A PROOF FOR THEOREM

The number of plan-execution misalignments follows a Poisson distribution (Papoulis & Unnikrishna Pillai, 2002):

$$P(M=k)=\frac{(\lambda t)^k e^{-\lambda t}}{k!} \quad k=0,1,2,3... \tag{4}$$

Our analysis considers two types of misalignments: **(1) Soft misalignment**: If this occurs at time s, an agent without immediate re-planning must recover to the stage at time s, wasting time from s to t (e.g., a robot that drops an object must return to the drop location to pick it up). We assume DoReMi can detect this misalignment within Δt second and recover immediately. **(2) Critical misalignment**: If this occurs at time s, a delayed re-planning invariably results in failure; only immediate re-planning can address this misalignment (e.g., unexpected obstacles appear in front of the robot).

Theorem 1 *The following equations describe the wasted time t_w under soft misalignment and the failure probability P_f under critical misalignment without immediate detection and re-planning:*

$$\mathbb{E}(t_w)=\sum_k P(M=k)\mathbb{E}(t_w|M=k)=\frac{\lambda(\mu^2+\sigma^2)}{2}-\lambda\mu\Delta t \tag{5}$$

$$\mathbb{E}(P_f)=1-\mathbb{E}(e^{-\lambda t})\approx\lambda\mu-\frac{\lambda^2(\mu^2+\sigma^2)}{2} \tag{6}$$

A.1 LEMMA

Lemma 1 *Given a Poisson process which is conditional on n arrivals in the time interval $(0,t)$, the conditional pdf (probability density function) of event occurrence time $t_1, t_2..., t_n$ satisfy (Papoulis & Unnikrishna Pillai, 2002):*

$$f(t_1,...,t_n|M(t)=n)=\frac{n!}{t^n} \quad 0\le t_1\le ...\le t_n\le t \tag{7}$$

Proof Since the inter-arrival times of Poisson distribution are independent exponentially distributed, the joint pdf of the n first arrival times is:

$$\begin{aligned} f(t_2,t_2,...,t_n)&=f(t_1)f(t_2|t_1)...f(t_n|t_{n-1})\\ &=\lambda e^{-\lambda t_1}\lambda e^{-\lambda(t_2-t_1)}...\lambda e^{-\lambda(t_n-t_{n-1})}\\ &=\lambda^n e^{-\lambda t_n}\end{aligned} \tag{8}$$

And conditional pdf can be derived:

$$\begin{aligned} f(t_1,t_2,...,t_n|M(t)=n)&=\frac{f(t_2,t_2,...,t_n,M(t)=n)}{P(M(t)=n)}\\ &=\frac{f(t_1,t_2,...,t_n)P(M(t)=n|t_1,t_2,...,t_n)}{P(M(t)=n)}\\ &=\frac{\lambda^n e^{-\lambda t_n}e^{-\lambda(t-t_n)}}{e^{-\lambda t}(\lambda t)^n/n!}\\ &=\frac{n!}{t^n}\end{aligned} \tag{9}$$

That is to say, each event can be considered as "placed" independently and uniformly at a given time in $[0,t]$.

A.2 PROOF FOR THEOREM 1

Soft Misalignment Based on lemma A.1, each event can be considered to occur independently and uniformly at a given time in $[0,t]$, the event that occurs at s will lead to time cost $t-s$. Total time cost without immediate replanning ($E(M) = \sum P(M = k) * k = \lambda t$):

$$\begin{aligned}\mathbb{E}(t_{delay}) &= \sum_{t_i}(t - t_i) = \sum_k P(M = k)\mathbb{E}_t(t_w|M = k) \\ &= \mathbb{E}_t[\sum_k P(M = k) * kt/2] \\ &= \mathbb{E}_t[\lambda t^2/2] = \lambda(\mu^2 + \sigma^2)/2\end{aligned} \tag{10}$$

Time cost without immediate replan (every event have detection time Δt):

$$\mathbb{E}(t_{doremi}) = \sum_{t_i} \Delta t = \mathbb{E}_t[M] * \Delta t = \mathbb{E}_t[\lambda t \Delta t] = \lambda \mu \Delta t \tag{11}$$

The wasted time $\mathbb{E}(t_w)$ is the difference between $\mathbb{E}(t_{delay})$ and $\mathbb{E}(t_{doremi})$:

$$\mathbb{E}(t_w) = \mathbb{E}(t_{delay}) - \mathbb{E}(t_{doremi}) = \frac{\lambda(\mu^2 + \sigma^2)}{2} - \lambda\mu\Delta t \tag{12}$$

Critical misalignment Once critical misalignment comes, a delayed replanning will lead to failure. So the failure ratio P_f equals to the probability that the misalignment occurrence number is greater than 1. We assume misalignment happen ratio λ is very small and we use second-order Tyler expansion to approximate the original equation.

