Distilling and Retrieving Generalizable Knowledge for Robot Manipulation via Language Corrections

Lihan Zha[‡], Yuchen Cui[‡], Li-Heng Lin[‡], Minae Kwon[‡], Montserrat Gonzalez Arenas[§], Andy Zeng[§], Fei Xia[§], Dorsa Sadigh[‡]

***Abstract*—Today's robot policies exhibit subpar performance when faced with the challenge of generalizing to novel environments. Human corrective feedback is a crucial form of guidance to enable such generalization. However, adapting to and learning from online human corrections is a non-trivial endeavor: not only do robots need to remember human feedback over time to retrieve the right information in new settings and reduce the intervention rate, but also they would need to be able to respond to feedback that can be arbitrary corrections about high-level human preferences to low-level adjustments to skill parameters. In this work, we present *Distillation and Retrieval of Online Corrections* (DROC), an LLM-based system that can respond to arbitrary forms of language feedback, distill generalizable knowledge from corrections, and retrieve relevant past experiences based on textual and visual similarity for improving performance in novel settings. DROC is able to respond to a sequence of online language corrections that address failures in both high-level task plans and low-level skill primitives. We demonstrate that DROC effectively distills the relevant information from the sequence of online corrections in a knowledge base and retrieves that knowledge in settings with new task or object instances. DROC outperforms baseline CaP [1] by using only half of the total number of corrections needed in the first round and requires little to no corrections after two iterations. We show further results and videos on our project website: https://sites.google.com/stanford.edu/droc.**

I. Introduction

From generating high-level plans to writing robot code – pre-trained large language models (LLMs) have shown to exhibit a wide range of capabilities on robots that can in-context adapt to feedback and adjust to language corrections. For example, InnerMonologue [2] and ReAct [3] show that LLMs can use language feedback to modify task-level step-by-step plans, while Language-to-Reward demonstrates that LLMs can respond to low-level feedback by changing the reward function [4]. In many cases, these feedback mechanisms have shown to be important for improving LLM-driven policies in their capacity to adapt to new settings [5–8].

While prior work can respond to language corrections, they often take a rigid approach that does not allow for *arbitrary forms* of human feedback. For instance, a human user might provide an instruction "Put the scissors in the top drawer" as shown in Fig. 1, which leads to a robot planning on "picking up the scissors". However, if the drawer is not already open, the correct plan requires the robot to first open the drawer before picking up the scissors. A human observing the robot might decide to provide corrective language that addresses this planning error. With such a correction, the robot can finally proceed with the correct high-level plan, and attempt to "open the top drawer". However, the primitive that the robot executes might miss the grasp point on the handle. A human observer again can provide language corrections to guide the robot to finally achieve the skill primitive proposed by the task planner. Considering this example, it is non-trivial to interpret these different types of feedback: to interpret the first correction ("You should open the drawer first"), one needs to know the robot's original plan; responding to the second correction ("Move a little bit to the right") requires the robot to infer what reference frame the human is using for directions; and for the next correction ("a bit more"), the robot needs to know the history of action it took. The ability to respond to these arbitrary corrections requires a method that is flexible enough to identify if the corrections are about the high-level task or low-level skill primitives, and is able to leverage the prior context when responding to a sequence of online corrections.

In addition, handling corrections is predominantly done in a *temporary* fashion in prior work – limited by what can fit within context length of the language model, and can be lost if not saved as the input context buffer overrides past feedback interactions with new ones. This can be frustrating if a robot needs to be corrected for every new task for the same underlying reason. As an example, in Fig. 1, the robot should be able to learn that it has only one arm and reuse this constraint when planning for future tasks. However, remembering the relevant information from a sequence of language corrections such as skill parameters can be challenging in more general settings beyond remembering simple constraints.

In this work, we address these challenges and enable LLM-based robot policies to respond to *arbitrary* forms of feedback and further *remember* and *retrieve* feedback for future tasks. We present DROC (Distillation and Retrieval of Online Corrections), a simple yet surprisingly effective formulation for responding to, remembering, and retrieving feedback. A key aspect of DROC is that it effectively uses an LLM to directly infer *how-to-respond*, *what-to-remember*, and *what-to-retrieve*. Specifically, DROC prompts an LLM to reason about relevant context required to respond to online corrections further identifying if the correction is about high-level task plans or low-level skill primitives. DROC distills language feedback into re-usable knowledge via LLMs to improve generalization of both high-level plans and low-level skill primitives to new settings while reducing the number of human corrections needed over time. Finally, DROC leverages visual similarities of objects for knowledge retrieval when language alone is not sufficient. Experiments across multiple long-horizon manipulation tasks show that DROC excels at (i) responding to online corrections, and (ii) adapting to new objects and

[‡] Computer Science Department, Stanford University, Stanford, CA, USA.
[§] Google Deepmind, Moutain View, CA.
Corresponding Emails: {lihanzha,yuchenc}@stanford.edu

