, Zhaorun Chen, Chenhang Cui, Xiyao Wang, Yun Li, Linjun Zhang, and Huaxiu Yao. Calibrated self-rewarding vision language models. *arXiv preprint arXiv:2405.14622*, 2024.

[54] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

[55] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

V. Additional Introduction to GRAPE

We detail the GRAPE iterative preference optimization procedure in Algorithm 1.

Algorithm 1 GRAPE Iterative Preference Optimization

Require: Base VLA policy π_θ, a collection of task instructions $Q = \{q_i\}$, stage decomposer $\mathcal{M}_D$, max iterations K, reward weights $\{\lambda_1, \lambda_2, \lambda_3\}$, stage-wise keypoints $\{\kappa_{S_i}\}$ cost functions $\{C_j^{S_i}\}$ and thresholds $\{\tau_j^{S_i}\}$

Ensure: policy π^* aligned towards customized objective

for $k = 1, \ldots, K$ **do**
  Sample trajectories $\mathcal{D}^k = \{\zeta_i\}_{i=1}^M$ using π_θ with Q
  for trajectory $\zeta \in \mathcal{D}^k$ **do**
    Decompose ζ into multiple stages S ▷ Eq. (7)
    Compute the cost for each stage C_{S_i}
    Calculate external reward $R_{\text{ext}}(\zeta)$ ▷ Eq. (8)
    Compute policy self-reward $R_{\text{self}}(\zeta)$ ▷ Eq. (10)
    Examine task success $I_{\text{success}}(\zeta)$ ▷ Eq. (11)
    Aggregate GCPG reward $R_{\text{GCPG}}(\zeta)$ ▷ Eq. (9)
  end for
  Rank $\mathcal{D}^k$ by their $R_{\text{GCPG}}(\zeta)$ rewards
  Pair $\{\zeta_w, \zeta_l\}$ from top-m and bottom-m trajectories
  Update π_θ using TPO loss ▷ Eq. (5)
end for

VI. Additional Experiment Results and Analysis

subsectionAblation Study of Reward Model In this section, we conduct an ablation study to analyze the contribution of each reward component in Eq. (9) to the final performance: the external objective-aligned reward $R_{\text{ext}}(\zeta)$, the self-evaluated reward $R_{\text{self}}(\zeta)$, and the success indicator $I_{\text{success}}(\zeta)$. Additionally, we perform a separate ablation study to emphasize the importance of utilizing the entire reward score for preference selection. This approach is compared against a method that randomly selects one successful trajectory as the preferred trajectory and one failed trajectory as the rejected trajectory. The results in the Simpler-Env environment are reported in Table II.

The results indicate that: (1) incorporating the full reward score Eq. (9) for preference ranking significantly enhances performance compared to random selection based on success alone; (2) all reward components contribute to model performance. These findings align with our expectations. Specifically, $R_{\text{self}}(\zeta)$ enhances the robustness of the GRAPE by encouraging it to select trajectories with higher generation probabilities. In parallel, $R_{\text{ext}}(\zeta)$ guides the model toward learning specific behaviors, such as safety and efficiency. Finally, $I_{\text{success}}(\zeta)$ serves as a critical indicator, steering the model to prioritize successful trajectories.

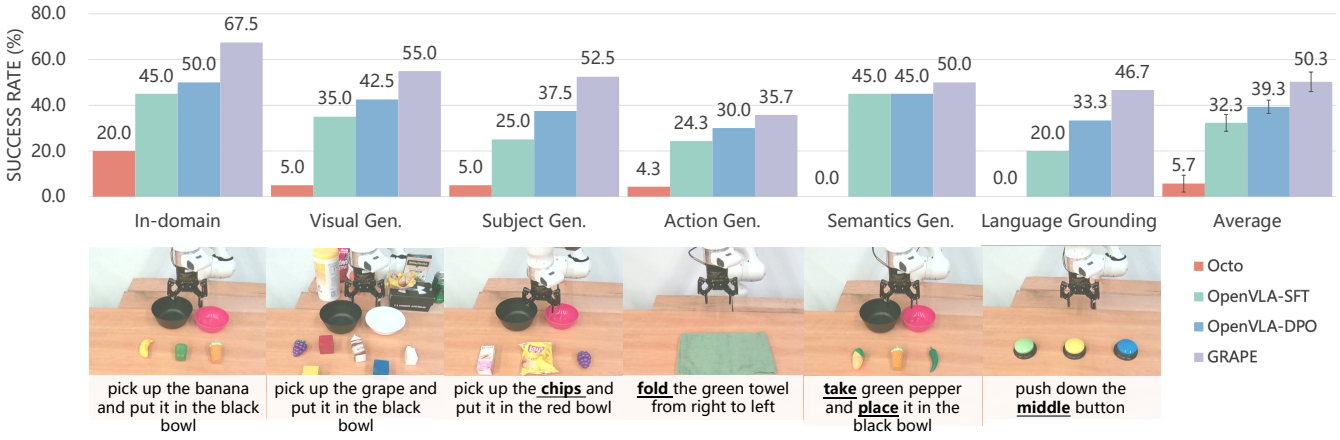

Fig. 3: Comparison of GRAPE with OpenVLA and Octo fine-tuned on the same data on the real-world environment. We report the in-domain performance, which includes four tasks and five generalization evaluations (*visual*, *subject*, *action*, *semantic*, and *language grounding*), incorporating multiple tasks. We report the average performance across all tasks.

A. Analysis of Iterative Preference Optimization

In this section, we analyze the iterative preference optimization performance. We conduct the experiments on the SimplerEnv environment and report the results with respect to the training iterations in Figure 4. Here, SFT means the supervised fine-tuned OpenVLA model before preference optimization. In our experiments, GRAPE achieves 17.5%, 9.0%, 15.0%, 21.0% improvements in in-domain performance, subject generalization, physical generalization and semantic generation, respectively. The findings suggest that GRAPE progressively enhances model performance across iterations, showcasing its ability to enhance the quality of generated preference data and achieve better generalization. Notably, the magnitude of improvement diminishes over time, aligning with our expectations as the model approaches convergence.

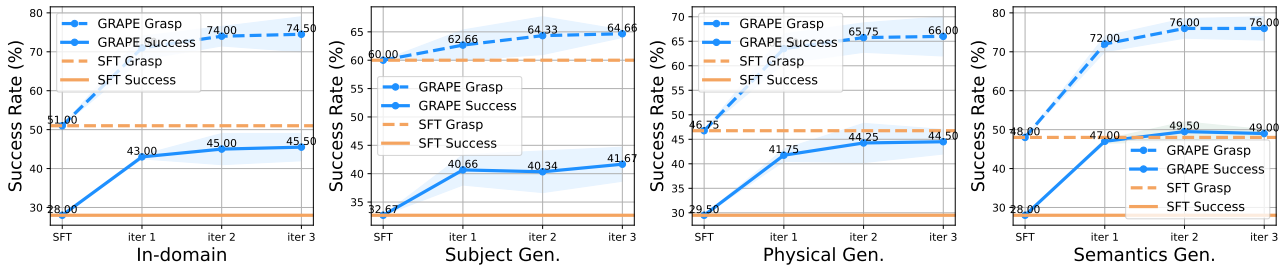

Fig. 4: Performance of GRAPE during iterative preference optimization via TPO. We demonstrate the average success rate for each iteration across in-domain tasks and three types of generation tasks (*subject*, *physical*, *semantics*).

TABLE I: Results with respect to different objectives. GRAPE-Safety, GRAPE-Efficiency, GRAPE-TC are models trained with safety, efficiency, task completion objectives, respectively. Here, we use collision rate (CR), step length (SL), success rate (SR) to evaluate the safety, efficiency and task completion capabilities.

