From Local Corrections to Generalized Skills: Improving Neuro-Symbolic Policies with MEMO

Benjamin A. Christie[1], Yinlong Dai[1], Mohammad Bararjanianbahnamiri[1], Simon Stepputtis[2], and Dylan P. Losey[1]

Abstract*— Recent works use a neuro-symbolic framework for general manipulation policies. The advantage of this framework is that — by applying off-the-shelf vision and language models — the robot can break complex tasks down into semantic subtasks. However, the fundamental bottleneck is that the robot needs skills to ground these subtasks into embodied motions. Skills can take many forms (e.g., trajectory snippets, motion primitives, coded functions), but regardless of their form *skills act as a constraint*. The high-level policy can only ground its language reasoning through the available skills; if the robot cannot generate the right skill for the current task, its policy will fail. We propose to address this limitation — and dynamically expand the robot's skills — by leveraging user feedback. When a robot fails, humans can intuitively explain what went wrong (e.g., "no, go higher"). While a simple approach is to recall this exact text the next time the robot faces a similar situation, we hypothesize that by collecting, clustering, and re-phrasing natural language corrections across multiple users and tasks, we can synthesize more general text guidance and coded skill templates. Applying this hypothesis we develop Memory Enhanced Manipulation (MEMO). MEMO builds and maintains a retrieval-augmented *skillbook* gathered from human feedback and task successes. At run time, MEMO retrieves relevant text and code from this skillbook, enabling the robot's policy to generate new skills while reasoning over multi-task human feedback. Our experiments demonstrate that using MEMO to aggregate local feedback into general skill templates enables generalization to novel tasks where existing baselines fall short. See supplemental material here: `https://collab.me.vt.edu/memo`

I. Introduction

Neuro-symbolic approaches are a promising way to obtain robot policies by combining the benefits of neural reasoning and control. Imagine that a human instructs the robot in Figure 1 to "toast food." Under neuro-symbolic paradigms, the robot first applies a general-purpose foundation model to determine the necessary steps (e.g., "open the door"). The robot then tries to ground this natural language instruction into numerical control parameters (e.g., a sequence of positions and velocities). Vision and language models excel at the first part — identifying the right high-level subtask — but language is ambiguous, and it is difficult to convert language into precise robot motions.

To address this problem, existing works equip the robot with *skills*. Skills are motion primitives or simple policies that are associated with semantic characteristics and interpretable parameters [1]–[3]. For instance, a "grasp" skill could control the robot arm to move to a target location and then close its gripper. Skills are an effective way to ground the robot's policy because (a) the semantics associated with every skill make it easy for the vision and language model to determine which skills to apply, and (b) the model can intuitively specify the small number of interpretable parameters that each skill requires. Although recent research has developed multiple ways in which robots can collect or generate skill libraries [4]–[7], these *skills are still inherent constraints*. If the robot does not know (or cannot generate) the right skills for a given task, the policy is unable to complete that task.

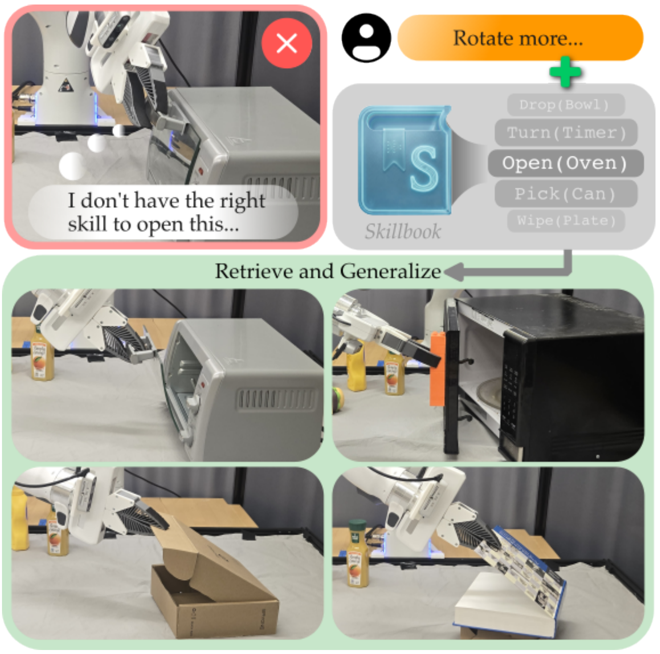


Fig. 1. Neuro-symbolic policy tries to "toast food." (Top Left) The robot fails because it lacks the necessary skill for opening the toaster. (Top Right) A human gives feedback. MEMO collects this and other feedback into a retrieval-augmented skillbook, and then clusters the skillbook to extract generalized text and code templates. (Bottom) The generalized entries guide the policy's code generation for new skills, e.g., an `open_door` skill.

To address the central challenge of skill libraries we introduce **MEMO**, a method that enables robots to dynamically expand the ways they ground high-level semantics into low-level actions by building new generalized skills from feedback. Consider Figure 1. When the robot makes a mistake, the human gives a natural language correction (e.g., "no, you need to rotate more"). Reasoning over this specific feedback helps the robot adapt its existing motions to their current context [8]–[10]. But for long-term improvements,

This work is supported in part by NSF Grant #2337884.

[1]Collab, Dept. of Mechanical Engineering, Virginia Tech, Blacksburg, USA.
[2]TEA Lab, Dept. of Mechanical Engineering, Virginia Tech, Blacksburg, USA. Email: `{benc00, daiyinlong}@vt.edu`

our hypothesis is that:

By aggregating natural language corrections from multiple users across multiple tasks into a skillbook*, we can create and refine a continuously evolving set of generalized skills, leading to improvements beyond the corrected tasks.*

A *skillbook* is a specific instance of retrieval-augmented generation (RAG) that we design for neuro-symbolic robot policies. The skillbook begins as a set of text entries, where each entry includes the intended subtask, state information, and the human's correction for that subtask. At run time the vision and language model retrieves relevant entries from the skillbook and then uses those entries to augment its reasoning for the current conversation. Returning to our example: when pulling open the toaster door, the robot remembers that it should "rotate more."