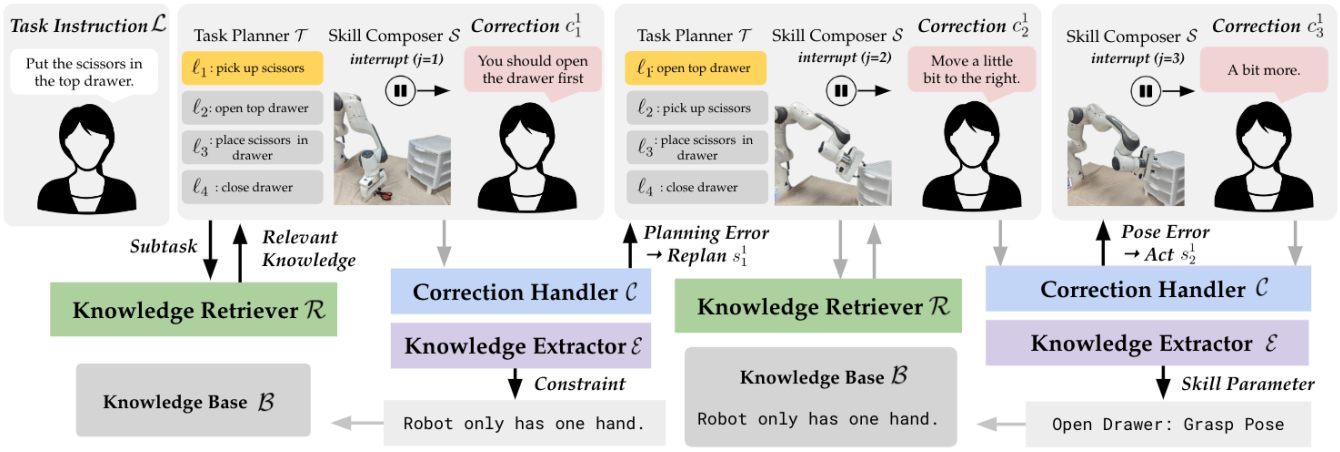

Figure 1: **Overview of DROC with example task "put the scissors in the top drawer"**: the human interrupted the robot when it attempts to pick up the scissors before opening the drawer, the correction handler regenerated a plan accordingly and the knowledge extractor extracts a high-level constraint; during executing the skill of *opening top drawer*, the human interrupted again to correct the grasping pose of the robot by providing two low-level commands.

configurations while consistently reducing the number of human corrections needed over time. DROC also achieves higher success rates and requires fewer corrections in new tasks compared to baselines such as Code as Policies (CaP) [1] and variants with simpler implementation of distillation or retrieval mechanisms.

II. Related Work

LLM-powered Robotics. Recent research has demonstrated planning capabilities of LLMs in robotics, including zero-shot generation of high-level task plans [9–12], adapting based on human feedback when uncertain about the task plan [2, 7], and directly producing robot code [1, 13]. In addition, LLMs are used in a variety of settings beyond task planning in robotics and human-robot interaction such as reward design [4, 14, 15], in-context learning [16], and reasoning about multimodal inputs such as gestures [17]. While most prior works focus on leveraging LLMs for task-level reasoning or generating high-level primitives in code, recent works have also studied effectively leveraging LLMs along with vision language models (VLMs) for responding to fine-grained language such as "move a bit to the right" by either leveraging reward functions [4], voxelized representations [18, 19], or keypoints [20]. On the other hand, a number of recent approaches have been leveraging VLMs directly as a success detector to ground the LLM plans and identify failures [21, 22]. However, existing VLMs are not trained on manipulation data, and hence are not able to detect or provide fine-grained feedback to fix low-level mistakes such as missing a grasping point by a few centimeters. While some prior works address correcting low-level failures and others address planning-level errors, none of the prior work can tackle corrections at both levels. Meanwhile, prior literature does not consider knowledge distillation and long-term generalization from corrections for LLM-based robot policies, and is only concerned about immediate responses to language corrections.

Knowledge Base for LLMs. Knowledge bases have previously been shown to be effective for retrieving examples when few-shot prompting LLMs [23–27]. Given a new example, in-context learning relies on the new example's similarity to previous examples to generate a response [28]. This makes retrieval-augmentation rather straightforward for traditional Q&A tasks – relevant examples can be sampled from the knowledge base by simply measuring the similarity between input questions (e.g., via sentence embeddings [29–31]). However to synthesize large amounts of feedback for robotics tasks, similarity-based methods are not enough; the method must also be able to summarize feedback [32, 33]. The design of DROC combines LLM capabilities previously demonstrated independently: (i) summarizing multiple rounds of feedback (a mechanism shared by [3, 33]), and (ii) to autonomously partition feedback into high-level or low-level to cover a broader range of re-usable adjustments [32] – in ways that together enable new modes of generalization from language feedback to new robot tasks or environments.

Learning from Language Corrections. Literature outside LLM-based frameworks has also explored how to incorporate language corrections for adapting policies. A body of work developed methods for training a policy conditioned on past rollouts and language corrections so that it can iteratively improve using human feedback [34–36] . Prior methods also propose to learn cost maps for modifying robot plans with natural language [6]. Both of these categories of work learn how to update a policy or a plan through post-hoc offline language corrections. Our method responds to human online corrections that modify the robot's behavior as it executes, so that a user can help the robot to recover from mistakes. However, there are a number of recent techniques that enable responding to real-time language corrections [37–39]. These works either make restrictive assumptions such as using distributed correspondence graphs to ground language, or they require extensive amount of language corrections and demonstrations. In contrast, our work does not require training data for interpreting corrections, and leverages LLMs to directly generate robot code given the language corrections.