Method	Real-World			Simulation		
	CR ↓	SL ↓	SR ↑	CR ↓	SL ↓	SR ↑
OpenVLA-SFT	53.33	142.32	34.61	66.50	72.68	27.50
GRAPE-Safety	**29.84**	146.11	54.31	**46.00**	74.49	37.00
GRAPE-Efficiency	58.45	**125.79**	51.67	57.50	**64.92**	38.50
GRAPE-TC	38.60	131.66	**58.46**	59.50	70.24	**42.50**

B. Analysis of Different Alignment Objectives

1) Quantitative Analysis

After demonstrating the effectiveness of GRAPE in improving the generalization of the VLA model (measured by success rate), we further investigate its potential to align the model with flexible objectives, such as efficiency and safety. Revisiting Eq. (8), we observe that adjusting the threshold parameters can guide the model to prioritize specific objectives by influencing trajectory preference selection. In this study, we focus on two new alignment objectives: safety and efficiency. Safety aims to minimize collisions between the robot and objects, while efficiency seeks to reduce the average number of steps required for the robot to complete a task. To achieve these objectives, we set a lower threshold for collision costs to emphasize safety and a lower threshold for path costs to prioritize efficiency. These modified settings are then applied to the original real-world and simulation evaluations. We train models to align with the safety and efficiency objectives, referring to these models as GRAPE-Safety and GRAPE-Efficiency, respectively (see detailed experimental setup in Appendix X-B).

The results are reported in Table I, where we use collision rates, step lengths, and success rates to evaluate safety, efficiency and generalization capabilities, respectively. According to Table I, the GRAPE-Safety and GRAPE-Efficiency have better performance on collision rate and step length respectively, meanwhile maintain a comparable success rate, compared with OpenVLA-SFT. The results indicate that GRAPE can be easily adapted to account for flexible alignment objectives such as safety, efficiency by adjusting the multi-stage cost functions accordingly, while incurring minimal drop in task success rate.

2) Case Study

We further demonstrate a case study in Fig. 7 to analyze GRAPE's adaptability towards different alignment objectives. Specifically, we consider a safety-critical *pickup* task where an obstacle is placed between the object and the target. Specifically, OpenVLA-SFT fails to complete the task without preference alignment. However, we can see that while GRAPE aligned towards task completion (on the second-row of Fig. 7) can effectively pick up and place the object, it also collides with the obstacle, due to the policy is aligned to aggressively boost task success without explicitly addressing safety concerns. On the contrary, GRAPE-safety learns to avoid colliding with the obstacle while efficiently completing the

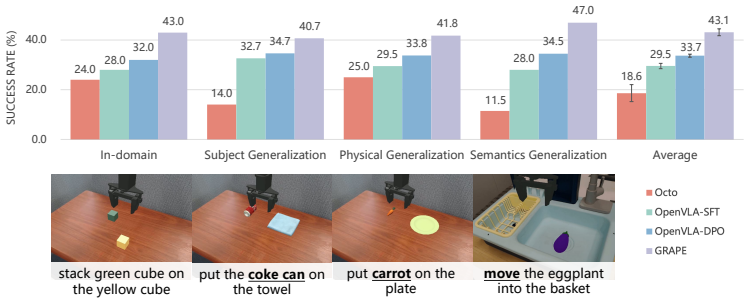

Fig. 5: Comparison of GRAPE with OpenVLA and Octo fine-tuned on the same data on the Simpler-Env environment. We report the in-domain performance, which includes four tasks and three generalization evaluations (*subject*, *physical*, and *semantic*), where each incorporates multiple tasks.

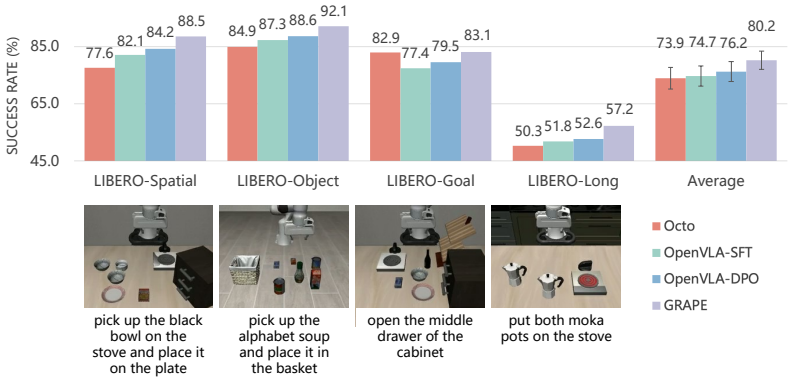

Fig. 6: Comparison of GRAPE with OpenVLA and Octo fine-tuned on the same data on the LIBERO environment. We report the performance on four types of LIBERO tasks.

task. Both Table I and Fig. 7 indicates that by simply tweaking the cost function, GRAPE can effectively adapt to different objectives. More cases and detailed safety evaluation tasks could be found in Appendix XII-A.

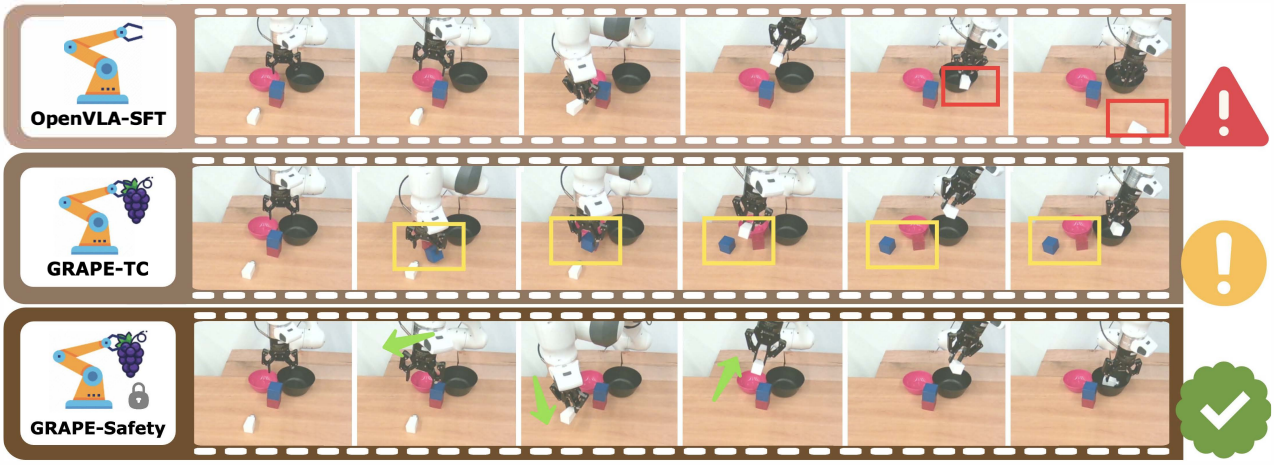

Fig. 7: Comparison of GRAPE aligned via *safety* objective (GRAPE-Safety) with GRAPE aligned via *task-completion* (GRAPE-TC) objective and OpenVLA-SFT. Specifically, we assess their performance on a safety-critical task with the instruction: *pick up the white box and place into the black pot*.

TABLE II: Ablation study of reward score. Here, Random w/ I_{success} refers to randomly selecting one successful trajectory as the chosen trajectory and one failed trajectory as the rejected trajectory, $R_{\text{self}}(\zeta)$ is the self-evaluated score provided by the log-likelihood of generating trajectory ζ, $R_{\text{ext}}(\zeta)$ represents objective-aligned multi-stage reward defined in Eq. (8), $I_{\text{success}}(\zeta)$ is a binary indicator function that indicates whether the trajectory ζ successfully completes the task.