But the fundamental advantage of MEMO is not targeted recall; it is how we evolve the skillbook over time. As the skillbook accumulates user feedback and task successes, MEMO utilizes a language model in an asynchronous background process to cluster related text entries. By reasoning over these clusters, MEMO refines the skillbook into parameterized code templates and generalized human corrections, which serve as a reference when the policy generates code for new skills. See Figure 1: by clustering several entries on how to open doors, the robot generates and refines a more invariant `open_door()` function parameterized by the handle pose and door dimensions. Now — instead of retrieving specific natural language corrections — the robot retrieves relevant code templates that go beyond the limits of both its existing skills and the code the model could have generated without any human guidance. By aggregating feedback into generalized functions, MEMO thus provides a meaningfully expanding set of skills that neuro-symbolic policies can apply to convert language into actions.

Overall, this work is a step towards general-purpose robots that leverage human feedback to improve long-term capabilities. We make the following contributions:

Collecting and Retrieving Feedback. We introduce a skillbook, a database containing human feedback and robot code. MEMO automatically paraphrases human corrections into task-specific and task-invariant entries, and stores these entries alongside any code templates the robot used to successfully complete its subtasks. Skillbook entries can be retrieved when the robot faces similar contexts.

Clustering Feedback around Skill Templates. We cluster skillbook entries while conditioning on code templates. This process removes repetitive or contradictory feedback while also summarizing multiple human corrections across different contexts to reach more general guidance. At run time, the robot's policy retrieves and reasons over this generalized feedback in order to generate code for new skills.

Improving Beyond Local Feedback. We collect a skillbook from user feedback across simulated tasks, and then compare MEMO to state-of-the-art baselines in simulation and the real world. Across previously unseen tasks MEMO achieves a higher zero-shot success rate than either robotics foundation models or neuro-symbolic approaches that only reason about relevant human feedback.

II. Related Work

MEMO combines the common-sense reasoning capabilities of large foundation models with a continuously expanding library of skills. We create these skills by aggregating human feedback into a skillbook and then leveraging a retrieval-augmented generation approach.

A. Foundation Models for Robot Learning

Robotics vision-language-action (VLA) models such as RT-2 [11], Octo [12], and $\pi_{0.5}$ [1] have shown remarkable success in robot control by directly learning mappings from observations and language instructions to robot actions. Pretrained on large-scale internet data, these models generalize across tasks and embodiments, but operate as monolithic policies, making it difficult to adapt to new behaviors without requiring large amounts of additional data collection and fine-tuning. A common challenge in these models is to ground the language and vision components into the dynamics of the physical world. To address this issue, reasoning can be separated from the control needed to actuate the real robot by leveraging skill libraries as an intermediate layer [5], [8]. With this paradigm, the foundation model selects and parameterizes a skill from a set of predefined options. For example, SayCan [7] grounds language model plans in a robot's affordances by scoring available skills, while ProgPrompt [6] generates executable programs that compose skill primitives. Although effective for a known domain, the reliance on a finite set of skills limits generalization. Code as Policies [4] generates open-ended code at test time — not relying on a set of pre-defined primitives — while LMP [2] argues that sufficiently general motion controllers, such as Dynamic Motion Primitives, enable general-purpose task execution when parameterized by VLMs. However, even with zero-shot code and controller generation, the resulting behaviors may still fail to complete the desired task. Our skillbook addresses this gap by anchoring code generation in physical, user-provided corrections, while providing a continuously growing set of capabilities that extend beyond both the robot's original repertoire and what a foundation model can produce without grounded feedback.

B. Skill Learning from Feedback and Experience

Recent advances in embodied AI have explored how foundation models can be leveraged to construct, expand, and reuse skill libraries for long-horizon control. Agentic systems such as Voyager [5] iteratively generate executable programs, store successful behaviors in an ever-growing code-based skill library, and reuse them to solve novel tasks. Building on this paradigm [13] and [14] introduce mechanisms for composable skills via LLM-guided abstractions. Similarly, low-level behaviors can be gathered with reinforcement learning [15] or derived from demonstrations [16], allowing high-level task and motion planners to sequence these new skills for zero- or few-shot transfer.

Approaches such as DROC [8] distill language corrections from human users into a text-based knowledge base of rules, constraints, and preferences that are retrieved for future tasks, while other methods use feedback to shape reward functions [17]. However, because these approaches utilize local feedback merely to adjust existing skill parameters or task plans, they remain limited to the original set of skills. In contrast, MEMO aggregates corrections across multiple users and different tasks in order to synthesize generalizable, parameterized code functions that can cover novel skills.

C. *Retrieval-Augmented Generation for Robotics*

Retrieval-augmented generation (RAG) has emerged as an efficient paradigm for grounding language model reasoning in external knowledge that would not otherwise fit into the context windows of the underlying LLMs. While largely applied in language models, recent work has adapted RAG to robotics for grounding an agent's decision-making in past experiences [18]. PragmaBot [19] employs a VLM to self-reflect on action outcomes, storing summarized experiences in a long-term memory that can be retrieved for future tasks. DROC [8] similarly retrieves past corrections based on textual and visual task similarity to inform skill creation in novel settings. These methods retrieve natural language experiences, corrections, or summaries to augment and contextualize the execution of desired tasks. ELLMER [20], on the other hand, retrieves code examples from a curated knowledge base, demonstrating the effectiveness of retrieving executable code rather than just natural language. However, unlike methods that rely on purely text-based memories or pre-curated static codebases, MEMO actively evolves its knowledge base. By clustering raw human feedback, our skillbook dynamically synthesizes executable, parameterized code templates that can be retrieved to guide skill generation.