III. DROC: Distillation and Retrieval of Online Corrections

In this section, we first present our problem formulation, then present our method, DROC, by discussing how to generate robot plans and skills in response to language corrections, and describing how to distill and retrieve generalizable knowledge.

Problem Formulation. Consider a manipulation problem defined by a natural language instruction $\mathcal{L}$ (e.g., "put the scissors in the drawer"). Directly generating control sequences in response to this instruction can be extremely hard because $\mathcal{L}$ may be complex and long-horizon. We leverage a task planner $\mathcal{T} : \mathcal{L} \mapsto (\ell_1, \ell_2, \ldots, \ell_M)$ to decompose the high-level instruction $\mathcal{L}$ into low-level skills ℓ_i (e.g., "open the drawer", "pick up the scissors"). For each skill, a skill composer maps the skill to the control policy $\mathcal{S} : \ell \mapsto p$. We follow Code-as-Policies (CaP) [1] and represent p as a code snippet. In practice, both $\mathcal{T}$ and $\mathcal{S}$ are not perfect and can make errors due to a variety of factors such as perception or control errors. A human interacting with this robot would only identify these errors *while* the robot is executing the skill. As shown in Fig. 1, the human only realizes the planner made a mistake when they see the robot attempt to pick up the scissors first. Once they spot the errors, the human can issue an arbitrary number of corrections during the execution of each skill p_i until ℓ_i is fulfilled or the plan $P = (\ell_1, \ell_2, \ldots, \ell_M)$ is correctly modified. We use the subscript j to stand for the round of corrections within the execution of l_i, and denote c_j^i as the human language correction and s_j^i as the solution to the correction. s_j^i takes the form of two types of language programs: 1) triggering $\mathcal{T}$ to generate the correct plan, or 2) triggering $\mathcal{S}$ to execute another language skill. At the end of correction round j, we define the interaction history as $H_j^i = \bigcup_{t=1:j}(P, \ell_i, p_i, c_t^i, s_t^i)$. We denote the total number of corrections at the end of the task to be J. The goal of a learning agent is to reduce the amortized number of corrections across tasks: $\bar{J} = \frac{1}{N}\sum_{k=1}^{N} J_k$, where J_k is the total number of corrections at the end of task $\mathcal{L}_k \in \{\mathcal{L}_{1:N}\}$.

The DROC framework. DROC can be decomposed into three reasoning modules: correction handler $\mathcal{C}$ (*how-to-respond*), knowledge extractor $\mathcal{E}$ (*what-to-remember*), and knowledge retriever $\mathcal{R}$ (*what-to-retrieve*). To generate the solution s_j^i to the human correction c_j^i, we first extract relevant knowledge from the history H_{j-1}^i with the correction handler $\mathcal{C}$ and decide whether it should generate a modified plan (triggering $\mathcal{T}$ with added constraint) or code policy (triggering $\mathcal{S}$ with relevant history) to address the user's correction, i.e., $s_j^i = \mathcal{C}(H_{j-1}^i)$. Upon the completion of a plan or a skill, $\mathcal{E}$ distills generalizable knowledge from the full interaction history H and saves it to the knowledge base $\mathcal{B}$. The saved knowledge can be retrieved later from $\mathcal{B}$ by the knowledge retriever $\mathcal{R}$ to guide task planning or skill composing. We first introduce how task planner $\mathcal{T}$ and skill composer $\mathcal{S}$ are implemented.

Task planning with $\mathcal{T}$. To ground the output plan $P = (\ell_1, \ell_2, \ldots, \ell_M)$, we provide scene information, few-shot examples, and rule constraints to guide the plan generation. Below is the template of the prompt[1] along with an example:

```
Your role is to break down instructions into smaller sub-tasks.
# Examples: ...
Constraints:
1. The robot should manipulate one object and only move its gripper once
in each sub-task.
2. If the instruction is ambiguous, first refer to the constraints to
see whether you can replace the ambiguous reference; if not just leave
it as is.
3. Tablewares should be put in the top drawer.
Object state: top drawer(closed), bottom drawer(open), spoon(on table),
salt(in bottom drawer)
Instruction: put the spoon into the drawer
Plan:     1: "Open the top drawer",
          2: "Pick up the spoon",
          3: "Put down the spoon into the top drawer",
          4: "Close the top drawer"
```

We define some initial constraints (1 and 2) in the prompt to enforce the hierarchical abstraction between $\mathcal{T}$ and $\mathcal{S}$, and handle human's ambiguous object reference. The initial object states are set to be "N/A", and we ask LLM to update the object states after each task execution or correction in the knowledge distillation phase (e.g., the object state becomes "top drawer(open)" after the execution of the task "open the drawer"). The constraints are also modifiable during that phase (e.g., constraint 3). In practice, one could use vision-language models (VLMs) as scene descriptors to obtain object states; however, we found existing open-source VLMs are sensitive to viewpoints and object locations, and require extensive prompt tuning.