$$\begin{aligned}\mathbb{E}(P_f) &= \mathbb{E}_t[\sum_{k\geq 1} P(M = k)] = \mathbb{E}_t[1 - P(M = 0)] = 1 - \mathbb{E}_t(e^{-\lambda t}) \\ &= 1 - \mathbb{E}_t(1 - \lambda t + \lambda^2 t^2/2 + ...) \approx \lambda\mathbb{E}_t(t) - \lambda^2\mathbb{E}_t(t^2) \\ &= \lambda\mu - \frac{\lambda^2(\mu^2 + \sigma^2)}{2}\end{aligned} \tag{13}$$

B HUMANOID ROBOT TASK

B.1 BASIC HUMANOID ROBOT INFORMATION

Our robot has 21 links and 20 degrees of joint freedom(DOF), and each joint holds a corresponding motor.

Link names: "base", "left shoulder pitch", "left shoulder roll", "left arm", "left elbow", "right shoulder pitch", "right shoulder roll", "right arm" "right elbow", "left leg yaw", "left leg roll", "left leg pitch", "left knee", "left ankle roll", "left ankle pitch" "right leg yaw", "right leg roll", "right leg pitch", "right knee", "right ankle roll","right ankle pitch"

DOF joint names: "left shoulder pitch", "left shoulder roll", "left arm", "left elbow", "right shoulder pitch", "right shoulder roll", "right arm" "right elbow", "left leg yaw", "left leg roll", "left leg pitch", "left knee", "left ankle roll", "left ankle pitch" "right leg yaw", "right leg roll", "right leg pitch", "right knee", "right ankle roll","right ankle pitch"

B.2 LOW-LEVEL SKILL TRAINING

Controlling complex humanoid robots with a single policy is challenging. Thus, we train low-level skills at the category level. Following the separate framework in (Ma et al., 2022), we utilize reinforcement learning to train locomotion policy and use model-based methods to obtain manipulation policy. In the case of our humanoid robot, there are 12 motors dedicated to the legs and 8 motors allocated to the arms. Notably, the observation of arm motors is not incorporated into the locomotion policies.

The locomotion policy is responsible for directly controlling the 12 motors associated with the legs, leading to a 12-dimensional action space. These policies output the target position of motors and run at 50 Hz, followed by PD controller run at 1000 Hz with $k_p = 100$ and $k_d = 2.5$. The proprioceptive observation space of the robot includes various dimensions: 12-dimensional joint angles, 12-dimensional joint angular velocities, 12-dimensional last actions, 3-dimensional angles between the torso and gravity, 2-dimensional periodic clock signals, and reserved 3-dimensional command signals, resulting in a total of 44 basic observation spaces.

We train low-level skills with the Deepmimic algorithm based on the Legged Gym (Isaac Gym Environments for Legged Robots) environment `https://github.com/leggedrobotics/legged_gym` built with the Isaac Sim physics simulator. Motion capture data we used can be found in the poselib `https://github.com/NVIDIA-Omniverse/IsaacGymEnvs/tree/main/isaacgymenvs/tasks/amp/poselib`.

Locomotion Policy This neural network policy is conditioned on 3-dimensional commands which respectively represent the required velocity in x-direction, y-direction, and required yaw angular velocity. In order to obtain natural moving gaits, we use the Deepmimic algorithm with multiple Motion Capture Date(Mocap data), thus reward function has 2 parts including tracking commanded linear/angular velocity and imitating the style of Mocap data. This learned policy can help robots realize a category of sub-skills related to locomotion like: "Go forward fast", "Go forward at speed v", "Stand still", "Turn right/left", "Go to target place A", etc.

Arm Manipulation Policy Since we separate the control of arm and leg, we can use various manipulation policies including learned neural network policy or model-based policy, without influencing the leg locomotion policy. We use a linear interpolation controller to achieve the skill: "Pick up box", "Pick up box on the floor", "Put box on table", etc.

Hyperparameters Deepmimic algorithm pipeline is similar to PPO. Hyperparameters of the backbone Deepmimic algorithm can be found in table 3.

Parameters	Value
Number of Environments	4096
Learning epochs	5
Steps per Environment	24
Minibatch Size	24576
Episode length	20 seconds
Discount Factor	0.99
Generalised Advantage Estimation(GAE)	0.95
PPO clip	0.2
Entropy coefficient	0.005
Desired KL	0.01
Learning Rate	5e-4
Weight decay	0.01

Table 3: Hyperparameters of backbone PPO algorithm.

Training curves The training process for the navigation policies and stand/squat policies is illustrated in Figure 8. The navigation policy enables the robot to control its xy position within the world frame, while the height switch policy allows for adjusting the robot's z height within the world frame.