III. Problem Statement

We explore settings where a robot arm is performing tabletop manipulation tasks. These include simple pick-and-place tasks (e.g., "put the banana on the plate") as well as more complex interactive tasks (e.g., "toast food"). A human gives the robot their natural language description of the task τ, and the robot arm must convert this description into successful task execution.

Scene Graph. Let $o \in \mathcal{O}$ be the robot's observation of its environment state. In our setting o contains the robot's end-effector pose and an RGB-D image of the tabletop. From this observation, the robot autonomously extracts a 3D scene graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, o)$ which includes both the raw observation and its processed nodes and edges. Each node $v \in \mathcal{V}$ contains an object's semantic label (e.g., "door handle") and that object's pose within the robot's coordinate frame. Edges $e \in \mathcal{E}$ describe any natural language relationships between objects (e.g., "the toaster door is closed"). In our real-world experiments the robot builds this scene graph $\mathcal{G}$ using Gemini-Robotics ER [21] and LangSAM [22], [23].

Policy. The robot's policy converts the user's task description τ and the scene graph $\mathcal{G}$ into action $a \in \mathcal{A}$:

$$a \sim \pi(\cdot \mid \tau, \mathcal{G}, \rho) \tag{1}$$

Here ρ is a prior that guides the policy. If the policy is a foundational robotics model this prior could be a set of pretrained weights [1], [24], [25]. Alternatively, for neuro-symbolic policies the prior is a general-purpose prompt that explains how π should function [3], [26]. In our approach we instantiate the policy as a vision-language model, and ρ is a task-invariant system prompt that initializes the policy.

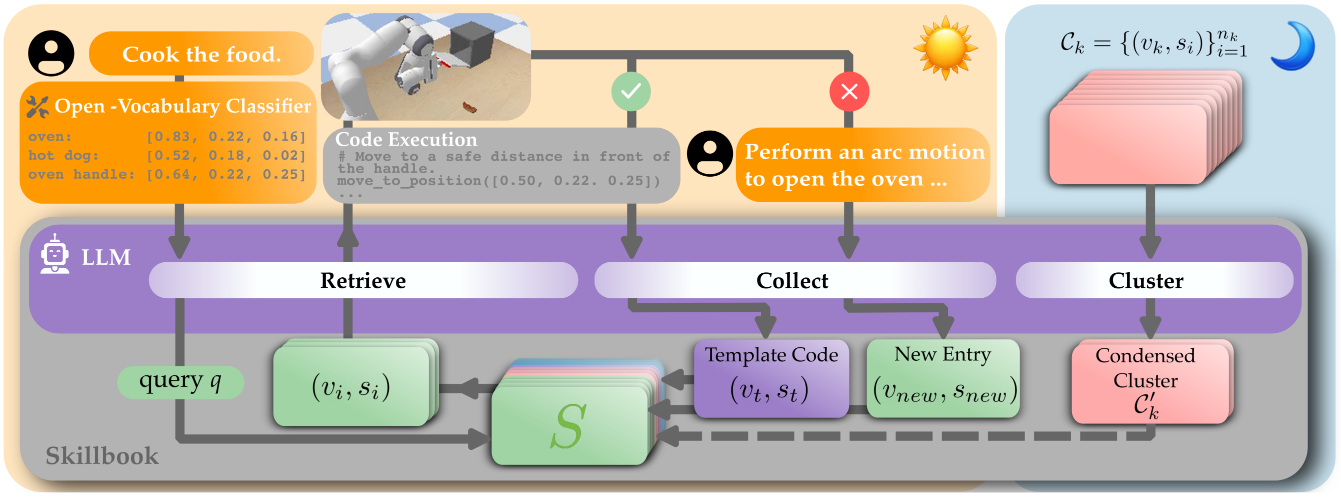
Skills. The actions output by the policy are not necessarily position or torque commands. Instead of these low-level behaviors, related works output trajectory snippets [1], motion primitives [2], or function calls [3]. In this paper we treat *skills* as actions. We define a skill as a parameterized function that intuitively grounds language reasoning into low-level motion control. For example, `grasp(pose)` could be a skill that moves the robot's end-effector to `pose` and then closes its grippers. Under this framework each action $a = (k, \theta)$ separates into a skill k and the parameters θ. The set of all skills $k \in \mathcal{K}$ is the skill library, and the action space $\mathcal{A} = \mathcal{K} \times \Theta$ incorporates this skill library and its parameters $\theta \in \Theta$. We emphasize that the robot's policy in Equation (1) is constrained to the library: if the policy does not have access to a skill or sequence of skills that are necessary to complete the current task, the robot will inevitably fail.

Feedback. Ideally the robot's policy π is able to complete the user's task τ on its first attempt by choosing the correct skills and parameters for those skills. But in practice the robot will sometimes fall short. When these failures occur, the user can provide text descriptions that explain what went wrong (e.g., "no, you need to grasp the handle from the side"). The robot should leverage this feedback to improve its future behavior. Naively, we can just incorporate the human's text into the prior ρ, thereby helping the robot address a specific error. But our goal is for the robot to make long-term improvements to π. Instead of simply recalling the human's text for one task, we want the robot to build a skill that can be leveraged across multiple tasks — e.g., an `open_door()` function. Based on the human's short-term and localized corrections, our goal is to extract generalized feedback that the robot can leverage to dynamically expand $\mathcal{A}$ and increase its capabilities.