Skill composing with $\mathcal{S}$. To ground the code policy p_i generated by $\mathcal{S}$, we provide function APIs and few-shot examples to an LLM that generates the code corresponding to each skill primitive, similar to the practice in prior work [1].

The skill composer can use perception APIs to call a VLM-based perceiver to detect the task-related object. For implementing the VLM perceiver, we use Segment-Anything [40] to obtain all objects' masks in the scene, and use CLIP [41] to label each mask. DROC also provides task-related knowledge to $\mathcal{S}$, which can be used as primitive parameters to guide code generation. Such task-related knowledge is obtained from the knowledge distillation and retrieval phase, which will be discussed later.

Correction handling with $\mathcal{C}$. Given a language correction c_j^i, DROC prompts the LLM to decide whether the correction is high-level (pertains to a constraint e.g., "robot can only grasp one object at a time", user preference) or low-level (primitive parameter e.g., relative gripper pose, or object information). If it's high-level, we pass the correction, the current skill and the original plan to the LLM for subsequent task planning. Otherwise if the correction is low-level, we ask the knowledge extractor to first extract relevant context from the (short-term) interaction history and then pass it as additional context to the LLM for subsequent code-writing.

[1]Full prompts are available on our website https://sites.google.com/stanford.edu/droc.

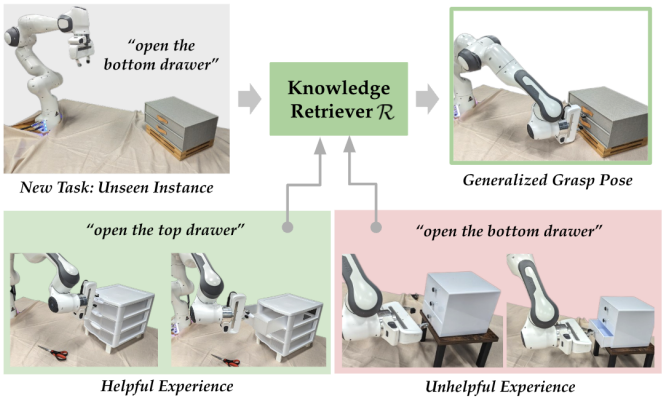

Figure 2: **Motivation for Visual Retrieval.** To retrieve the relevant task for opening the bottom gray drawer, textual similarity of the task instructions alone cannot filter the correct experience to reuse and similarity between visual features of the object (drawer handles specifically) are important for retrieving the correction past experience.

A naïve instantiation of the knowledge extractor $\mathcal{E}$ would be one that is prompted with the entire interaction history H and outputs the corresponding knowledge, which results in lengthy context that buries important information required for reasoning. At the same time we observe that the low-level correction c_j^i alone is semantically meaningful enough for deciding what history to retrieve for interpreting it. Thus, we only provide c_j^i to the LLM, and limit the retrievable history to four categories: (a) Last interaction. (b) Initial interaction. (c) No dependence. The prompt template we use is presented below:

```
A human is issuing corrections to a robot, which encounters errors during
executing a task. These corrections may depend on the robot's past
experiences. Your task is to determine what does a correction depend on:
(a) Last interaction. (b) Initial interaction. (c) No dependence.
# Examples: ...
"Move right a little bit": (c)
"Keep going": (a)
"Now you can continue": (b)
```

Once H_r^i is retrieved, $\mathcal{C}$ will prompt the LLM again to generate the solution s_j^i. We implement two types of correction handlers $\mathcal{C}_\mathcal{T}$ and $\mathcal{C}_\mathcal{S}$ depending on the error level, and s_j^i can be a re-plan or a new code policy. The prompt structures of $\mathcal{C}_\mathcal{T}$ and $\mathcal{C}_\mathcal{S}$ are similar to that of $\mathcal{T}$ and $\mathcal{S}$, with (H_r^i, c_j^i) and additional guidance concatenated at the end of the prompts. Once s_j^i is generated, we execute it on the robot, and repeat the above process until either the skill ℓ_i is fulfilled or the plan P has been successfully modified as deemed by the human.

To enable easy corrections and adaptations, we ground each correction to the related object or the task. For fine-grained corrections, we prompt $\mathcal{C}_\mathcal{S}$ to reason about whether the reference frame is an absolute frame or an object-centric frame. We empirically observe that human users tend to issue corrections in the object-centric frame. For example, when the robot is executing the task "open the drawer", the correction "move forward a bit" should be interpreted as "move towards the drawer a bit", which requires the knowledge of the drawer's frame to properly deal with the correction. To obtain the object-centric frame, we provide the LLM with the object's geometric properties (e.g., "normal vector") and ask it to represent the object's frame with these properties. We also ground the scale of movements to the size of the related object whenever the correction contains vague distance expressions (e.g., "a little bit"), which makes our system more adaptable.