	In-domain		**Subject Gen.**		**Physical Gen.**		**Semantics Gen.**		**Average**	
	Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success
Random w/ I_{success}	62.00%	35.50%	60.33%	33.00%	44.00%	33.50%	54.50%	36.50%	55.21%	34.63%
w/o $R_{\text{self}}(\zeta)$	66.50%	38.00%	62.33%	37.00%	51.25%	36.75%	68.00%	42.50%	62.02%	38.56%
w/o $R_{\text{ext}}(\zeta)$	63.50%	37.50%	61.00%	34.33%	48.50%	35.50%	62.50%	40.00%	58.88%	36.83%
w/o I_{success}	58.50%	32.00%	59.67%	34.67%	42.25%	31.75%	58.50%	39.00%	54.73%	34.36%
GRAPE	**71.00%**	**43.00%**	**62.67%**	**40.67%**	**63.50%**	**41.75%**	**72.00%**	**47.00%**	**67.29%**	**43.11%**

VII. Related Works

Vision-Language-Action Models. Previous robot learning works [14], [22], [23], [29], [30], [33], [34] typically take a hierarchical planning strategy. For example, Code as Policies [30] and EmbodiedGPT [34] use LLMs and VLMs to generate high-level action plans, then rely on a low-level controller for local trajectories. However, such models suffer from limited low-level skills and are hard to generalize to everyday tasks. VLAs tend to scale up low-level tasks by incorporating VLM as backbones and directly generating actions within the model. They generally achieve action planning via two mainstream approaches: (1) Discretizing the action space [6], [7], [25], as in OpenVLA [25], preserves the autoregressive language decoding objective by truncating actions into a small set of *action tokens*. However, this introduces errors, leading some methods [4] to adopt newer structures [52] that integrate diffusion heads for action prediction, avoiding discretization. (2) Diffusion models [2], [16], [24], [31], [48], such as Diffusion Policy [16], serve as the action head, generating a sequence of future actions through iterative denoising instead of stepwise action generation.

While these models vary in structure, they are consistently supervised-trained on successful rollouts via behavior cloning, which can hardly be generalized to unseen manipulation tasks. However, our GRAPE first aligns VLA policies on a trajectory level via trial and error, effectively boosting generalizability and customizability.

Reinforcement Learning and Preference Optimization. Reinforcement learning (RL) [17], [40], [55] plays a pivotal role in the post-training of foundation models [1], [12], [13], [15], [18], [19], [44], [49], which has been extensively leveraged to align the pre-trained FMs to comply with human values embedded through preference data. In the meantime, RL has also shown tremendous success in training policies for robotics tasks [10], [11], [14], [46], [47], [54]. While it is intuitively beneficial to post-align VLA via RL, few prior works have reported such success, mainly due to that (1) manipulation objectives are usually diverse and complex, making the reward hard to define analytically [20]; (2) while such reward can be modeled from human preferences, annotating such preferences in robotics manipulation tasks are usually lengthy [43]; (3) the imperfect numerical differentiation of rewards usually leads RL algorithms such as PPO [40] to collapse [8]. However, various recent works [38], [45] have successfully aligned the policy via RL without explicit reward modeling. Inspired, GRAPE aligns the policy by contrasting trajectories with each other, avoiding issues in rewarding modeling. Besides, we introduce an automatic preference synthesis pipeline that easily scales with diverse manipulation tasks and adapts to different alignment objectives.

VIII. Additional Description of GRAPE and Hyperparameter Settings

Customized Cost Generation. In our real-world experiments, we first input image-text pairs containing prompts and initial states into the Vision-Language Model (VLM) Hamster [29]. Using the stage information and stage points generated by Hamster, we segmented the collected trajectories. This helps analyze complex task sequences more precisely, giving detailed attention to each stage. And we utilized Grounded-SAM [39] or methods combining SAM [39] and DinoV2 [37] to extract key point information from the images. These key points, combined with our self-collected trajectory data, enable us to refine the execution steps and path planning of tasks based on the stage information generated by the Hamster model. For example, for a simple pick-and-place task, we can decompose it into multiple explicit stages: Grasp the grape, Move the grape onto the plate, Place the grape on the plate.

To generate detailed operational information and cost functions for each stage, we utilized GPT-4o [1] with customized prompts. This approach makes stage planning more precise and efficient, allowing us to meet specific task requirements and constraints. Furthermore, we enhanced our method by incorporating various task-specific constraints, including: **Collision constraints:** Ensuring the robot avoids collisions with obstacles. **Path constraints:** Optimizing the efficiency and safety of the robot's movement path. By adopting this strategy, we achieve greater flexibility and specificity in task planning, and better adapting to different task scenarios.

Iterative Preference Optimization. For Iterative Preference Optimization, we first utilize the fine-tuned VLA model for online data sampling. For each task, we sample $\mathcal{N}_t$ trajectories to facilitate further selection. To simplify the experimental setup, we set $\mathcal{N}_t = 5$ for each task, which has been found to perform effectively in practice.

After sampling, each trajectory is automatically labeled using the GCPG reward, as defined in Eq. (9). Based on the distribution of R_{self}, R_{ext}, and I_{success} observed in preliminary experiments, we set $\lambda_1 = 0.01$, $\lambda_2 = 0.01$, and $\lambda_3 = 2$. These values ensure that R_{self}, R_{ext}, and I_{success} contribute comparably to the final reward value. Subsequent experiments validate the reasonableness of these parameter choices. Using the GCPG reward assigned to each trajectory, we identify the trajectory with the highest reward as y_w and the trajectory with the lowest reward as y_l for each task. This selection process enables the construction of the TPO Dataset, $\mathcal{D}_{traj}$, for TPO training.

For the TPO training process, we employ LoRA [21] and the AdamW optimizer, setting the learning rate to 2×10^{-5} and the batch size to 16. The model is trained for a single epoch before being utilized for iterative online sampling. During iterative online sampling, the experimental settings remain consistent with the aforementioned descriptions.

IX. Detail Experiment Datasets

In this section, we describe the datasets collected for supervised fine-tuning (referred to as the SFT dataset) and preference alignment (referred to as the TPO dataset).

A. Real-World Dataset

SFT Dataset. In our real-world robot experiments, we use a robotic platform composed of a Franka robotic arm and a Robotiq gripper for data collection. To ensure consistency in data collection and evaluation, all operations are performed in the same experimental environment.

During data collection, we gathered a dataset of 220 instances of pick and place tasks involving common objects such as bananas, corn, milk, and salt. Additionally, we collected data on 50 instances of tasks involving pressing buttons of different colors. Since the number of objects used for the button-pressing tasks is limited, we introduced background noise and interfering objects during the testing phase to create unseen scenarios.

To further enhance the capabilities of OpenVLA in handling different actions, we also collected data on 50 instances of knock down tasks. These diverse task datasets help improve the model's generalization ability in processing different types of actions.

TPO Dataset. In the real-world experiments, we utilized a model fine-tuned on the real-world SFT dataset via OpenVLA for trajectory sampling. Each task was conducted five times. In the TPO dataset, we experimented with 15 different tasks, including 10 pick and place tasks, 3 push button tasks, and 2 knock down tasks, accumulating a total of 75 data entries. After a selection process, we derived a preference dataset consisting of 30 trajectories.

B. Simulation Datasets

SFT Dataset: For Simpler-Env, the SFT dataset comprises 100 trajectories, amounting to approximately 2,900 transitions. These rollouts are generated from Simpler-Env using Octo, following the methodology described in [50]. For LIBERO, it is worth noting that we neither collect new data nor fine-tune the OpenVLA model. Instead, we directly utilize the OpenVLA-SFT model provided by the OpenVLA team, which significantly streamlines the pipeline.

TPO Dataset. In the case of Simpler-Env, trajectories are sampled for each task using the OpenVLA-SFT model, with five trials conducted per task. This process yields a TPO dataset consisting of 80 trajectories. For LIBERO, OpenVLA-SFT models (one model per task) are employed to sample data across four tasks in LIBERO. For each task, five trajectories are sampled for each sub-task, resulting in a TPO dataset comprising a total of 20 trajectories.

X. Detailed Experiment Settings and Additional Result

A. Real-World

1) Real-World Experiment Setup

In real-world experiment, we used the Franka robot arm, which is known for its precision and flexibility. However, we encountered a problem with the original Franka gripper, which was not long enough, limiting our ability to handle some of the tasks, resulting in inefficient completion and a high failure rate. To solve this problem, we decided to replace the original Franka grippers with Robotiq grippers, which are not only longer, but also provide more grip and flexibility, which greatly improves the efficiency and success rate of the tasks.