IV. MEMO: Memory Enhanced Manipulation

To convert local feedback into general skills we propose **MEMO**. Let $\mathcal{S}$ be a knowledge base of natural language corrections, general feedback, and skill templates. We will refer to $\mathcal{S}$ as a *skillbook*. In its most basic form, MEMO is a method to build and maintain the skillbook $\mathcal{S}$ so that the robot's neuro-symbolic policy can expand its action space across multiple tasks. We re-define Equation (1) to condition π on both the fixed prior ρ and the adaptive skillbook $\mathcal{S}$:

$$a \sim \pi(\cdot \mid \tau, \mathcal{G}, \rho, \mathcal{S}) \tag{2}$$


Cook the food.
Open -Vocabulary Classifier
Code Execution
Perform an arc motion to open the oven ...
$\mathcal{C}_k = \{(v_k, s_i)\}_{i=1}^{n_k}$
LLM
Retrieve
Collect
Cluster
query q
(v_i, s_i)
$\mathcal{S}$
Template Code (v_t, s_t)
New Entry (v_{new}, s_{new})
Condensed Cluster $\mathcal{C}'_k$
Skillbook

```
oven:         [0.83, 0.22, 0.16]
hot dog:      [0.52, 0.18, 0.02]
oven handle:  [0.64, 0.22, 0.25]
```

```
# Move to a safe distance in front of
the handle.
move_to_position([0.50, 0.22. 0.25])
...
```

Fig. 2. MEMO builds generalizable skills by retrieving, collecting, and clustering human feedback. MEMO first decomposes a task into its atomic actions (i.e., subtasks). Before selecting an action, it queries the skillbook for any subtask-relevant feedback or functions that it can reference. If the user intervenes, MEMO stores a paraphrased form of their feedback in the skillbook for future reference. Otherwise, we store the action as a skill template. Offline MEMO clusters and compresses the entries in the skillbook to mitigate information loss and gather more general skills for future action generation.

At run time, the robot's policy retrieves relevant language feedback or function templates from $\mathcal{S}$, and then reasons across this information when generating code for parameterized skills a. In Section IV-A we explain the contents of $\mathcal{S}$ and how the robot uses human feedback and task success to fill its skillbook. Next, in Section IV-B we describe how the policy π in Equation (2) retrieves entries from $\mathcal{S}$ that are relevant for the current task. Finally, in Section IV-C we outline how the skillbook is clustered to extract more generalized entries from multiple pieces of local feedback. Together, this process of collecting, retrieving, and clustering corrections into a skillbook forms our MEMO framework. We visualize this MEMO approach in Figure 2.

A. Collecting Feedback into Skillbook Entries

We instantiate the knowledge gathered from human feedback as a vector database. Here $\mathcal{S} = \{(v, s)\}$, where each v is an embedding vector and s is the associated skill information. Intuitively, we can think of v as the key used to identify the entry, and s as the information stored within that entry.

Parsing Human Feedback. During task execution a human can stop the robot at any time and provide natural language feedback about its mistakes. Of course, different users express the same ideas with different words, phrases, and syntax. The robot therefore paraphrases the human's feedback with its language model to remove any information that is too task-specific, redundant, or irrelevant. For instance: if the human says "the robot should place its end-effector at position $(0.5, -0.3, 0.2)$ to grab the handle," the paraphrased text could be "move to the door handle."

Each piece of human feedback becomes a new entry in $\mathcal{S}$. Specifically, the paraphrased text is the information s, and the embedding vector v is set based on the robot's context. For our tabletop manipulation tasks we divide this context into two parts: the natural language *action* that the robot is trying to perform, and the *object(s)* that the robot is interacting with. If the human provides feedback when the robot is, for example, trying to open the toaster, the action token could be "open" and the object token could include "toaster" and/or "door." Together these action and object tokens form the embedding vector $v = (v_{act}, v_{obj})$. Optionally, designers can append the current scene graph $\mathcal{G}$ to v to aid in state-conditioned retrieval.

When paraphrasing the human's task-specific feedback we also prompt the language model to try and extract higher-level corrections. For example, if the user interrupts with statements like "you hit the table while you were moving," then the language model may identify more high-level guidance such as "ensure the robot stays a safe height above the table." The advantage of high-level feedback is that it can be applied across multiple contexts, and is not necessarily tied to the current action or objects. We capture this task invariance within the skillbook $\mathcal{S}$ by indexing general statements s with a shared global key v_g.

Leveraging Task Success as Implicit Feedback. User feedback tells the robot about its *incorrect* motions. But we can also reinforce *correct* behaviors by storing successful code within the skillbook $\mathcal{S}$. When the robot successfully completes a subtask, the robot converts the code it wrote for that subtask into a function template. Here *code* refers to the skills $k \in \mathcal{K}$ the robot invoked, the order of those skills, and any chosen skill parameters $\theta \in \Theta$. The robot then takes the code for $(a_1, a_2, \ldots)$ and queries its language model for a more general form; i.e., removing any hardcoded values. The inputs to this function template can be the environment scene, the current robot configuration, and the positions of task-relevant objects. The template is then stored as a new entry (v_f, s_f). The language-generated code template is s_f, and the vector for retrieving that code is v_f. As before, v_f includes the action the robot used that code to complete, as well as any objects the code template interacts with.

B. Retrieving Relevant Skillbook Entries

In Section IV-A we explained how our skillbook is populated from user feedback and task success. The size of this skillbook $\mathcal{S}$ grows over time as the robot attempts new manipulation tasks; as such, we cannot expect the robot to reason across its entire database when choosing actions in Equation (2). Recent studies have shown that language models experience performance degradation as their input length increases [27], [28]. We therefore apply a retrieval-augmented generation (RAG) approach to collect the most relevant entries in $\mathcal{S}$ based on the robot's task τ and context $\mathcal{G}$. These relevant entries are then entered into the current conversation. Importantly, *the robot does not simply copy any skill templates retrieved from* $\mathcal{S}$ — instead, we use these templates as a reference, and the robot constructs new code to generate embodied motions based on the system prior ρ and retrieved entries in skillbook $\mathcal{S}$.