Knowledge Distillation with $\mathcal{E}$. Given a history log with task instructions, generated LLM outputs, and several rounds of feedback interleaved, DROC prompts the LLM to summarize task-related knowledge, variables to save, modified code/plan, and updated states, then stores this relevant information into the knowledge base. At the end of each skill or plan, we provide the skill description ℓ_i and the interaction history H to $\mathcal{E}$ and use chain-of-thought prompting [42] to first reason about what are the types of knowledge that can be generalized to similar tasks in the future, and then extract the values of these types of knowledge from H. Below is the template of the prompt:

```
Your task is to extract reusable knowledge from the provided interaction
history.
# Examples: ...
Task name: {TASK_NAME}
Task-related knowledge: # LLM's answer here
Interaction history: {HISTORY}
Variables to save: # LLM's answer here
Modified code/plan: # LLM's answer here
Updated object state: # LLM's answer here
```

In our implementation, we separate the plan-level distillation and the skill-level distillation prompts. Examples of plan-level knowledge include task constraints (e.g., "the scissors cannot be put in a full drawer"), robot constraints (e.g., "the robot can only grasp one thing at a time"), and user preferences (e.g., "The user prefer milk to coke"). Examples of skill-level knowledge are task parameters (e.g., gripper pose, pull distance) and object information (e.g., visual feature, label). The distilled knowledge is saved to the knowledge base $\mathcal{B}$ in a dictionary format, with the key to be each task's name. We also ask $\mathcal{E}$ to update the objects' states after each skill is successfully performed. For ambiguous references, we remember the label of the object and also treat this information as a task constraint.

Knowledge Retrieving with $\mathcal{R}$. Given a new task, DROC prompts the LLM to decide which past experiences are relevant. In addition, our knowledge retriever leverages visual similarities for measuring relevancy when language alone is not sufficient. When queried by an instruction or a skill description, the knowledge retriever $\mathcal{R}$ indexes into the knowledge base $\mathcal{B}$ and retrieves relevant knowledge to help address the query. There are two retrieval metrics: (a) task semantics; (b) visual feature of the task-related object. Some knowledge is shared across different tasks (e.g., robot constraints, user preferences) and can always be retrieved, while other types of knowledge are specific to individual task category or object. The intuition here is that only knowledge that is both semantically and visually similar to the query can be retrieved. To implement this intuition, we first prompt $\mathcal{R}$ in a zero-shot manner to pick all the tasks that are semantically similar to the query:

```
I'll give you a list of tasks a robot has previously performed and a new
task to address. Your goal is to determine the following:
1. Does the new task fall into the same category with any previous task?
(E.g. "open" and "put" are different categories of tasks)
```

```
2. If so, which specific previous tasks are they? Answer in list format.
Previous tasks: 1. Open the top drawer. 2. Pick up the scissors. 3. Put
the mug on the shelf. 4. Pick up the yellow marker.
New task: Pick up the spoon.
Response:
    1: "Yes", 2: [2, 4]
```

Then, we can compare the queried object's visual feature to visual features of objects from the chosen tasks, and retrieve the one with highest visual similarity to add to the prompt of $\mathcal{T}$ or $\mathcal{S}$ as guidance. We motivate our design choice of using visual features for retrieval with the example shown in Fig. 2. In order to "open the bottom drawer" shown in the image, the robot needs to retrieve a grasp pose that associates with a horizontal drawer handle instead of a knob. It cannot retrieve the correct information using only skill or task similarity. Another advantage of visual retrieval is when the queried object's name contains visual information (e.g., "white drawer"), we can calculate the semantic-visual similarity between the name and the visual features of the chosen tasks to choose the most similar one. It is also important to note that different pre-trained vision or vision-language models encode different features. For manipulation tasks, we often care about the state and shape of an object more than the texture similarity. We use Dino-V2 [43] features out of this purpose for visual-visual retrieval, and use CLIP features for visual-semantic retrieval.

IV. Experiments

We evaluate our approach on a real-world tabletop environment with a Franka Emika Panda Robot. We use GPT-4 [44] for all LLM modules. We design experiments to test ***DROC***'s core capabilities: 1) accurately responding to online corrections, 2) distilling generalizable knowledge, and 3) retrieving relevant knowledge in novel tasks to improve performance.

Tasks. We will test ***DROC***'s ability to respond to both skill-level and plan-level corrections using separate set of tasks summarized in Table I. For skill-level, we experiment with 5 table-top manipulation tasks, as shown in top row of Fig. 3.

We iterate each task 3 times. We start with an initial setting and issue corrections until the first iteration is fulfilled. Then, with the distilled knowledge from last iteration, we change the setup (objects' location, instance etc.) and repeat the task.

To evaluate ***DROC*** on plan-level corrections, we design seven long-horizon tasks (see Fig. 4 for examples) to test four types of generalization of knowledge: (a) *User preferences*, which we aims to investigate whether ***DROC*** can distill user-specific knowledge and apply it to future tasks; (b) *Feasibility of plans*, which we want to see whether ***DROC*** can understand the constraints of different tasks and the robot from corrections; (c) *Common-sense reasoning*, which we aim to test if ***DROC*** can ground LLMs' powerful common-sense knowledge to robotics tasks by explicitly distilling task-related common-sense knowledge from corrections; (d) *Scene information*, which we aims to see if ***DROC*** can understand the scene by distilling scene information from corrections. Orange texts in Table I show the knowledge distilled from train tasks (i.e., tasks on the left) that can be retrieved for test tasks.

Table I: Summary of skill-level and plan-level tasks.