The purpose of this experiment was to assess the cross-task generalization capabilities of OpenVLA under the GRAPE framework and to compare its performance with several baseline models. Considering the generally poor zero-shot generalization performance of most VLA models, we performed supervised fine-tuning using the comprehensive rollout dataset D_r collected from real scenes to construct a fine-tuned model. The selection of baseline models included those adjusted with domain-specific data, as well as the Octo model, RVT-2 model, and OpenVLA-SFT model.

2) Real-World Tasks

As shown in Figure 3, we performed a comprehensive evaluation on a real machine for several tasks. These tasks cover five different generalization scenarios: Visual Generalization, Subject Generalization, Action Generalization, Semantics Generalization, and Language Grounding. Specifically, for each generalization scenario, we set the following tasks:

- **Visual Generalization** includes 8 tasks, e.g., pick up the GRAPE and put it in the black bowl, with noise objects and noisy backgrounds. Some tasks have only noisy backgrounds.
- **Subject Generalization** includes 4 tasks, e.g., pick up the K and put it in the black bowl.

- **Action Generalization** includes 7 tasks, e.g., fold the green towel from right to left .
- **Semantics Generalization** includes 4 tasks, e.g., stack carrot and put it on the blue plates.
- **Language Grounding** includes 3 tasks, e.g., pick up left object to left plate.

We conducted experiments on 30 total different tasks, attempting each task ten times, totaling 300 executions. To ensure fairness in the evaluation, we maintained the same starting position in each model test. Additionally, we matched the image resolution when training all models and used exactly the same initial object positions in all evaluations. We set specific success criteria for each task. For example, in the pick-and-place task, a successful grasp is defined as successfully grasping the target object. In the push-button and knock-down tasks, a successful grasp is defined as correctly approaching and manipulating the target object. Overall task success is defined as the object being accurately placed at the target location, successfully knocked down, or the target button being successfully pressed. Due to the strictness of these criteria, some models found it difficult to achieve success in specific tasks.

TABLE III: Comparison of GRAPE models in diffierent iteration rounds. We assess their performance in in-domain tasks and three kinds of generalization evaluations. Each task's performance is evaluated on the overall grasp rate and success rate.

	In-domain		**Subject Gen.**		**Physical Gen.**		**Semantics Gen.**		**Average**	
	Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success
Iter-1	71.00%	43.00%	62.67%	**40.67%**	63.50%	41.75%	72.00%	47.00%	67.29%	43.11%
Iter-2	74.00%	45.00%	64.33%	40.33%	65.75%	44.25%	**76.00%**	**49.50%**	70.02%	44.77%
Iter-3	**74.50%**	**45.50%**	**64.67%**	**40.67%**	**66.00%**	**44.50%**	**76.00%**	49.00%	**70.29%**	**44.92%**

B. Simulation Experiments

1) Simpler-Env

We utilize Simpler-Env [28] as the experimental environment in our study. SIMPLER [28] (Simulated Manipulation Policy Evaluation for Real Robot Setups) is a collection of simulated environments created to assess robot manipulation policies in a way that closely reflects real-world scenarios. By leveraging simulated environments, SIMPLER effectively serves as a practical alternative to real-world testing, which is often costly, time-consuming, and challenging to replicate.

Simpler-Env Tasks. In our paper, we use four in-domain tasks from WidowX robot in Simpler-Env. We also design three kinds of generalization tasks in Simpler-Env. These tasks are described below:

In-Domain Tasks Shown in Fig. 5:

1) Put Carrot on Plate: The robot is positioned in front of a platform with a plate and a carrot. The robot's goal is to grasp the carrot and put it onto the plate.
2) Put Eggplant in basket: The robot is positioned in front of a sink with a basket and a Eggplant. The robot's goal is to grasp the Eggplant and put it in the basket.
3) Stack Green Cube on Yellow Cube: The robot is positioned in front of a platform with a green cube and a yellow cube. The robot's goal is to grasp the green cube and stack it on the yellow cube.
4) Put Spoon on towel: The robot is positioned in front of a platform with a spoon and a towel. The robot's goal is to grasp the spoon and put it on the towel.

Three Kinds of Generalization Tasks Shown in Fig. 5:

1) Subject Generalization: The robot is positioned in front of a platform, similar to the environment in in-domain tasks. But the robot's goal is to grasp some new objects(i.e. pepsi can, coke can, sprite can) and put it onto the plate.
2) Physical Generalization: The robot is positioned in front of a platform, similar to the environment in in-domain tasks. But the robot's goal is to grasp some original objects with different sizes and collision boxes, then put it onto the plate.
3) Semantics Generalization: The robot is positioned in front of a platform, similar to the environment in in-domain tasks. And the instruction is similar to in-domain tasks, too. But the instruction has been modified by GPT-4o [1] while maintaining its original meaning.

2) LIBERO

We further utilize LIBERO [32] as the experimental environment in our study. LIBERO (LIfelong learning BEnchmark on RObot manipulation tasks) includes a set of 130 language-conditioned robot manipulation tasks inspired by human activities, organized into four distinct suites. Each suite is crafted to examine distribution shifts in object types, spatial arrangements of objects, task goals, or a combination of these factors. LIBERO is built to be scalable, extendable, and specifically tailored for advancing research in lifelong learning for robotic manipulation.

LIBERO tasks In our paper, we use four in-domain tasks from LIBERO, which are shown in Fig. 6. These tasks is described below:

- **LIBERO-Spatial** includes the same set of objects arranged in various layouts, testing the model's ability to understand spatial relationships.
- **LIBERO-Object** features consistent scene layouts with varying objects, evaluating the model's ability to understand different object types.
- **LIBERO-Goal** includes of the same objects and layouts but different task goals, testing the model's knowledge of different task-oriented behaviors.
- **LIBERO-10** consists of long-horizon tasks with diverse objects, layouts, and tasks.

Eash task mentioned above has 10 sub-tasks, with similar task instructions and scenes. Here are some cases from various LIBERO tasks:

- Open the top drawer of the cabinet and put the bowl in it.
- Pick up the book and place it to the right of the caddy.
- Turn on the stove and put the frying pan on it.
- Stack the right bowl on the left bowl and place them in the tray.

XI. Additional Real-World and Simulation Results

We provide additional results in Table IV , Table V, and Figure 12 with detailed task description. Each table has in-domain tasks and several kinds of generalization evaluations. These experiments are conducted across Octo-SFT, OpenVLA-SFT and GRAPE.

TABLE IV: We present the performance of various action policy on real-world robotic manipulation tasks categorized by different types of generalization. The tasks include in-domain, visual generalization with and without noise, subject generalization, action generalization, semantics generalization, and language grounding. Each task's performance is evaluated based on the number of successful grasps and the overall success rate, comparing results from Octo-SFT, OpenVLA-SFT, OpenVLA-DPO, and GRAPE. Average success rates are calculated for each generalization category to demonstrate the effectiveness of the tested models under different conditions.