Retrieving. Within our neuro-symbolic policy the robot first uses a vision and language model to reason across the task τ. The policy next determines its semantic sequence of subtasks: e.g., when asked to "toast food" the first subtask might be to "open the toaster." At this point the policy in Equation (2) performs retrieval to search for relevant entries in its knowledge base $\mathcal{S}$. More formally, the policy compares the action and object(s) it plans to interact with against the embeddings v in the skillbook:

$$r(q, v) = \frac{1}{\lambda_1 + \lambda_2}\Big(\lambda_1 \cos\left(q_{act}, v_{act}\right) + \lambda_2 \cos\left(q_{obj}, v_{obj}\right)\Big)$$

Here $q = \{q_{act}, q_{obj}\}$ is the robot's query (i.e., the actions and objects in its intended subtask) and $\cos$ is the cosine similarity. We weight this similarity by hyperparameters $\lambda_1, \lambda_2 > 0$. Increasing the value of λ_1 means that the robot is more likely to retrieve entries with synonymous actions: e.g., if the subtask calls for "grasping," the system might retrieve entries in $\mathcal{S}$ that "grab," "pick up," or "hold." The retrieved information is then added to the context window for policy π. This matches our framework in Equation (2): the robot selects actions while reasoning across its static system prompt ρ and the relevant skillbook entries $(v, s) \in \mathcal{S}$. The robot then generates code to complete its current subtask; this code is based on the available templates in $\mathcal{S}$ and the pre-programmed skill library in ρ.

C. Clustering Entries into Generalized Skills

From Section IV-B the robot now has a way to retrieve relevant information and apply it to the current task. But the point of our skillbook is not just to leverage *localized* feedback for specific situations (e.g., using text about how to open the toaster the next time we encounter the same toaster). Rather, we want to extract *generalized* code and language that the robot can use to generate skills for a variety of tasks. In addition, we need the skillbook to scale with an increasing number of users and tasks. For instance, if the skillbook contains 50 entries about how it should open a door, these entries quickly become repetitive, inefficient, and even contradictory. We therefore *cluster* the skillbook entries offline to refine $\mathcal{S}$ and aggregate generalized knowledge.

TABLE I
EXPERIMENTAL TASKS, TASK TYPES, AND THE TOTAL AMOUNT OF HUMAN FEEDBACK COLLECTED FOR MEMO AND DROC−V.

Task Name	Type	# Feedback
Test Tasks (used to collect user feedback)		
Pick the left can	SR	2
Put the banana on the plate	SR	2
Put the cube away	LH	11
Put the food in the microwave	TR	14
Put the food in the pan and place it in the microwave	SR	22
Put the trash away	SR	29
Stack the cubes	LH	3
Turn on the faucet	TR	2
Set the table	LH	11
Clean up the table	LH	23
Heat the food	SR	16
Cook the food	SR	2
Empty the fridge	LH	23
Make toast	SR	7
Move the lonely object to the others	SR	5
Move the right can to the left can	SR	2
Open the fridge	TR	2
Open the bottle	TR	4
Wipe the plate	CR	7
Season the food	SR	37
Held-Out Tasks (used for evaluation)		
Place the apple on the table	LH	–
Pour the can	TR	–
Close the bottle	TR	–
Empty the cabinet	SR	–
Put the food in the oven	CR	–
Total Feedback		**224**

Clustering. Our first step in this offline phase is to group the skillbook entries by their embeddings. Specifically, we form K different clusters $\mathcal{C}_k = \{(v_k, s_i)\}_{i=1}^{n_k}$ by grouping all feedback entries s_i that share the same embedding vector v_k. Our goal is to condense this cluster into more compact and generalized guidance. Returning to our example — instead of 50 descriptions of how to open 50 different doors, we would rather have a small number of entries that cover all of these instances. MEMO accomplishes this compression by passing each cluster $\mathcal{C}_k$ through a language model to obtain a condensed feedback description $\mathcal{C}'_k$. To address contradictory feedback and mitigate potential information loss, we condition the process on any related function templates (v_f, s_f). This design biases the clustering so that it prunes any feedback which does not agree with the function template — and since the function template is built from successful code, we know that the human's guidance should be consistent with s_f. The language model therefore inputs $\mathcal{C}_k$ and (v_f, s_f), and outputs a new set of skillbook entries $\mathcal{C}'_k$ that rephrases and combines similar entries while removing redundant or contradictory feedback. Each entry in $\mathcal{C}'_k$ should align with the corresponding skill template and provide additional guidance not currently included within that template. The size of $\mathcal{C}'_k$ is equal to or smaller than the size of the original $\mathcal{C}_k$, both in terms of the number of entries and the number of characters within those entries.

Clustering helps the skillbook generalize because it reasons across multiple pieces of related feedback and then

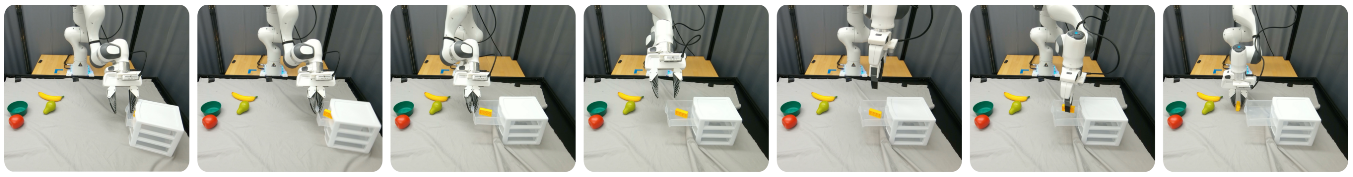


Fig. 3. Our setup for real-world experiments. Here the robot uses MEMO to "Empty the Cabinet" in a zero-shot manner.