Skill-level Tasks	**Object Variations**	**Knowledge**
Open drawer	2 drawers	Grasp, Pull
Put scissors in drawer	2 scissors, 2 drawers	Grasp, Pull, Place
Put tape in drawer	5 tapes, 2 drawers	Grasp, Pull, Place
Hang cup on the rack	Flipped cup, upright cup	Grasp, Place
Pick up object	6 objects	Grasp

Plan-level Tasks	**Test Tasks**	**Knowledge**
Put scissors in drawer	Clean the table	Pref.
"User wants stationery in white drawer"		
Bring cup of coffee	Make cup of coffee	Pref. + Feasi.
"User doesn't drink black coffee"		
Heat milk in fridge	Slice carrot	Feasi.
"Robot only has one hand"		
Sort blocks to drawer	Sort blocks to drawer	Feasi.
"Same color block goes to same drawer"		
Put shoes on rack	Sort clothes into shelf	Comm.
"Same types of clothing go to same place"		
Set dinner table	I want to have lunch	Comm. + Scene. + Feasi.
"Fork on left, hot dish on heat mat"		
Place book on shelf	Put DVD on shelf	Scene.
"White shelf is full"		

Baselines. For skill-level tasks, we compare with the following baselines: (a) ***CaP***: Code as Policies [1], which handles corrections as new instructions and does not have a knowledge distillation or retrieval module; (b) ***Ours–H***: ***DROC*** with no initial history to show that knowledge distillation from prior tasks are important; (c) ***Ours–E***: ***DROC*** without relevant context extractor (uses all interaction history for correction handling), and (d) ***Ours–V***: ***DROC*** that does not leverage visual retrieval. The baselines share the same prompts with our task planner $\mathcal{T}$ and skill composer $\mathcal{S}$, and have access to exactly the same function APIs. For plan-level tasks, we compare with ***Ours–R***, an ablation that does not distill knowledge and naively retrieves saved plans.

Skill-Level Corrections. We report the amortized number of corrections for each task over learning iterations in Fig. 3. Three of the authors served as oracle users to provide feedback. The same user provided feedback within the same task such that the corrections are consistent across iterations. Overall, ***DROC*** outperforms all baselines by requiring less number of corrections, which strongly supports that ***DROC*** can synthesis generalizable knowledge from the corrections and use them to quickly adapt to unseen task settings. We further analyze the results for evaluating the core capabilities of ***DROC*** and our specific design choices:

DROC enables more effective corrections. The comparison between ***DROC*** (ours) and ***CaP*** suggests that ***DROC*** can respond to corrections significantly more effectively as it requires less than half corrections in the first round.

DROC distills generalizable knowledge within the same task (same language, same object). The comparison between the initial corrections needed for ***DROC*** (ours) and ***Ours–H*** demonstrates that the distilled knowledge helps with learning different variations of the same task.

Visual retrieval enables more accurate skill generalization. For the "Hang cup on Rack" task, we provide the system with 2 sets of knowledge with different visual features (cup's

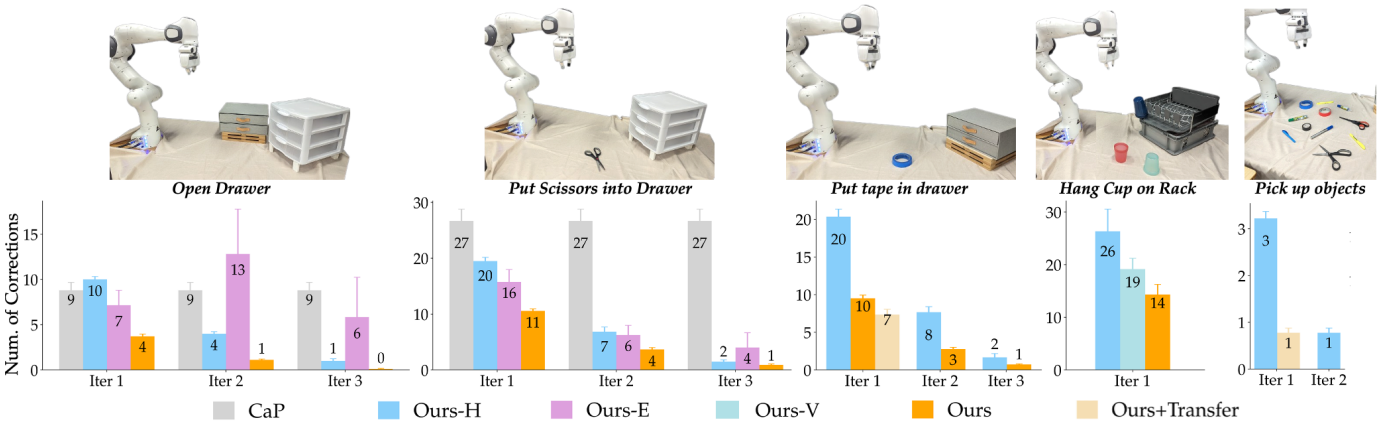

Figure 3: **Skill-level results.** For all tasks, the results are averaged over six rounds of experiments. The error bars reflect the standard errors across different rounds. Each iteration corresponds to a different task setting. The number of corrections declines as the iteration increases, which shows that ***DROC*** can generalize and adapt to unseen new settings. For the "Hang Cup on Rack" task, we are not showing decline of corrections over iterations but instead ablate the correction and distillation module of our system.