Generalization	**Task**	**Octo-SFT**		**OpenVLA-SFT**		**OpenVLA-DPO**		**GRAPE**	
		Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success
In-domain	pick up the corn and put it in the black bowl	3	3	2	2	5	3	8	7
	pick up the banana and put it in the black bowl	2	0	6	6	8	6	9	7
	pick up the milk and put it in the white bowl	4	2	10	8	8	8	9	9
	pick up the salt bottle and put it in the white bowl	4	3	4	2	5	3	6	4
	Average	32.5%	20%	55%	45%	65%	50%	**80%**	**67.5%**
Visual Generalization (w/o noise background)	pick up the corn and put it in the black bowl	2	1	6	3	6	4	6	6
	pick up the banana and put it in the black bowl	0	0	3	2	4	1	4	1
	pick up the milk and put it in the white bowl	4	0	4	4	6	6	9	7
	pick up the salt bottle and put it in the white bowl	2	2	6	5	6	6	8	8
	pick up the GRAPE and put it in the black bowl	0	0	6	5	8	5	8	6
	Average	16%	6%	50%	38%	60%	44%	**70%**	**56%**
Visual Generalization (w/o noise background and object)	pick up the GRAPE and put it in the black bowl	1	0	4	2	5	3	6	4
	pick up the milk and put it in the white bowl	2	1	7	5	6	4	5	4
	pick up the salt bottle and put it in the white bowl	0	0	2	2	6	5	8	8
	Average	10%	3.3%	43.3%	30%	56.7%	40%	**63.3%**	**53.3%**
Subject Generalization)	pick up the chips and put it in the red bowl	4	0	2	2	4	3	6	5
	pick up the K and put it in the black bowl	2	0	4	4	6	5	7	6
	pick up the box juice and put it in the yellow plate	2	0	8	3	8	5	8	6
	pick up the Fanta can and put it in the white bowl	2	2	4	1	5	2	6	4
	Average	25%	5%	45%	25%	57.5%	37.5%	**67.5%**	**52.5%**
Action Generalization	push down the blue button	1	0	4	4	6	4	6	6
	push down the green button	1	0	6	4	7	5	4	4
	push yellow the button	2	2	6	3	7	4	8	5
	knock down the green bottle	3	1	2	2	3	2	4	2
	knock down the popcorn	0	0	4	2	4	3	4	3
	fold the green towel from right to left	1	0	2	1	3	1	4	2
	fold the white towel from left to right	1	0	3	1	4	2	4	3
	Average	12.9%	4.3%	38.6%	24.3%	**48.6%**	30%	**48.6%**	**35.7%**
Semantics Generalization	take green pepper and place it in the black bowl	0	0	10	6	9	7	10	8
	move icecream and put it in the red bowl	0	0	6	4	5	4	4	4
	stack carrot and put it on the blue plates	0	0	8	8	6	5	6	6
	Lift GRAPE and place it in the black bowl	0	0	2	0	3	2	2	2
	Average	0%	0%	**65%**	45%	57.5%	45%	55%	**50%**
Language Grounding	pick up left object to left plate	0	0	4	0	5	1	5	2
	push down right button	0	0	6	2	6	5	8	7
	pick up right object to right plate	0	0	4	4	5	4	6	5
	Average	0%	0%	46.7%	20%	53.3%	33.3%	**63.3%**	**46.7%**
Total Average		14.3%	5.7%	48.3%	32.3%	56.3%	39.3%	**62.6%**	**50.3%**

XII. Case Study

A. Case Study of Real-World Generation Tasks

We provide an illustration for each specific task included in the suite evaluation for *in-domain* tasks in Fig. 8 and for each type of generation task, including *subject generalization* in Fig. 9, *language grounding* in Fig. 13, *visual generalization*

TABLE V: We compared the performance of Octo-SFT, OpenVLA-SFT, and GRAPE across various robotic tasks within in-domain, subject, physical, and semantics generalization categories. It shows grasp percentages and success rates for each task, illustrating how each VLA performs under different generalizations.

Generalization	**Task**	**Octo-SFT**		**OpenVLA-SFT**		**OpenVLA-DPO**		**GRAPE**	
		Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success
In-domain	put the carrot on the plate	32.00%	16.00%	36.00%	30.00%	46.00%	36.00%	**68.00%**	**48.00%**
	put the eggplant in the basket	70.00%	44.00%	58.00%	32.00%	70.00%	36.00%	**84.00%**	**48.00%**
	stack the green cube on the yellow cube	52.00%	0.00%	56.00%	20.00%	52.00%	26.00%	**76.00%**	**40.00%**
	put the spoon on the towel	54.00%	**36.00%**	52.00%	28.00%	52.00%	30.00%	**56.00%**	34.00%
	Average	52.00%	24.00%	50.50%	28.00%	55.00%	32.00%	**71.00%**	**43.00%**
Subject Generalization (unseen objects)	put the coke can on the towel	24.00%	14.00%	60.00%	**38.00%**	66.00%	36.00%	**78.00%**	32.00%
	put the pepsi can on the towel	28.00%	16.00%	58.00%	38.00%	60.00%	42.00%	**64.00%**	**50.00%**
	put the sprite can on the towel	24.00%	12.00%	**62.00%**	22.00%	58.00%	26.00%	46.00%	**40.00%**
	Average	25.33%	14.00%	60.00%	32.67%	61.33%	34.66%	**62.67%**	**40.67%**
Physical Generalization (unseen object sizes/shapes)	put the carrot on the plate(size:0.5)	38.00%	22.00%	56.00%	38.00%	60.00%	46.00%	**78.00%**	**64.00%**
	put the carrot on the plate(size:1.1)	26.00%	12.00%	32.00%	24.00%	42.00%	30.00%	**64.00%**	**42.00%**
	put the carrot on the plate(wider collision box)	28.00%	16.00%	34.00%	26.00%	46.00%	32.00%	**62.00%**	**42.00%**
	put the carrot on the plate(longer collision box)	32.00%	14.00%	38.00%	30.00%	50.00%	36.00%	**66.00%**	**48.00%**
	put the spoon on the towel(size:0.5)	62.00%	38.00%	66.00%	**40.00%**	66.00%	38.00%	**72.00%**	38.00%
	put the spoon on the towel(size:1.1)	52.00%	**32.00%**	50.00%	28.00%	**58.00%**	**32.00%**	56.00%	30.00%
	put the spoon on the towel(wider collision box)	48.00%	30.00%	44.00%	24.00%	46.00%	28.00%	**50.00%**	**32.00%**
	put the spoon on the towel(longer collision box)	56.00%	36.00%	54.00%	26.00%	54.00%	28.00%	**60.00%**	**38.00%**
	Average	42.75%	25.00%	46.75%	29.50%	52.75%	33.75%	**63.50%**	**41.75%**
Semantics Generalization (unseen instructions)	put the vegetable on the plate	16.00%	6.00%	32.00%	28.00%	40.00%	32.00%	**66.00%**	**48.00%**
	move the eggplant into the basket	18.00%	8.00%	50.00%	30.00%	56.00%	34.00%	**78.00%**	**44.00%**
	put the green cube onto the yellow cube	32.00%	6.00%	62.00%	26.00%	74.00%	42.00%	**88.00%**	**60.00%**
	place the spoon onto the towel	42.00%	26.00%	48.00%	28.00%	48.00%	30.00%	**56.00%**	**36.00%**
	Average	27.00%	11.50%	48.00%	28.00%	54.50%	34.50%	**72.00%**	**47.00%**
Total average		36.77%	18.63%	51.44%	29.54%	55.90%	33.73%	**67.29%**	**43.11%**

in Fig. 10, *action generalization* in Fig. 11, and *semantic generalization* in Fig. 12. Specifically, we demonstrate the initial and final states of GRAPE in handling each of these challenging tasks, as detailed in the corresponding captions. In addition, we include a safety task to demonstrate how GRAPE adheres to safety requirements once aligned with safety constrains.

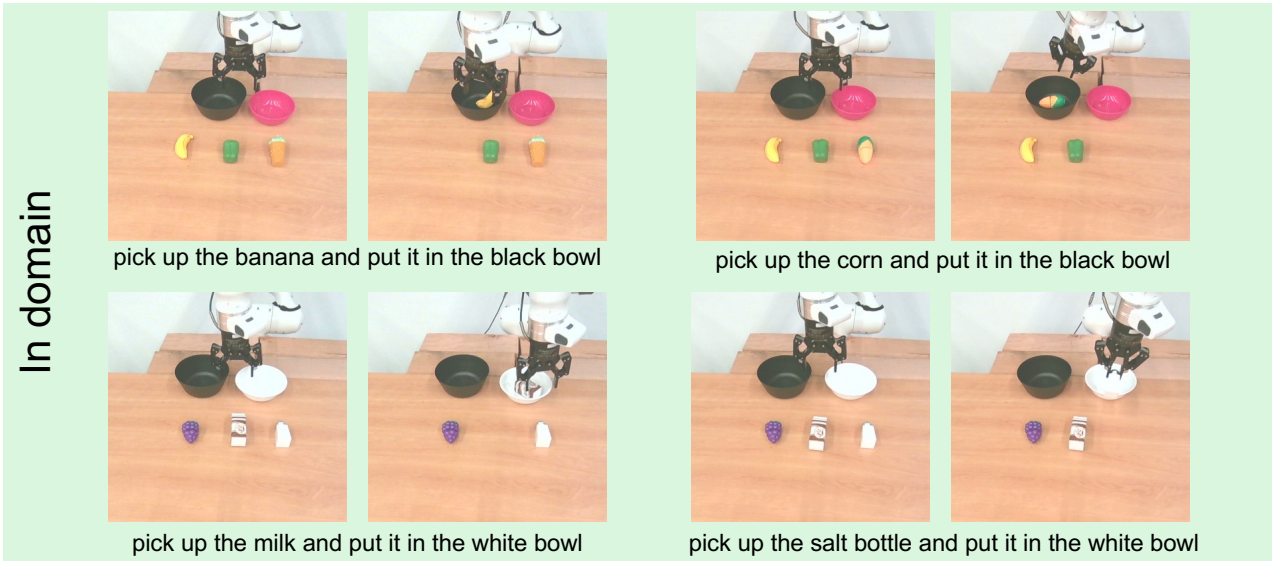

Fig. 8: Illustrations of real-world tasks that we evaluated for *in-domain capabilities*, where we report the detailed results in Table IV. Specifically, we demonstrate the initial and final state of GRAPE in handling each of the four challenging tasks detailed in the captions.