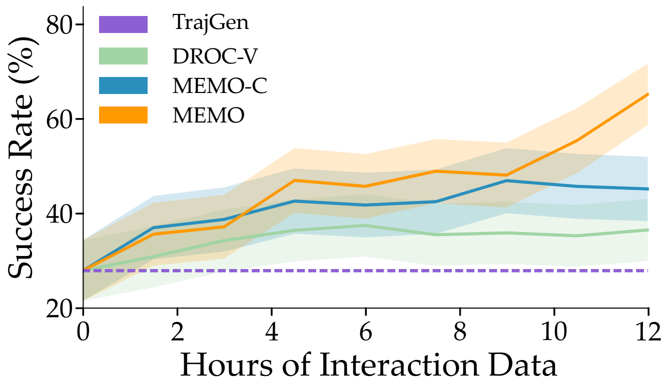


Fig. 4. Zero-shot success rate for our held-out evaluation tasks in simulation. The x-axis shows the size of the skillbook in terms of user hours. Note that as the skillbook grows, the performance of MEMO−C and DROC−V tends to stagnate without clustering local corrections into generalized guidance. The average zero-shot success rate at the onset of the study (with no entries in the skillbook) is approximately 30%, while as the skillbook grows in size, MEMO approaches a success rate of 80%.

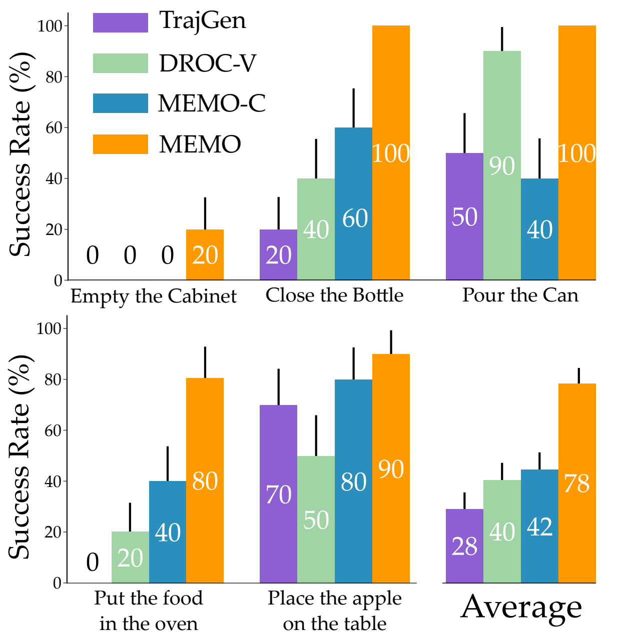


Fig. 5. Breaking down Figure 4 with the complete 12 hours of user feedback. Across the simulated evaluation tasks, MEMO achieves a higher zero-shot success rate than alternatives. Without the inclusion of offline clustering, MEMO is unable to generate the necessary skills for "Empty the Cabinet", "Close the Bottle", and "Pour the Can." In unseen tasks MEMO achieves a success rate of 78% as compared to 40% for DROC−V.

provides a summarized version. This also improves retrieval: instead of collecting 50 "opening" entries with varying levels of usefulness, now the robot can retrieve a more compact set of mutually exclusive and aligned guidance for Equation (2). Finally, because we condition clustering on templates from successful subtasks, the robot is able to prune erroneous feedback that may cause conflicts within the original skillbook.

V. Experiments

In this section we aim to answer three main questions in order to assess the capabilities of MEMO: 1) We examine whether incorporating entries retrieved from the skillbook improves zero-shot generalization to unseen task configurations; 2) We evaluate whether our clustering approach helps MEMO better understand tasks and resolve conflicting feedback while maintaining a compact skillbook; 3) We investigate cross-task learning by examining how new skills are formed in the skillbook in a real-world evaluation.

Evaluation Tasks. We evaluate MEMO in a tabletop manipulation setting across simulated and real-world experiments using a 7-DoF Franka Emika Panda robot with a UMI gripper. In simulation we leverage 25 tasks, twenty for creating our skillbook and the remaining five for evaluation. All 25 tasks are listed in Table I. We specifically selected tasks designed to challenge the system across multiple aspects, including long-horizon planning (**LH**), contact-rich manipulation (**CR**), semantic reasoning about objects and task goals (**SR**), and coordinating both translational and rotational control (**TR**). We ensure that at least one task of each category is represented in the five evaluation tasks.

We leverage the same five evaluation tasks in the simulated experiments and in the real world. The robot's control interface is identical between the simulated and real-world setups, i.e., the functions that control the robot require the same input parameters. For perception in the real world, we utilize an Orbbec Femto Mega RGB-D camera as well as two Intel RealSense D435 cameras for training and deployment of $\pi_{0.5}$, one mounted on the robot's wrist and one positioned as a side view camera. The side camera is configured to approximately match the field of view of the main Orbbec camera. Given these camera feeds, we leverage Gemini Robotics-ER 1.5 [21] to detect objects and LangSAM [22], [23] to obtain fine-grained segmentation masks to reconstruct 3D object bounding boxes. Finally, for specific object components (e.g., a cabinet handle) we attach an ArUco marker to estimate pose in SE(3) space. We use *gemini-3-flash-preview* as our large language model and *all-MiniLM-L6-v2* as our sentence transformer.

Metrics. To assess the performance of all methods across simulation and real-world tasks, we report the overall task success rate across five trials.

TABLE II

Performance on real-world tasks. We report the success percentage and the average amount of feedback per task (#). We find that **MEMO** reaches the highest overall performance with the least amount of feedback during evaluation.