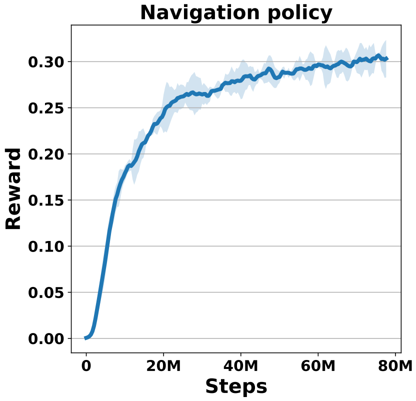


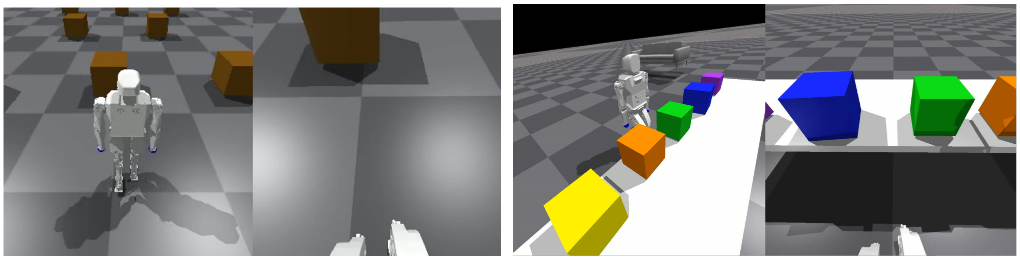
Figure 8: Traning curves for navigation policy by Deepmimic algorithm.

B.3 TASK DETAILS

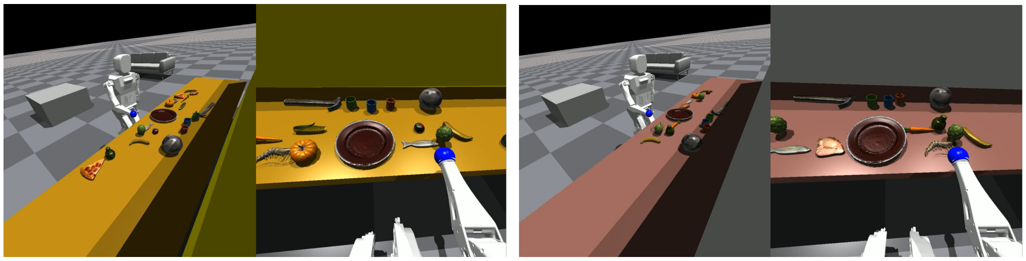
Obstacle-avoidance task: The finish line is 8m-16m far away from the robot. The task instruction is "Reach the finish line which is 15m in front of you". And the robot will plan low-level skills "go forward 15m". However, unknown obstacles may appear in the way and the robot needs to replan the skill "turn right/left" to avoid collision.

Move-box task: The robot is required to move the box in a certain color from place A to place B. The box color is randomly selected from {red, yellow, orange, green, blue, purple}.

Prepare food: The robot is required to collect 2-5 types of food. The food is in random positions. The robot has a chance to fail to pick up the food and the food may drop from the end-effector during transportation.



(a) Reach line (b) Move box



(c) Prepare food (d) Prepare food

Figure 9: Robot and tasks. For every task, the photo on the Left is recorded from a third-view camera and the photo on the right is from the first-view camera. We use first-view images as inputs to VLM models.

B.4 VISION LANGUAGE MODEL FINETUNING

Vision language model input images are the first-view camera attached to the robot. Visualization of this camera can be found in Figure 9. Zero-shot transferred BLIP-2 model performs well in the obstacle-avoidance task and the move-box task. However, in complex prepare-food tasks, the accuracy of VLM detector drops significantly. To better handle complex tasks, we finetune BLIP-2 ViT-g FlanT5XL model (Li et al., 2023b) with 4.1B parameter on prepare food task, we use LoRA (Hu et al., 2021) method to finetune the parameters in an efficient war.

Train dataset collection: we collect 5 demonstrations trajectory which are 128 seconds long. Then we label the data every 1 second, which result in a total of 128 image-text pairs. The demonstrations only contain fruit objects on a white plain table without decorations, as shown in Figure 10.

Object in training dataset: apple, green apple, strawberry, banana, orange, lemon.

Unseen objects: carrot, chicken, corn, fish, meat, peach, pear, pizza, pumpkin, shrimp, vegetable, soccer, cups, red plate, hammer.

Unseen background: The backgrounds are randomly generated colors, 3 samples are shown in Figure 10.

Unseen tasks: obstacle-avoidance and move-box are two unseen tasks. Compared to the zero-shot transferred BLIP-2 model, the finetuned BLIP-2 model performs better in move-box tasks and performs similarly in obstacle-avoidance tasks. Finetuned BLIP-2 model can discover box dropped more quickly.