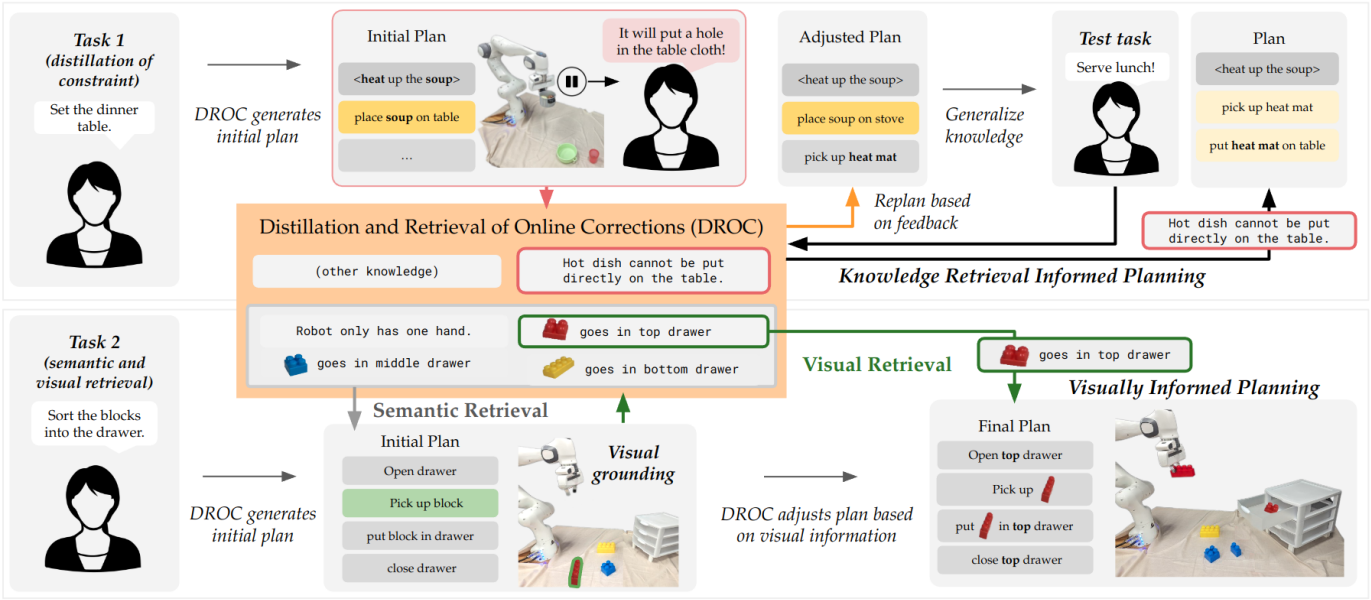

Figure 4: **Illustrative examples for plan-level test cases.** (1) upon interruption, ***DROC*** responds to correction by identifying it is a plan-level error and replanning, and distills the constraint for future tasks; (2) given a task with ambiguity, ***DROC*** retrieves past experiences base on semantic and visual similarities.

color, cup's orientation). Because upright cups and flipped cups require different policies (the robot needs to flip the upright cup before hanging it), our system needs to retrieve the correct knowledge through visually comparing cups' orientation at the presence of distraction from cups' colors. Through ablating with ***Ours–V***, we show that visual retrieval is an important element in ***DROC***.

DROC distills generalizable skill parameters across objects. We further tested cross-object transfer in "pick up object" and "put tape in drawer" tasks and report results as ***Ours+Transfer***. Specifically, for pick up object, we reuse history from other objects and for "put tape in drawer" , we reuse history from "put scissors in top white drawer". By reusing history from different tasks, our method further reduces the number of online corrections needed.

Plan-Level Corrections. We evaluate how ***DROC*** respond to plan-level corrections and generalize to new tasks with a suite of task scenarios we curated to cover a diverse range of potential mistakes. The train tasks and test tasks can be found in Table I. We repeat 3 trials for each task scenario that begins with the train task and then move to the test task with the knowledge distilled from the train task. We only report the average number of corrections needed for the test tasks in Table II because ***DROC*** and ***Ours–R*** share the same correction module and perform exactly the same on train tasks. The comparison between ***DROC*** and ***Ours–R*** again shows that ***DROC*** can distill and retrieve knowledge that empower generalization, leading to smaller number of corrections required on the test tasks. We don't ablate DROC's correction module here because without that it's too difficult for the task planner to understand

humans' corrections so as to modify the original plans. We visually illustrate two cases in Fig. 4: one demonstrating the effectiveness of knowledge distillation and retrieval of high-level constraint and the other showcasing how semantic retrieval and visual retrieval aid high-level task planning.

Table II: Number of corrections required for plan-level test tasks.

Task Types	Ours	Ours–R
User Preferences	**0.5**	1
Feasibility of Plans	**0.67**	1.33
Common-sense Reasoning	**0**	1.5
Scene Information	**0.5**	1.5

V. Conclusion

We propose DROC, an LLM-based system for enabling robots to respond to arbitrary forms of online language corrections, distill generalizable information, and retrieve relevant feedback for novel tasks. We demonstrate that DROC outperforms baselines for responding to both low-level and high-level corrections, and can effectively generalize knowledge within the same tasks and across similar tasks. An important future direction of DROC is to design an efficient knowledge base that scales to large number of interactions. DROC is an exciting step towards enabling robots to reliably learn from corrections, paving a path for richer modes of human-robot interaction.