Task Name	MEMO	MEMO−C	DROC−V [8]	$\pi_{0.5}$ [1]
Place the apple on the table	**100** (0.2)	**100** (0.6)	80 (1.4)	20
Pour the can	**100** (0.2)	**100** (1.2)	60 (1.8)	20
Close the bottle	**100** (1.0)	**100** (2.0)	80 (2.2)	0
Empty the cabinet	**80** (2.8)	**80** (3.2)	60 (3.8)	0
Put the food in the oven	**60** (3.4)	40 (4.0)	20 (4.6)	20
Overall Performance	**88** (**1.52**)	84 (2.2)	60 (2.76)	12

Baselines and Ablations. We compare MEMO against state-of-the-art baselines, as well as ablated versions of MEMO.

Our primary baselines are DROC−V [8] and $\pi_{0.5}$ [1]. DROC−V is a neuro-symbolic approach that stores human feedback to apply context-specific corrections. Intuitively, we can think of DROC−V as similar to MEMO, but without the code templates, general feedback, and clustering within our method. We provide the same set of human feedback to DROC−V as we use to build our skillbook. The baseline $\pi_{0.5}$ is a representative vision-language-action (VLA) model. We train $\pi_{0.5}$ on a set of real-world demonstrations composed of all 25 tasks utilized in simulation for 20000 steps, following the officially released Libero fine-tuning scheme [1].

We also ablate our method to test the following versions:

1) **MEMO−C:** We test our method without the *clustering* stage. This enables us to explore whether clustering helps MEMO obtain more generalized skills.
2) **MEMO−S** (TrajGen): We test our method without the use of any skillbook. Effectively, this version of MEMO is equivalent to **TrajGen** [3], but with the inclusion of an open vocabulary classifier.

A. Building the Skillbook: Human Subject Study

MEMO is centered around the creation of a skillbook that accumulates and clusters feedback from users in order to improve its capability to generalize to novel tasks. To this end, we utilize our 20 training tasks in simulation and ask 20 human subjects to provide free-form natural language feedback to improve the robot's behavior. The 20 participants are made up of 7 females and 13 males (age 24 ± 5.5) recruited from the Virginia Tech community; some participants chose to participate in the study multiple times. Eleven participants report familiarity with the use of large-language models "multiple times per week." Seven of the 20 participants report having no robotics experience prior to the study. All users provide informed written consent consistent with IRB #23-1237 and were compensated for their time.

Participants observe the robot's motion as part of the current subtask to a) determine if the robot succeeded in its task or b) provide unconstrained natural language feedback that they believe will improve task performance. The users could interrupt the robot at any time during task execution and provide feedback. When they did, their feedback was paraphrased by the language model, labeled with an *action-object(s)-scene* triplet corresponding to the current subtask, and added to the skillbook. The environment was then reset to the beginning of the respective subtask and the robot attempted to complete the task again, now with the previously provided feedback as a part of its skillbook. Table I lists the total number of feedback entries provided for each task. Each session lasted 30 minutes. During these 30 minutes, participants worked through as many tasks as possible following the order shown in Table I; each new participant picked up from where the previous participant left off. With this protocol the 20 participants provided three rounds of feedback. We found that 13 participants were required for the first iteration of all 20 tasks, 4 participants were able to complete the second iteration, and only 3 participants were needed for the third and final iteration.

B. Zero-Shot Generalization

Now that we have the skillbook collected in Section V-A, we are ready to test our first hypothesis — whether retrieved entries from the skillbook improve zero-shot performance. We evaluate in simulation on five held-out evaluation tasks (see Table I) for which MEMO has no prior feedback. We compare MEMO against three baselines: MEMO−C, MEMO−S (TrajGen), and DROC−V. During this evaluation no methods received feedback on the held-out tasks. Figure 4 shows that the addition of our skillbook consistently improves performance (78% task success) over TrajGen and DROC−V (40% and 28%, respectively). DROC−V has been trained by providing the same set of feedback for its offline storage (without our templated code generation), while TrajGen does not utilize any feedback (thus resulting in a horizontal line). We observe that our clustered skillbook increases performance — particularly for large amounts of feedback past the 10-hour mark — demonstrating the importance of feedback aggregation (36% improvement). Based on this zero-shot performance, we conclude that MEMO generalizes user feedback to similar tasks, and this generalization is due to the code templates and clustering.

C. Skillbook Clustering

As seen in Section V-B, clustering our skillbook significantly improves performance. To investigate *why* clustering helps, we separately assess MEMO's success rate for all five held-out tasks (see Figure 5). We observe that — with the clustered skillbook — MEMO is able to generate skills and complete tasks that it cannot regularly solve without this clustering (see 78% average success rate over 42% for the next-best method). Consider the *"Pour the Can"* task: when checking the skillbook, we find that clustering removes references that are not actually relevant to the task but may be erroneously retrieved (see 40% vs. 100% success rate). Indeed, MEMO$-$C actually falls below several of the baselines in Figure 5 because it retrieves incorrect or extraneous feedback from the skillbook, causing the policy to generate ineffective skills. These results highlight that compression plays a critical role in MEMO beyond just its simplification of the relevant context; compression also resolves erroneous retrieval of unrelated and conflicting context.

D. Real-World Evaluation

Thus far we have focused on simulation tasks for both constructing and testing the skillbook. We finally take this same skillbook and apply MEMO to a real-world robot arm where we can explore the transferability of our approach and the creation of the relevant skills from the skillbook (see Figure 3). We test the same five held-out evaluation tasks that were used in simulation. Table II shows the task success rate of MEMO, including the MEMO$-$C, DROC$-$V, and $\pi_{0.5}$ baselines. Each task is attempted five times while allowing for up to five instances of feedback per attempt. We observe that MEMO provides similar or better performance to MEMO$-$C, while requiring significantly less feedback (1.52 pieces of feedback for MEMO as compared to 2.76 for MEMO$-$C). Similarly, MEMO significantly outperforms DROC$-$V in both performance and issued feedback, and is also more effective than $\pi_{0.5}$ in its overall performance. We emphasize that the feedback which MEMO uses was collected in simulation, and not the real world; hence, the fact that MEMO can extrapolate code for new and performant skills suggests that there is some cross-task and cross-environment skill transfer.