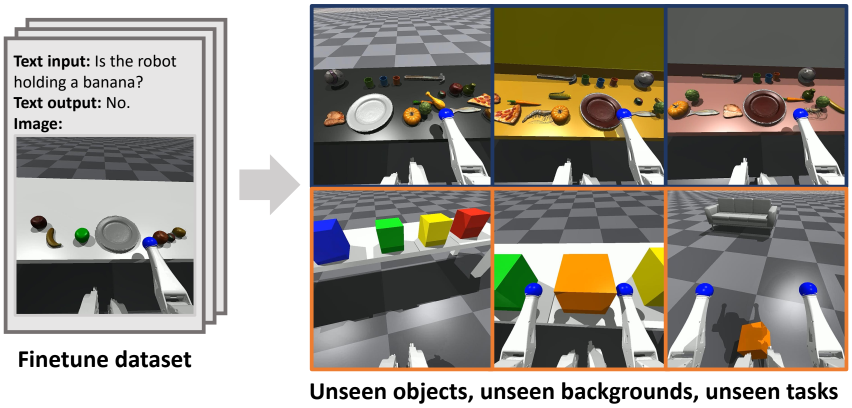
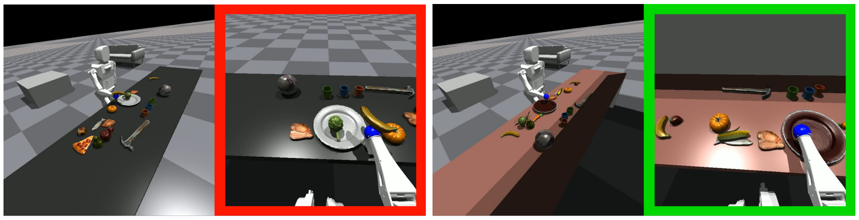

Text input: Is the robot holding a banana?
Text output: No.
Image:
Finetune dataset
Unseen objects, unseen backgrounds, unseen tasks

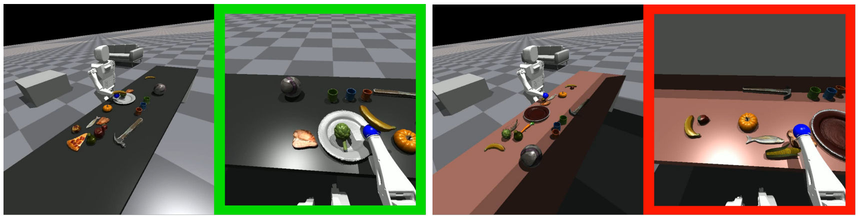
Figure 10: Train dataset only contains fruit objects on plain white tables. And the test tasks are much more complicated with unseen objects, backgrounds, and tasks.

Ensemble VLM Answers: The VLM text output for a single vision-question pair may not always be correct. To reduce the probability of true-negative(TN) samples, an ensemble approach can be utilized by incorporating 2 consecutive frames. Practically, We detect constraint violations by considering 2 consecutive time-step images where VLM identifies the same constraint violation. The time step duration, denoted as Δt, is set to 0.2 seconds.

Fine-tuned VLM can benefit unseen objects and unseen backgrounds. First, we compare the zero-shot transferred BLIP-2 model and fine-tuned BLIP-2 model in unseen objects and backgrounds. Zero-shot transferred BLIP-2 model may fail to detect a drop during transportation and lead to low task success rates. A fine-tuned BLIP-2 model can detect constraint violation with high accuracy and lead to high task success rates. Some comparisons are shown in Figure 11. we use a green border around the image to indicate that our VLM detector determines there is no constraint violation in the image, and a red border to indicate that VLM believes there is a constraint violation in the image.



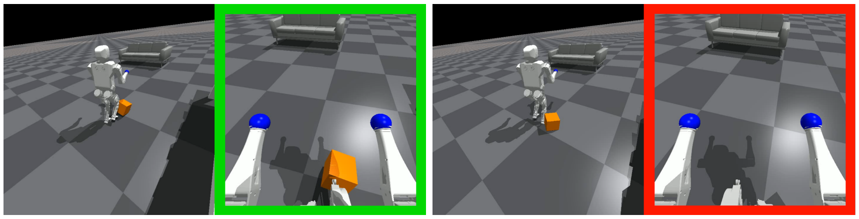
(a) Wrong detection of zero-shot transferred BLIP-2 model.



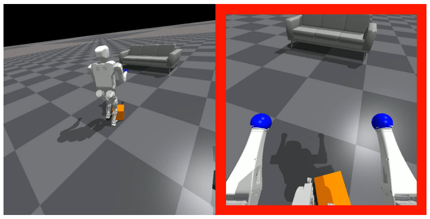
(b) Correct detection of finetuned BLIP-2 model on unseen objects and background

Figure 11: Fine-tuned VLM performs well in unseen objects and unseen backgrounds.