References

[1] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *arXiv preprint arXiv:2209.07753*, 2022.

[2] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," in *arXiv preprint arXiv:2207.05608*, 2022.

[3] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[4] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[5] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots that use language," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 25–55, 2020.

[6] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox, "Correcting robot plans with natural language feedback," *arXiv preprint arXiv:2204.05186*, 2022.

[7] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley *et al.*, "Robots that ask for help: Uncertainty alignment for large language model planners," *arXiv preprint arXiv:2307.01928*, 2023.

[8] M. Kwon, H. Hu, V. Myers, S. Karamcheti, A. Dragan, and D. Sadigh, "Toward grounded social reasoning," *arXiv preprint arXiv:2306.08651*, 2023.

[9] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[10] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser, "Tidybot: Personalized robot assistance with large language models," *arXiv preprint arXiv:2305.05658*, 2023.

[11] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can and not as i say: Grounding language in robotic affordances," in *arXiv preprint arXiv:2204.01691*, 2022.

[12] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *arXiv preprint arXiv:2303.12153*, 2023.

[13] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," *arXiv preprint arXiv:2209.11302*, 2022.

[14] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," *arXiv preprint arXiv:2303.00001*, 2023.

[15] H. Hu and D. Sadigh, "Language instructed reinforcement learning for human-ai coordination," in *40th International Conference on Machine Learning (ICML)*, 2023.

[16] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng, "Large language models as general pattern machines," *arXiv preprint arXiv:2307.04721*, 2023.

[17] L.-H. Lin, Y. Cui, Y. Hao, F. Xia, and D. Sadigh, "Gesture-informed robot assistance via foundation models," in *7th Annual Conference on Robot Learning*, 2023.

[18] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023.

[19] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[20] P. Sundaresan, S. Belkhale, D. Sadigh, and J. Bohg, "Kite: Keypoint-conditioned policies for semantic manipulation," *arXiv:2306.16605*, 2023.

[21] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," *arXiv preprint arXiv:2306.15724*, 2023.

[22] Y. Guo, Y.-J. Wang, L. Zha, Z. Jiang, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," *arXiv preprint arXiv:2307.00329*, 2023.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[24] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[25] G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave, "Few-shot learning with retrieval augmented language models," *arXiv preprint arXiv:2208.03299*, 2022.

[26] Z. Wang, M. Li, R. Xu, L. Zhou, J. Lei, X. Lin, S. Wang, Z. Yang, C. Zhu, D. Hoiem *et al.*, "Language models with image descriptors are strong few-shot video-language learners," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8483–8497, 2022.

[27] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong,

S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani *et al.*, "Socratic models: Composing zero-shot multimodal reasoning with language," *arXiv preprint arXiv:2204.00598*, 2022.

[28] S. C. Chan, I. Dasgupta, J. Kim, D. Kumaran, A. K. Lampinen, and F. Hill, "Transformers generalize differently from information stored in context vs in weights," *arXiv preprint arXiv:2210.05675*, 2022.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[30] H. Choi, J. Kim, S. Joe, and Y. Gwon, "Evaluation of bert and albert sentence embedding performance on downstream nlp tasks," in *2020 25th International conference on pattern recognition (ICPR)*. IEEE, 2021, pp. 5482–5487.

[31] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[32] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv: Arxiv-2305.16291*, 2023.

[33] A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang, "Expel: Llm agents are experiential learners," *arXiv preprint arXiv:2308.10144*, 2023.

[34] J. D. Co-Reyes, A. Gupta, S. Sanjeev, N. Altieri, J. Andreas, J. DeNero, P. Abbeel, and S. Levine, "Guiding policies with language via meta-learning," in *International Conference on Learning Representations*, 2018.

[35] A. F. C. Bucker, L. F. C. Figueredo, S. Haddadin, A. Kapoor, S. Ma, and R. Bonatti, "Reshaping robot trajectories using natural language commands: A study of multi-modal data alignment using transformers," *ArXiv*, vol. abs/2203.13411, 2022.

[36] A. F. C. Bucker, L. F. C. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti, "Latte: Language trajectory transformer," *ArXiv*, vol. abs/2208.02918, 2022.

[37] A. Broad, J. Arkin, N. Ratliff, T. Howard, and B. Argall, "Towards real-time natural language corrections for assistive robots," in *RSS Workshop on Model Learning for Human-Robot Communication*, 2016.

[38] Y. Cui, S. Karamcheti, R. Palleti, N. Shivakumar, P. Liang, and D. Sadigh, "No, to the right: Online language corrections for robotic manipulation via shared autonomy," in *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 2023, pp. 93–101.

[39] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, "Interactive language: Talking to robots in real time," *IEEE Robotics and Automation Letters*, 2023.

[40] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023.

[41] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[42] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023.

[43] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.

[44] OpenAI, "Gpt-4 technical report," 2023.