VI. Conclusion

Neuro-symbolic policies rely on skills to ground their semantic subtasks into robot motions. To dynamically expand the skills that a robot arm can generate, we propose MEMO. MEMO is based on aggregating human language corrections and successful code snippets into a retrieval-augmented skillbook. Rather than simply storing and retrieving the most relevant entries, MEMO clusters the human's feedback while conditioning on successful code templates. The robot then reasons across these generalized functions and text guidance at run time, enabling the policy to generate code for new skills while still being informed by human feedback. Our experiments show that (a) using the skillbook improves generalization to unseen task configurations, (b) clustering the skillbook results in cross-task learning that forms new skills from existing ones, and (c) MEMO achieves a higher zero-shot success rate than neuro-symbolic approaches which only reason about relevant human feedback.

References

[1] P. Intelligence *et al.*, "$\pi_{0.5}$: a vision-language-action model with open-world generalization," in *Conference on Robot Learning (CoRL)*, 2025.

[2] Y. Dai, B. A. Christie, D. J. Evans, D. P. Losey, and S. Stepputtis, "Language movement primitives: Grounding language models in robot motion," *arXiv preprint arXiv:2602.02839*, 2026.

[3] T. Kwon, N. Di Palo, and E. Johns, "Language models as zero-shot trajectory generators," *IEEE Robotics and Automation Letters*, vol. 9, no. 7, pp. 6728–6735, 2024.

[4] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023.

[5] 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:2305.16291*, 2023.

[6] 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," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[7] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, *et al.*, "Do as I can and not as I say: Grounding language in robotic affordances," in *Conference on Robot Learning (CoRL)*, 2023.

[8] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, and D. Sadigh, "Distilling and retrieving generalizable knowledge for robot manipulation via language corrections," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[9] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, and Y. Chebotar, "Inner monologue: Embodied reasoning through planning with language models," in *Conference on Robot Learning*, pp. 1769–1782, 2023.

[10] 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.

[11] B. Zitkovich, T. Yu, S. Xu, P. Xu, *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*, pp. 2165–2183, 2023.

[12] O. M. Team *et al.*, "Octo: An open-source generalist robot policy," *arXiv preprint arXiv:2405.12213*, 2024.

[13] G. Tziafas and H. Kasaei, "Lifelong robot library learning: Bootstrapping composable and generalizable skills for embodied control with language models," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 515–522, 2024.

[14] X. Zhao, C. Weber, and S. Wermter, "Agentic skill discovery," *Robotics and Autonomous Systems*, p. 105248, 2025.

[15] M. Dalal, M. Liu, W. Talbott, C. Chen, D. Pathak, J. Zhang, and R. Salakhutdinov, "Local policies enable zero-shot long-horizon manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13875–13882, 2025.

[16] V. Myers, C. Zheng, O. Mees, K. Fang, and S. Levine, "Policy adaptation via language optimization: Decomposing tasks for few-shot imitation," in *Conference on Robot Learning*, pp. 1402–1426, 2025.

[17] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, and J. Humplik, "Language to rewards for robotic skill synthesis," in *CoRL*, 2023.

[18] Y. Zhu, Z. Ou, X. Mou, and J. Tang, "Retrieval-augmented embodied agents," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17985–17995, 2024.

[19] K. Qu, G. Lan, R. Zurbrügg, C. Chen, C. E. Mower, H. Bou-Ammar, and M. Hutter, "A pragmatist robot: Learning to plan tasks by experiencing the real world," *arXiv preprint arXiv:2507.16713*, 2026.

[20] R. Mon-Williams, G. Li, R. Long, W. Du, and C. G. Lucas, "Embodied large language models enable robots to complete complex tasks in unpredictable environments," *Nature Machine Intelligence*, vol. 7, no. 4, pp. 592–601, 2025.

[21] G. R. Team *et al.*, "Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer," *arXiv preprint arXiv:2510.03342*, 2025.

[22] S. Liu, Z. Zeng, T. Ren, *et al.*, "Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection," in *European Conference on Computer Vision (ECCV)*, pp. 38–55, 2024.

[23] L. Medeiros, "Lang segment anything: Segment anything model with text prompts." `https://github.com/luca-medeiros/lang-segment-anything.git`, 2025. accessed March 2, 2026.

[24] M. J. Kim, Y. Gao, T.-Y. Lin, Y.-C. Lin, Y. Ge, G. Lam, P. Liang, S. Song, M.-Y. Liu, and C. Finn, "Cosmos policy: Fine-tuning video models for visuomotor control and planning," *arXiv preprint arXiv:2601.16163*, 2026.

[25] M. J. Kim, K. Pertsch, S. Karamcheti, *et al.*, "OpenVLA: An open-source vision-language-action model," in *CoRL*, 2024.

[26] M. Kwon, Y. Kim, and Y. J. Kim, "Fast and accurate task planning using neuro-symbolic language models and multi-level goal decomposition," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16195–16201, 2025.

[27] Y. Du, M. Tian, S. Ronanki, S. Rongali, S. Bodapati, A. Galstyan, A. Wells, R. Schwartz, E. A. Huerta, and H. Peng, "Context length alone hurts LLM performance despite perfect retrieval," *arXiv preprint arXiv:2510.05381*, 2025.

[28] Y. Zhou, H. Liu, Z. Chen, Y. Tian, and B. Chen, "GSM-Infinite: How do your LLMs behave over infinitely increasing context length and reasoning complexity?," *arXiv preprint arXiv:2502.05252*, 2025.