Fine-tuned VLM can benefit unseen tasks. We also compare the zero-shot transferred BLIP-2 model and fine-tuned BLIP-2 model with unseen tasks. In move-box tasks, the zero-shot transferred BLIP-2 model can detect the box drop when the box is totally dropped on the floor or even disappears in the camera. However, the finetuned BLIP-2 model seems to figure out the correct robot body part and can detect box drop as soon as the box leaves the robot arm, as shown in Figure 12. The average detection time decreased from 2.5 seconds to 0.4 seconds.



(a) Zero-shot transferred BLIP-2 identify box drop until box disappear in the horizon



(b) Fine-tuned BLIP-2 immediately detect the box drop in this unseen task.

Figure 12: Fine-tuned VLM can benefit unseen tasks

Accuracy analysis

After fine-tuning, the accuracy of VLM in the prepare-food task has significantly increased, as shown in Table 4.

	Before finetune				After finetune			
	TP	TN	FP	FN	TP	TN	FP	FN
Obstacle	120	5	0	14	121	4	0	14
Move box	140	0	6	22	140	0	2	26
Prepare food	78	27	8	25	99	6	1	32

Table 4: Accuracy analysis of VLM on humanoid tasks.

VLM ablation study We also conduct an ablation study on different types of VLM with max disturbances in our tasks, as shown in Table 5. BLIP-2 model performs similarly to the Instruct-BLIP model. However, all zero-shot transferred models can not perform well in complicated prepare-food tasks.

Success rate%	**BLIP-1**	**BLIP-2**	**Instruct-BLIP**
Obstacle-avoidance	88	90	92
Move-box	64	94	92
Prepare-food	16	37	40

Table 5: Ablation study on zero-shot transferred VLM

C ROBOT ARM MANIPULATION TASK

C.1 IMPLEMENTATION DETAILS

Low-level Policy The low-level policy is similar to CLIPort(Shridhar et al., 2022) and Transporter Network (Zeng et al., 2020). Detailed references for its implementation can be found at `https://github.com/google-research/ravens`. This policy has been trained to perform single-step pick-and-place tasks based on language descriptions, and its performance is nearing perfection. However, for the purpose of our study, we presume this original policy to be perfect and introduce additional perturbations to the location placement.

Ensembling Multi-step Detection by VLM In the robot arm manipulation tasks, we use the zero-shot transferred BLIP-2 model. To decrease the true-negative error of the VLM detector, We detect constraint violations by considering 2 consecutive time-step images where VLM identifies the same constraint violation.

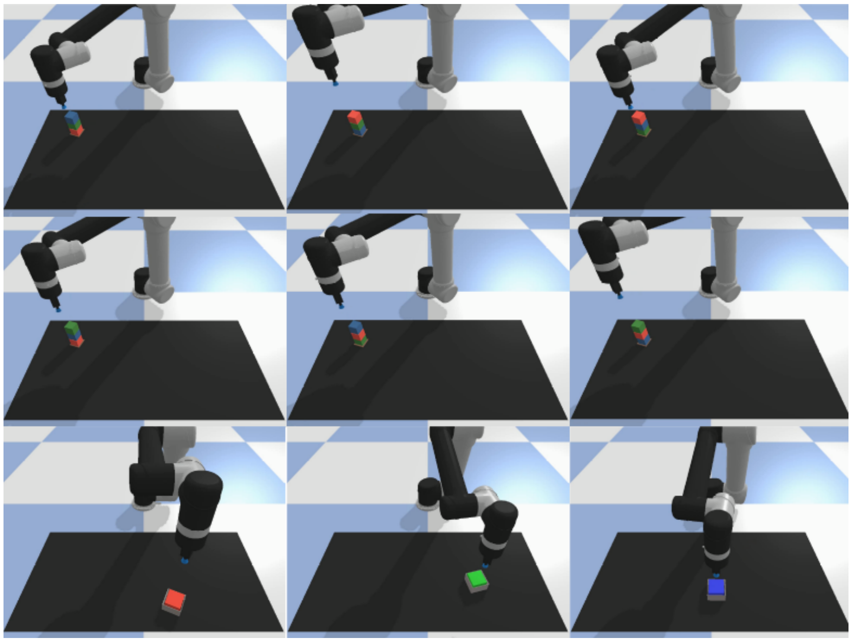


Figure 13: Our method is agnostic to different stack orders.

Baseline To adapt to our tasks, we slightly modify the original implementation of three baselines: (1) SayCan: similar to the original implementation of SayCan `https://github.com/google-research/google-research/tree/master/saycan`, we don't use a value function here given this environment does not have RL-trained policies or an associated value function. Instead, we use affordances obtained from ground-truth object information. The low-level policies are adopted as the same as ours. We use Vicuna-13B (Chiang et al., 2023) to output the probabilities of each low-level policy in each scene. (2) CLIPort: the implementation is based on `https://github.com/google-research/ravens`. The oracle success detector is replaced with our VLM detector. (3) Inner Monologue: We reproduce the implementation based on (Huang et al., 2022b). Both the low-level policies and the LLM planner are the same as ours. The original success detector and the scene descriptor are also replaced with our VLM.