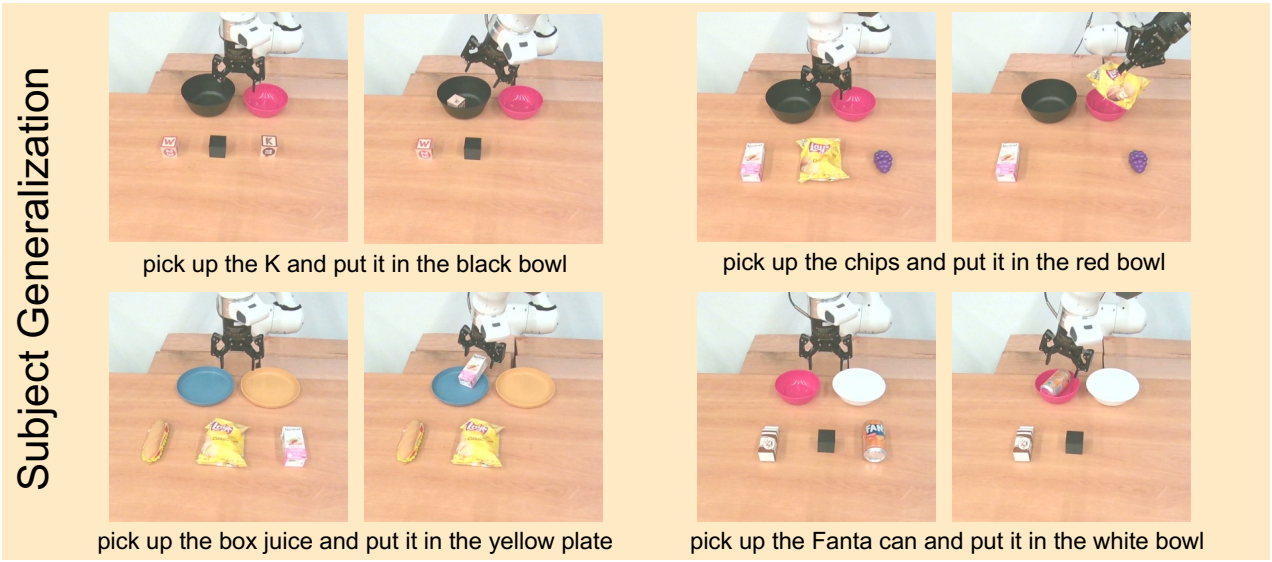

Fig. 9: Illustrations of real-world tasks that we evaluated for *subject generation*, where we report the detailed results in Table IV. Specifically, we demonstrate the initial and final state of GRAPE in handling each of the four challenging tasks detailed in the captions.

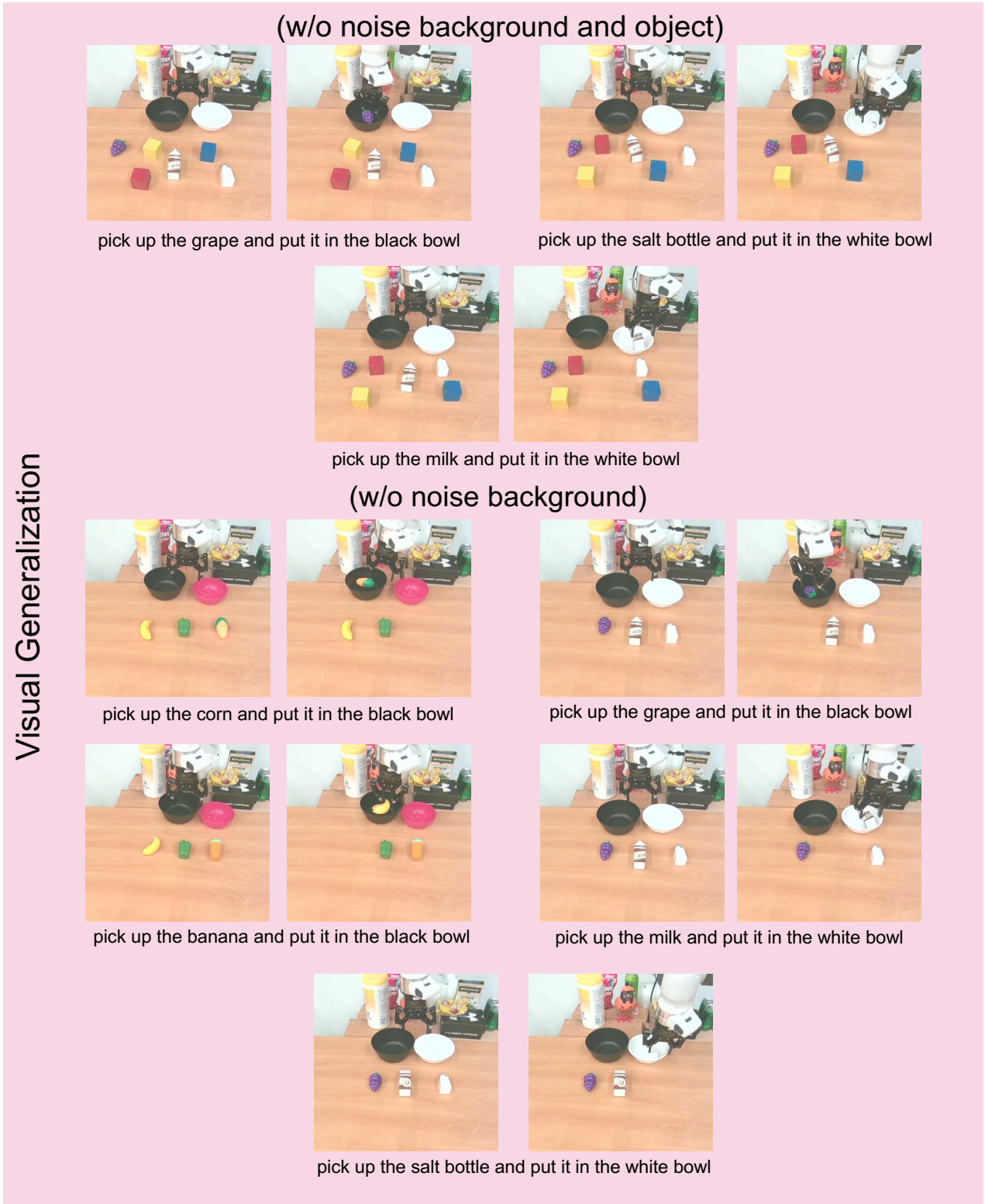

Fig. 10: Illustrations of real-world tasks that we evaluated for *visual generation*, where we report the detailed results in Table IV. Specifically, we demonstrate the initial and final state of GRAPE in handling each of the eight challenging tasks detailed in the captions.