C.2 ABLATION STUDY

We evaluated the robustness of DoReMi in various environmental conditions by testing our method under distinct levels of perturbations. We observed that when the positional noise level n of the end-effector exceeds 0.03 cm, the frequency of constraint violations escalates, leading to an almost zero success rate and dramatically increased execution time, despite accurate detection of constraint

Tasks with disturbance		**Success Rate(%) ↑** **Inner Monologue**	 **DoReMi (w/o ensembling)**	 **DoReMi (ours)**	**Execution Time(s) ↓** **Inner Monologue**	 **DoReMi (w/o ensembling)**	 **DoReMi (ours)**
Stack in order with noise n random drop p=0.05	**n=0.0**	**100 (±0)**	**100 (±0)**	**100 (±0)**	7.9 (±0.7)	7.5 (±0.5)	**7.4 (±0.5)**
	n=1.0	94 (±7)	96 (±4)	**98 (±4)**	9.3 (±3.3)	8.6 (±2.9)	**8.1 (±1.0)**
	n=2.0	83 (±8)	88 (±7)	**94 (±7)**	17.3 (±5.8)	12.1 (±2.9)	**10.8 (±2.7)**
	n=3.0	63 (±9)	67 (±10)	**73 (±11)**	36.3 (±7.2)	25.8 (±7.1)	**19.9 (±3.9)**
Stack in order with noise n random drop p=0.15	**n=0.0**	92 (±6)	92 (±6)	**94 (±7)**	10.6 (±4.3)	9.7 (±3.2)	**8.9 (±2.2)**
	n=1.0	88 (±7)	90 (±7)	**92 (±6)**	14.8 (±5.1)	12.5 (±4.2)	**10.3 (±3.2)**
	n=2.0	73 (±11)	79 (±9)	**85 (±7)**	25.2 (±6.3)	21.3 (±5.7)	**14.0 (±3.7)**
	n=3.0	23 (±9)	33 (±10)	**44 (±11)**	47.8 (±6.5)	40.6 (±6.9)	**29.3 (±4.1)**

Table 6: Ablation study over different degrees of perturbations and whether adopt ensembling or not. The results show the mean and standard deviation over 4 different seeds each with 12 episodes.

violations. This observation aligns with the theoretical expectation derived from a simple computation of block placement probabilities.

Our primary interest lies in the response of DoReMi to a spectrum of drop perturbation levels, in addition to the $p = 0.1$ presented in Table 1. As illustrated in Table 6, DoReMi demonstrates admirable performance under a variety of scenarios, outperforming the best-performing baseline. We attribute this largely to DoReMi's robust detection mechanism and its ability to swiftly recover from misalignment between plan and execution.

We are also curious to explore how the ensembling of multi-step detection would perform under various environmental settings. As indicated in Table 6, our findings suggest that ensembling can markedly enhance DoReMi's effectiveness in scenarios where strong perturbations exist and a single detection error could potentially result in a complete episode failure.

C.3 VISION LANGUAGE MODEL ANALYSIS

Accuracy analysis: To analyze the accuracy of the VLM detector, we categorize all the detection results into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN), using these to calculate relevant accuracy metrics as outlined in Table 7.

A True Positive (TP) refers to the VLM correctly identifying that no constraint violation has occurred, whereas a True Negative (TN) signifies a successful detection of a constraint violation. A False Positive (FP) is when the VLM fails to recognize a constraint violation, and a False Negative (FN) is when the VLM incorrectly identifies a normal condition as a violation.

Utilizing the count in each of these categories, we compute the True Positive Rate (TPR) as $TPR = \frac{TP}{TP+FN}$. TPR reflects the accuracy with which the VLM identifies normal conditions. Similarly, the True Negative Rate (TNR) is calculated as $TNR = \frac{TN}{TN+FP}$, representing the accuracy of VLM in detecting constraint violations.

Further, we determine the Positive Prediction Value (PPV) as $PPV = \frac{TP}{TP+FP}$, and the Negative Prediction Value (NPV) as $NPV = \frac{TN}{TN+FN}$. These metrics correspond to the precision of the VLM detector in identifying normal conditions and constraint violations, respectively.

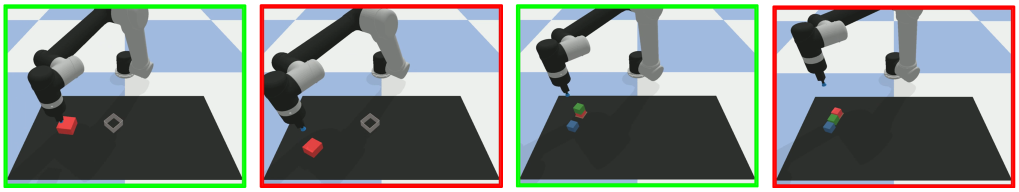
As per Table 7, both TPR and PPV maintain high values across various settings, which suggests that the VLM detector excels at identifying normal conditions. However, TNR is typically lower, particularly under conditions of low perturbations, indicating that the VLM detector may not be adept at detecting all constraint violations. The fluctuating detections become particularly pronounced when violations are infrequent. Similarly, NPV also trends lower across settings, signifying that our VLM detector might misidentify normal conditions as constraint violations at times, leading to redundant re-planning efforts.

C.4 CASE VISUALIZATION

Similar to B.4, we use a green border around the image to indicate that our VLM detector determines there is no constraint violation in the image, and a red border to indicate that VLM believes there is a constraint violation in the image.

Stack-block-in-order		TP	TN	FP	FN	TPR	TNR	PPV	NPV
p=0	n=0	150	0	0	0	1.00	N/A	1.00	N/A
	n=1	188	0	0	4	0.98	N/A	1.00	0.00
	n=2	249	27	3	6	0.98	0.90	0.99	0.82
	n=3	272	81	1	2	0.99	0.99	1.00	0.98
p=0.05	n=0	173	5	0	0	1.00	1.00	1.00	1.00
	n=1	204	7	1	1	1.00	0.88	1.00	0.88
	n=2	196	23	3	4	0.98	0.88	0.98	0.85
	n=3	253	92	2	6	0.98	0.98	0.99	0.94
p=0.1	n=0	202	13	1	1	1.00	0.93	1.00	0.93
	n=1	180	8	0	1	0.99	1.00	1.00	0.89
	n=2	212	14	3	1	1.00	0.82	0.99	0.93
	n=3	231	100	4	7	0.97	0.96	0.98	0.93
p=0.15	n=0	182	24	1	0	1.00	0.96	0.99	1.00
	n=1	178	12	2	0	1.00	0.86	0.97	1.00
	n=2	175	26	1	3	0.98	0.96	0.99	0.90
	n=3	208	128	6	9	0.96	0.96	0.97	0.93

Table 7: Statistics of VLM detection. The number of results of TP, TN, FP, FN are summed over 4 different seeds each with 12 episodes.



(a) **Q**:Is the robot holding the red block? **A**:Yes (b) **Q**:Is the robot holding the red block? **A**:No (c) **Q**:Is the green block on the red block? **A**:Yes (d) **Q**:Is the green block on the red block? **A**:No

Figure 14: Case visualization for robot arm experiment.

D PROMPTS AND PIPELINE

We provide the prompt and the whole pipeline in this section. These prompts enable LLMs to generate both the next step and constraint through few-shot in-context learning. In practice, the n^{th} generated constraint is used for $(n+1)^{th}$ skill, we find this way is more natural for LLM and LLM can generate more admissible constraints.

D.1 QUALITY OF LLM GENERATED CONSTRAINTS

Previous works demonstrate LLM can finish high-level planning in high quality. Additionally, we conducted an experiment to analyze the quality the constraints generated by LLMs.

Constraint admissible rate. We first conducted a user study to compare LLM-generated constraints and manually specified constraints. We sought the input of five individuals to assess the admissibility of these constraints, considering a constraint as admissible if at least four out of the five people reached a consensus. Our findings revealed the following: For 83 specific skills used in our tasks, LLM-generated constraints had a 98% admissible rate; For 50 random skills out of our tasks (like "open fridge" or "give milk to human"), they had a 94% admissible rate. These results underscore the remarkable proficiency of LLMs in generating constraints, driven by their profound understanding of the physical world. Some examples are shown in Table 8.

Constraint consistent rate. We then query the VLM with LLM-generated constraints and manually specified constraints under the same image input. The answers of the VLM under these two types of constraint inputs reach a consistency rate of **97%**, which proves that the LLM is able to generate very high-quality constraints and thus can be used for constraint generation.

Specific Low-level Skill	LLM-generated Constraints	VQA Question
`go forward 10m`	+no obstacle in the front	+Is there any obstacle in the front?
`pick` block	+**{agent}**hold block	+Is the **{agent}** holding box?
`place` box on table	+box on table -**{agent}**hold box	+Is the box on table? -Is the **{agent}** holding box?
`open` fridge	+fridge is open	+Is fridge open?
`Give` milk to human	+human hold mild -**{agent}**hold mild	+Is the human holding milk? -Is the **{agent}** holding milk?

Table 8: Examples of LLM-generated constraints. The symbol "**+**" indicates the addition of the constraint while "**-**" means popping out this constraint. Questions are in the general structure: "Is the {constraint}?"

D.2 PROMPT

The robot performs manipulation tasks. At the same time, the robot needs to satisfy some constraints to ensure the successful execution of each task. Just fill in the blank and directly output the next step.