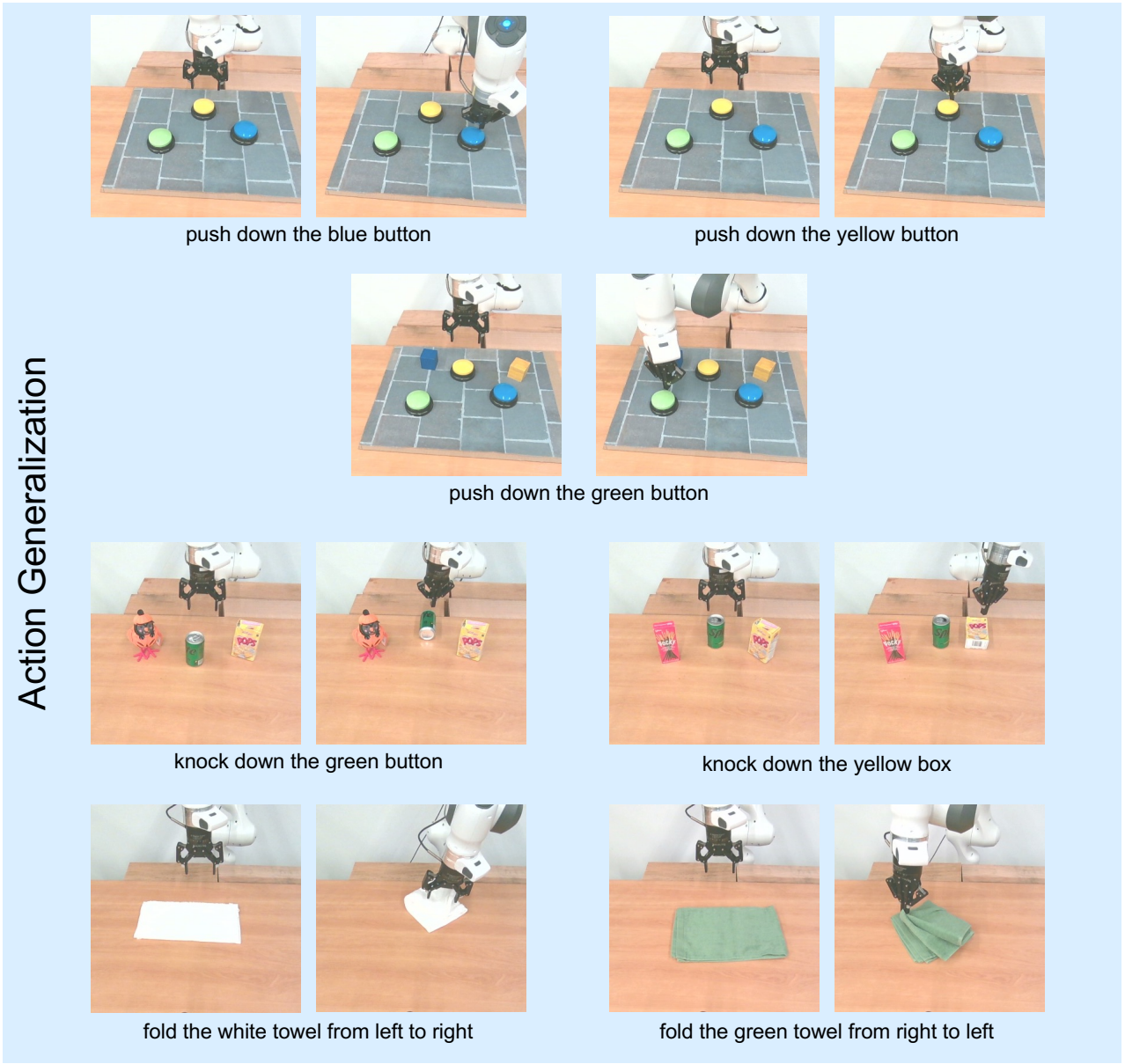

Fig. 11: Illustrations of real-world tasks that we evaluated for *action generation*, where we report the detailed results in Table IV. Specifically, we demonstrate the initial and final state of GRAPE in handling each of the seven challenging tasks detailed in the captions.

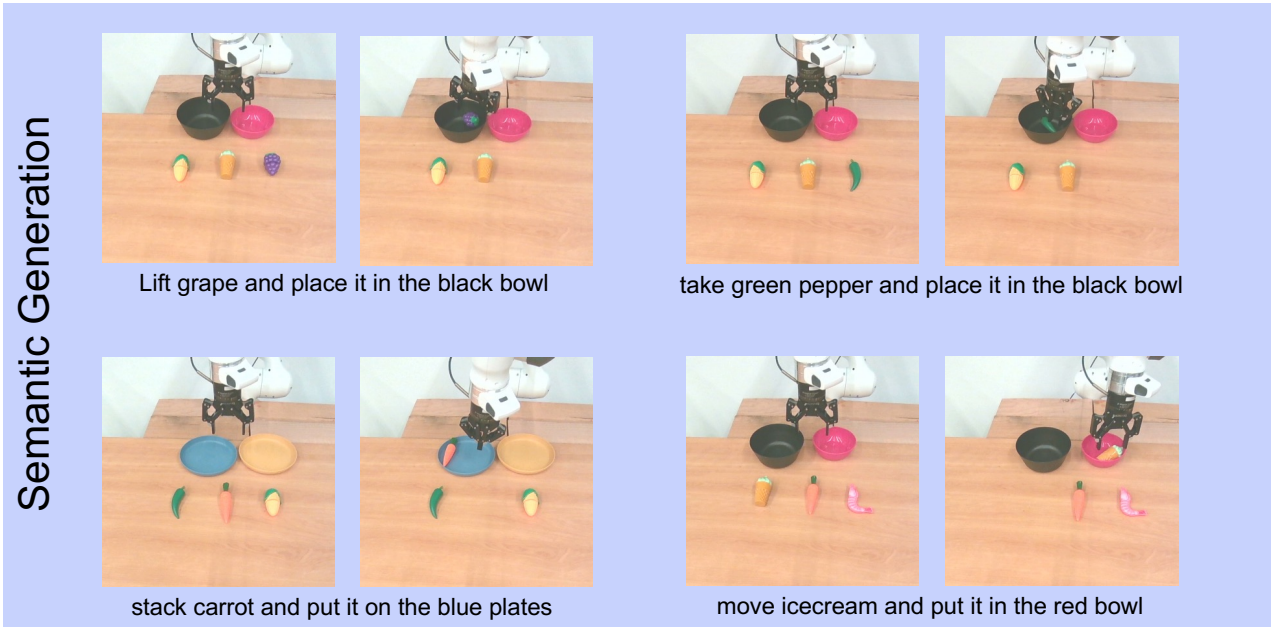

Fig. 12: Illustrations of real-world tasks that we evaluated for *semantic generation*, where we report the detailed results in Table IV. Specifically, we demonstrate the initial and final state of GRAPE in handling each of the four challenging tasks detailed in the captions.

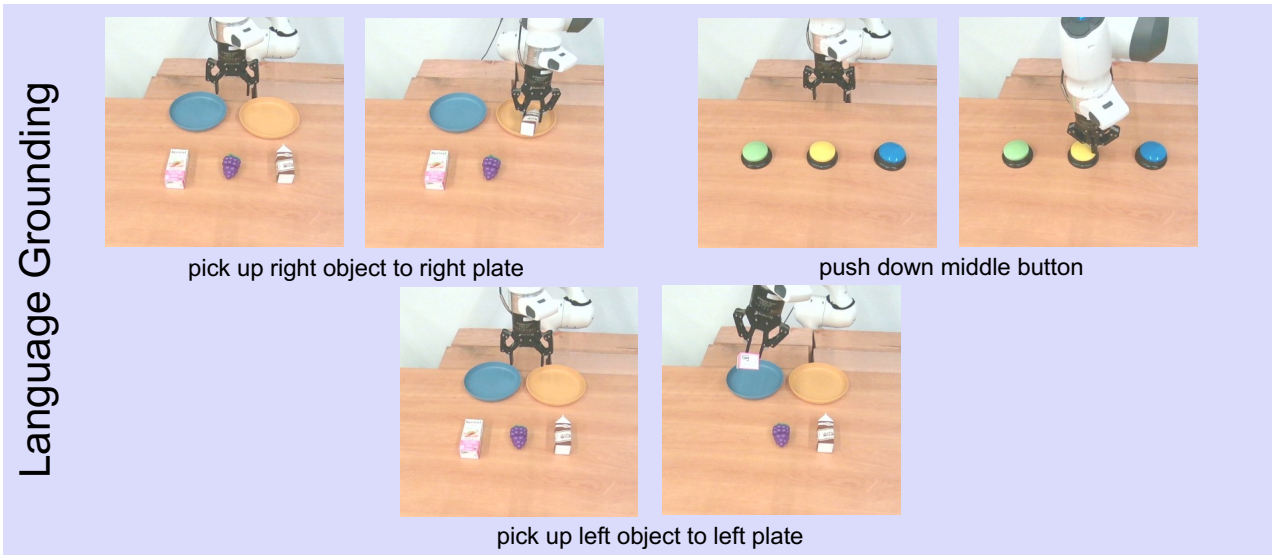

Fig. 13: Illustrations of real-world tasks that we evaluated for *language generation*, where we report the detailed results in Table IV. Specifically, we demonstrate the initial and final state of GRAPE in handling each of the five challenging tasks detailed in the captions.