Task: Go forward

(0) Start, [Constraint: no obstacle in the front], (1) Go forward, [Constraint: no obstacle in the front], [Constraint violation: obstacle on the left], (2) Turn right, [Constraint:],(3) Go forward, [Constraint: no obstacle in the front], (4) Done.

Task: Collect meat and banana on the plate.

(0) Start, [Constraint:], (1) Go to meat, [Constraint:], (2) Pick up meat, [Constraint: The robot is holding meat], (3) Go to plate, [Constraint: The robot is holding meat], [Constraint violation: The robot is not holding meat] (4) Pick up meat, [Constraint: The robot is holding meat], (5) Go to plate, [Constraint: The robot is holding meat], (6) Place meat in plate, [Constraint:], (7) Go to banana, [Constraint:], (8) Pick up banana, [Constraint: The robot is holding banana], (9) Go to plate, [Constraint: The robot is holding banana], (10) Place banana in plate, [Constraint:], (10) Done.

Task: Stack blocks in the order of blue, red, and green.

(0) Start, [Constraint:], (1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the blue block, [Constraint: The red block is on the blue block], (3) Pick the green block, [Constraint: The robot is holding green block, the red block is on the blue block], (4) Place the green block on the red block, [Constraint: The red block is on the blue block, the green block is on the red block], [Constraint violation: the green block is not on the red block], (5) Pick the green block, [Constraint: The robot is holding green block, the red block is on the blue block], (6) Place the green block on the red block, [Constraint: The red block is on the blue block, the green block is on the red block], (7) Done.

D.3 OVERALL PIPELINE

To help understand, here we provide a specific case in the timeline, the abstract task is "Stack blocks in the order of brown, red, and green."

1. **Planning and constraint generation for step 1:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green. (1)
- LLM output: Pick the red block, [Constraint: The robot is holding red block]

2. **Step 1 execution**
3. **Planning and constraint generation for step 2:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2)
- LLM output: Place the red block on the brown block, [Constraint: The red block is on the brown block]

4. **Step 2 execution**
5. **Planning and constraint generation for step 3:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3)
- LLM output: Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block]

6. **Step 3 execution**
7. **Planning and constraint generation for step 4:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (4)
- LLM output: Place the green block on the red block, [Constraint: The red block is on the brown block, the green block is on the red block]

8. **Step 4 execution, Detector: the red block is not on the brown block**
9. **Planning and constraint generation for step 5:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (4) Place the green block on the red block, [Constraint: The red block is on the brown block, the green block is on the red block], [**Detector**: the red block is not on the brown block], (5)
- LLM output: Pick the red block, [Constraint: The robot is holding red block]

10. **Step 5 execution**
11. **Planning and constraint generation for step 6:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (4) Place the green block on the red block, [Constraint: The red block is on the brown block, the green block is on the red block], [Detector: the red block is not on the brown block], (5) Pick the red block, [Constraint: The robot is holding red block], (6)
- LLM output: Place the red block on the brown block, [Constraint: The red block is on the brown block]

12. **Step 6 execution**
13. **Planning and constraint generation for step 7:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (4) Place the green block on the red block, [Constraint: The red block is on the brown block, the green block is on the red block], [Detector: the red block is not on the brown block], (5) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (6) Place the red block on the brown block, [Constraint: The red block is on the brown block], (7)
- LLM output: Pick the green block, [Constraint: The robot is holding green block, The red block is on the brown block]

14. **Step 7 execution**
15. **Planning and constraint generation for step 8:**

- LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3) Pick the green block,

[Constraint: The robot is holding green block, the red block is on the brown block], (4) Place the green block on the red block, [Constraint: The red block is on the brown block, the green block is on the red block], [Detector: the red block is not on the brown block], (5) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (6) Place the red block on the brown block, [Constraint: The red block is on the brown block], (7) Pick the green block, [Constraint: The robot is holding green block, The red block is on the brown block], (8)
- LLM output: Place the green block on the red block, [Constraint: The red block is on the brown block, The green block is on the red block]

16. **Step 8 execution**
17. **Planning and constraint generation for step 9:**
    - LLM input: ...(prompt)...Task: Stack blocks in the order of brown, red, and green.(1) Pick the red block, [Constraint: The robot is holding red block], (2) Place the red block on the brown block, [Constraint: The red block is on the brown block], (3) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (4) Place the green block on the red block, [Constraint: The red block is on the brown block, the green block is on the red block], [Detector: the red block is not on the brown block], (5) Pick the green block, [Constraint: The robot is holding green block, the red block is on the brown block], (6) Place the red block on the brown block, [Constraint: The red block is on the brown block], (7) Pick the green block, [Constraint: The robot is holding green block, The red block is on the brown block], (8) Place the green block on the red block, [Constraint: The red block is on the brown block, The green block is on the red block]
    - LLM output: Done.
18. **Step 9 execution**