Fig. 14: Illustrations of real-world tasks used for safety evaluation, extending the tasks presented in Figure 7. The figure shows key frames from GRAPE's trajectory in two challenging scenarios. Due to the lack of safety reward alignment, the OpenVLA-SFT approach fails, while GRAPE-Safety successfully navigates obstacles and completes the task once the safety rewards are properly aligned.

B. Case Study of Multi-stage Cost Functions

We demonstrate some case studies of the multi-stage cost functions generated using our proposed pipeline given different alignment objectives.

1) Task Completion

Cost Functions for Task Completion Alignment

```
# The task involves picking up the grape and placing it in the black bowl.
# The stages involved are:
# 1. Grasp grape
# 2. Move grape to black bowl
# 3. Drop grape in black bowl

num_stages = 3

### stage 1: Grasp grape

def stage1_target_constraint1(end_effector, keypoints):
    """Align the end-effector with the grape's center."""

    grape_center = keypoints[0]
    target_cost = np.linalg.norm(end_effector - grape_center)
    return target_cost

### stage 2: Move grape to black bowl

def stage2_target_constraint1(end_effector, keypoints):
    """Calculate the relative distance between grape and black bowl."""

    black_bowl_center = keypoints[1]# Assuming keypoint 1 is the black bowl
    target_cost = np.linalg.norm(end_effector - black_bowl_center)
    return target_cost

### stage 3: Drop grape in black bowl

def stage3_target_constraint1(end_effector, keypoints):
    """Ensure the grape rests in the black bowl."""

    black_bowl_center = keypoints[1]
    target_cost = np.linalg.norm(end_effector - black_bowl_center)
    return target_cost
```

2) Safety

Cost Functions for Cost-Efficiency Alignment

```
# The task involves picking up the grape and placing it in the black bowl.
# The stages involved are:
# 1. Grasp grape
# 2. Move grape to black bowl
# 3. Drop grape in black bowl

num_stages = 3

### stage 1: Grasp grape

def stage1_collision_constraint1(end_effector, keypoints):
```

```
    """Approach the grape from above to avoid collision."""

    grape_center = keypoints[0]
    collision_cost = 0 if end_effector[1] > grape_center[1] else 1
    return collision_cost

### stage 2: Move grape to black bowl

def stage2_collision_constraint1(end_effector, keypoints):
    """Ensure the grape is aligned above the black bowl."""

    obstacles = keypoints[2:]#Assuming keypoints[2:] are obstacles
    threshold = 0.1  # Minimum allowable clearance
    collision_cost = sum(
        max(0, threshold - np.linalg.norm(end_effector - obstacle))
        for obstacle in obstacles
    )
    return collision_cost

### stage 3: Drop grape in black bowl

def stage3_collision_constraint1(end_effector, keypoints):
    """Approach the grape from above to avoid collision."""

    black_bowl_center = keypoints[1]
    collision_cost = 0 if end_effector[1] > black_bowl_center[1] else 1
    return collision_cost
```

3) Cost-Efficiency

Cost Functions for Safety Alignment

```
# The task involves picking up the grape and placing it in the black bowl.
# The stages involved are:
# 1. Grasp grape
# 2. Move grape to black bowl
# 3. Drop grape in black bowl

num_stages = 3

### stage 1: Grasp grape

def stage1_path_constraint1(end_effector, keypoints):
    """Align the end-effector with the grape's center."""

    grape_center = keypoints[0]
    distance = np.linalg.norm(end_effector - grape_center)
    step_size = 0.01  # Assuming a small step size
    path_cost = int(distance / step_size)
    return path_cost

### stage 2: Move grape to black bowl

def stage2_path_constraint1(end_effector, keypoints):
    """Calculate the relative distance between grape and black bowl."""
```

```
    black_bowl_center = keypoints[1]# Assuming keypoint 1 is the black bowl
    distance = np.linalg.norm(end_effector - black_bowl_center)
    step_size = 0.01  # Assuming a small step size
    path_cost = int(distance / step_size)
    return path_cost

### stage 3: Drop grape in black bowl

def stage3_path_constraint1(end_effector, keypoints):
    """Ensure the grape rests in the black bowl."""

    black_bowl_center = keypoints[1]
    distance = np.linalg.norm(end_effector - black_bowl_center)
    step_size = 0.01  # Assuming a small step size
    path_cost = int(distance / step_size)
    return path_cost
```

Prompt Template for Multi-stage Cost Proposal

USER: *Instructions*

The image shows a robot stage point in a workspace, each point in the diagram represents the point of the stage split:

- Stage_point_0 : Represents the initial position of the carrot.
- Stage_point_1 : Represents the intermediate position above the carrot for grasping.

Determine how many stages are involved in the task. Grasping must be an independent stage. Some examples:

1. Task: Put the carrot on the plate

a) Stages:

- **Grasp carrot**
- **Move carrot to plate**
- **Drop carrot on plate**

b) Stage 1: Grasp carrot

- **Path constraints**:
  - Align the end-effector with the carrot's center.
- **Collision constraints**:
  - The end-effector must approach the carrot from above to avoid collision.

c) Stage 2: Move carrot to plate

- **Path constraints**:
  - Calculate the relative distance between carrot and plate.
- **Collision constraints**:
  - The carrot is aligned above the plate.

d) Stage 3: Drop carrot on plate

- **Path constraints**:
  - The carrot must rest on the plate.
  - The carrot should not bounce out of the basket.
- **Collision constraints**:
  - The end-effector must approach the carrot from above to avoid collision.

Note:

- Sum all Path constraints cost the `path_cost` variable.
- Sum all Grasp constraints cost the `grasp_cost` variable.
- Sum all Collision constraints cost the `collision_cost` variable.
- Each constraint function takes an end-effector point and a set of keypoints as input, returning a numerical cost. The constraint is satisfied if this cost is zero or less.
- Define any number of path constraints per stage, but avoid using "if" statements in the functions.
- Avoid using path constraints when manipulating deformable objects (e.g., towels).
- Input format:
  - `end_effector`: `np.array` of shape `(3,)` representing the end-effector position.
  - `keypoints`: `np.array` of shape `(K, 3)` representing the keypoints positions.
- Use Python and NumPy functions freely in constraint functions.
- Use pairs of keypoints to create vectors if needed.
- Keypoints are indexed starting from 0, matching their order in the keypoints array.

Structure your output in a single Python code block as follows:

```
# ...

num_stages = ?

### stage 1 path constraints (if any)
def stage1_path_constraint1(end_effector, keypoints):
```

```
        """Put your explanation here."""
        ...
        return path_cost
    # Add more constraints if needed
    ...

    ### stage 1 collision constraints (if any)
    def stage1_collision_constraint1(end_effector, keypoints):
        """Put your explanation here."""
        ...
        return collision_cost

    # Add more constraints if needed
    ...

    # Repeat for more stages
    ...
```

Query
Query Task: ”{instruction}”
Query Image: