2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Lifelong Robot Library Learning: Bootstrapping Composable and Generalizable Skills for Embodied Control with Language Models

Georgios Tziafas[1] and Hamidreza Kasaei[1]

***Abstract*— Large Language Models (LLMs) have emerged as a new paradigm for embodied reasoning and control, most recently by generating robot policy code that utilizes a custom library of vision and control primitive skills. However, prior arts fix their skills library and steer the LLM with carefully hand-crafted prompt engineering, limiting the agent to a stationary range of addressable tasks. In this work, we introduce `LRLL`, an LLM-based lifelong learning agent that continuously grows the robot skill library to tackle manipulation tasks of ever-growing complexity. `LRLL` achieves this with four novel contributions: 1) a soft memory module that allows dynamic storage and retrieval of past experiences to serve as context, 2) a self-guided exploration policy that proposes new tasks in simulation, 3) a skill abstractor that distills recent experiences into new library skills, and 4) a lifelong learning algorithm for enabling human users to bootstrap new skills with minimal online interaction. `LRLL` continuously transfers knowledge from the memory to the library, building composable, general and interpretable policies, while bypassing gradient-based optimization, thus relieving the learner from catastrophic forgetting. Empirical evaluation in a simulated tabletop environment shows that LRLL outperforms end-to-end and vanilla LLM approaches in the lifelong setup while learning skills that are transferable to the real world. Project material will become available at the webpage https://gtziafas.github.io/LRLL_project/.**

I. INTRODUCTION

Building interactive agents that can continuously develop new skills and adapt to new scenarios remains a challenging frontier in robotics [1]–[3]. Such an agent should be able to interface natural language, percepts and actions in order to form policies that are reusable and expandable in an open-ended fashion [4]. Recent advances in end-to-end robot learning [5]–[8] learn capable multimodal policies but require copious amounts of data, which are very hard to scale in the robotics domain. Further, their reliance in gradient-based optimization hinders their applicability in a lifelong setup, due to the effect of catastrophic forgetting [9]–[11].

Meanwhile, an emerging paradigm has been to leverage the code-writing capabilities of modern LLMs [12]–[14] for synthesizing executable robot policy code from natural language [15]–[19]. In such a setup, vision and action skills are implemented as modules (either learned or scripted) in a first-party API. This allows the LLM to compose them arbitrarily in combination with classic programming structures (control flow, recursion etc.) and third-party Python APIs (e.g. `numpy`) in order to ground visual observation, perform low-level reasoning, and provide parameters for control primitives. This system bypasses model finetuning,

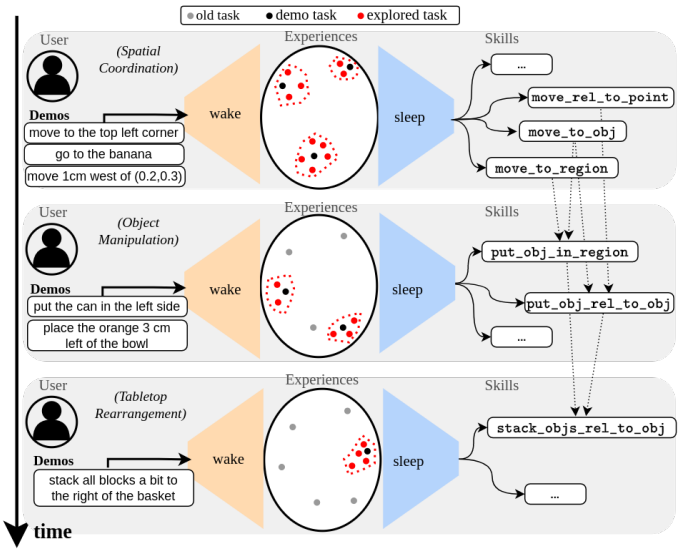

Fig. 1: Wake-sleep library learning from human guidance.

instead relying on careful prompt design and in-context examples to steer the LLM and aid generalization. However, the choice of the skills library and prompt examples remains a design choice that limits the span of tasks that the agent can tackle, and require an expert to continuously adapt the library and prompts to the LLM. In this work, we wish to address such limitations by proposing *LRLL*, a **L**ifelong **R**obot **L**ibrary **L**earning agent. *LRLL* learns hierarchical and generalizable skills across time spans, while staying within the regime of in-context learning and bringing non-expert human users in-the-loop.

Our learning algorithm is inspired by *wake-sleep optimization* [20] and its adaptation for library learning [21]. Learning takes place in cycles, each with two distinct phases: a) a *wake* phase, during which the agent interacts with its environment and users in order to grow its experiences, and b) a *sleep* phase, during which the agent reflects on its experiences in order to expand its capabilities. Accumulated experiences are distilled into skills throughout the learning cycles, therefore allowing complex tasks to be expressed as programs composed of simpler skills (see Fig 1). The human acts as a teacher, introducing a few demonstrations and hints to the agent during the wake phase. We assume the teacher follows a curriculum approach, where in each cycle the objective tasks can be built out of the learner's current repertoire of skills.

To stay within in-context learning, we use a frozen LLM

[1]Department of Artificial Intelligence, University of Groningen, the Netherlands, {g.tziafas,h.kasaei}@rug.nl

to generate the policy code, and design our algorithm as an interchange between two modules: a) an *experience memory*, where the agent can store and retrieve past instruction-code pairs based on similarity search, in order to feed context to the LLM (i.e. prompt retrieval), and b) a *skill library*, which comprises the collection of API calls the LLM can generate code from. At each cycle, the wake phase populates the memory with new instruction-code pairs, generated from an LLM-based exploration module that proposes and self-verifies new tasks in a simulator. During the sleep phase, the accumulated experiences are distilled into new skills via an LLM-based abstraction module. The new skills are appended to the library and the wake phase is replayed with refactored experiences, in order to compress the memory. This leads to a continual transfer of knowledge from human guidance, via exploration in simulation, to the library, without utilising gradients in any part of the process.

To apply our idea in robotics, we design a four-stage curriculum and prompt our agent to acquire a broad range of skills, including precise visual-spatial reasoning and long-horizon tabletop rearrangement. Empirically, we show that *LRLL* can automatically build a library of hierarchical, generalizable and interpretable skills, while outperforming end-to-end and stationary LLM baselines. We further perform ablations to demonstrate the effectiveness of each proposed component and explore design options. Finally, we illustrate that our algorithm can be transferred to a real robot for dual-arm tabletop rearrangement tasks, without any further adaptation. In summary, our key contributions are the following: a) *LRLL*, an LLM-based agent that can generate policy code, explore tasks in simulation, and expand its skillset over time, b) a formal recipe for enabling humans to bootstrap desired robot skills with minimal intervention, and c) extensive comparisons, ablation studies and hardware demonstrations that evaluate the effectiveness of each proposed component, assess overall generalization capabilities and test sim-to-real transferrability.

II. RELATED WORK

Language to Action Natural language has a long-standing history for controlling robots [4], serving both as a natural interface for human-robot interaction [22]–[25], as well as a generalizable intermediate representation [7], [26], [27]. Approaches range from semantic parsing [28]–[30], planning [31]–[34], reinforcement [35]–[37], imitation [7], [8], [26], [27], [38], and model-based [39]–[41] learning to more recent large-scale end-to-end multimodal instruction-following [5], [42]. While end-to-end policies are becoming more capable, they require prohibitive amounts of offline data or environment interactions. Further, their lifelong learning potential is limited by the effect of catastrophic forgetting [9]–[11]. In contrast, in this work we focus on a gradient-free approach where low-level actions are implemented as control primitives, out of which an LLM continuously builds more complex skills via few-shot human demonstration and interactive exploration.

LLMs for Robot Control More similar to this work, an emerging body of methods is chaining LLMs with external models [43]–[45] in order to propose grounded plans that sequence high-level actions [31]–[34]. This method invests on the current capabilities of LLMs for multi-step reasoning using external modules as tools [46], without adittional finetuning. Recent works [15]–[18] replace high-level actions with a library of primitives, and use the LLM to generate Python code that grounds the visual scene and parameterizes the primitives. This allows more complex policy logic than sequences of actions and offers more precise spatial grounding [19] and reasoning [15]. However, such works are stationary systems that do not further extend their library, and require manual prompt engineering to be applied in a general setup. *Code-as-Policies (CaP)* [15] demonstrates sparks of non-stationarity by letting the LLM recursively define unseen functions, but does not do so in a controlled, reusable fashion and does not consider human guidance. In this work, we wish to extend a *CaP*-like system to incorporate past interactions as a self-prompting mechanism and systematically use the LLM function generator to expand the skill library over time.

Memory and Context Retrieval Retrieval-augmented LLMs are a trending direction in NLP research [47]–[49], mostly as a means to reduce LLM hallucinations. In the robotics and embodied AI space, several works retrieve the most similar task-code pairs from memory based on similarity search and use them to prompt the LLM [17], [31], but do so in a stationary fashion. Recent works [50]–[52] expand the memory based on interactions, but there is no refinement of the base skills. In our work, we use the memory for LLM prompt retrieval, but progressively distill similar experiences to new skills that refactor and compress the memory.

Language-guided Skill Acquisition Iteratively refining robot skills with language feedback has been explored in the past with external parsers [53] and end-to-end language-conditioned policies [41], [54]–[56], but rely on either domain knowledge or extensive demonstration datasets, and therefore lack scalability. More recently, [57] leveraged LLMs with multi-step human feedback to generate reward functions that will train policies with model-predictive control. The concurrent work [58] utilizes an LLM to generate synthetic experiences in simulation together with success conditions, and distills the successful trials into a language-conditioned policy with behavioral cloning. Both ideas are conceptually close to our work, exploiting the LLM to generate information that will train a policy, but employ traditional approaches for learning the policy and hence struggle with the lifelong learning setup. Instead, *LRLL* leverages control primitives and an LLM to express the policy itself, and poses skill acquisition as library learning on top of the primitives. This allows complex skills to form from simpler skills over time, while retaining interpretability and bypassing training policies from scratch.

III. METHOD

In this section, we provide an overview of the proposed algorithm (Sec. III-A) and its components (Sec. III-B), and

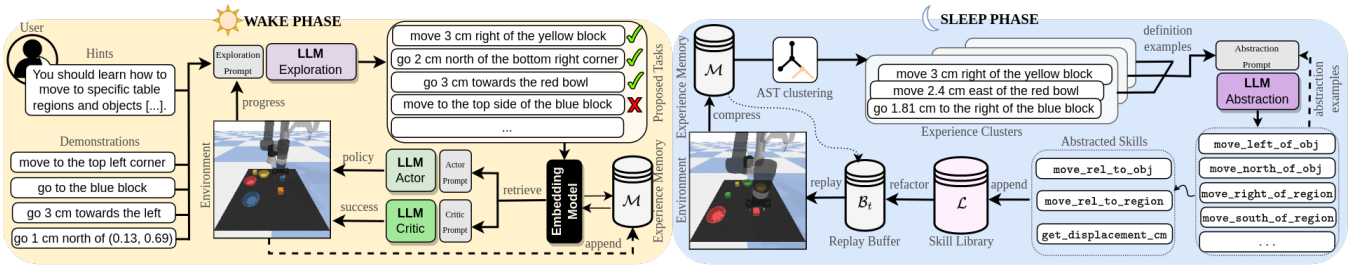

Fig. 2: Overview of an *LRLL* learning cycle. At the beginning of the wake phase, a human user provides demonstrations and hints, out of which an LLM-based *exploration* module proposes tasks to complete, while an LLM-based *actor-critic* agent interacts with the environment to execute and verify tasks. During sleep, the experiences are clustered according to their code's abstract syntax trees (AST) and distilled into new skills with an LLM *abstractor*. The new skills refactor the acquired experiences, which are replayed in the environment in order to compress the memory. We note that for brevity purposes, we omit showing the actor-critic modules during replay at the sleep phase. We illustrate examples from our first curriculum cycle (*spatial coordination*).

describe details for achieving policy and success generation, exploration and skill abstraction with LLMs (Sec. III-C).

A. LRLL Overview

Our framework (see Fig. 2) receives at each cycle t a set of N_t language demonstrations X_t that contain instruction-policy-success tuples for specific tasks: $X_t = \left\langle l^{(i)}, a^{(i)}, r^{(i)} \right\rangle_{i=1}^{N_t}$. The demos are appended to the agent's memory M_t. The policy code $a^{(i)}$ is factorized based on the current library skills L_t. The success code $r^{(i)}$ describes how the agent can use privileged simulation data to verify the success of a given instruction (e.g. check contact, object poses etc.). The output is a set of formal skills, expressed as Python functions, which can be directly used for real robot deployment. This process is repeated for an open-ended number of cycles, enabling the agent to expand its library in lifelong fashion. Each cycle consists of two phases:

Wake Phase During the wake phase, the agent interacts with its environment in order to grow its proficiency in solving more tasks. We use an LLM to iteratively propose new task instructions $l_{1:K} = \texttt{LLMExplore}(X_t, G_t, M_t)$ based on the input demos X_t, the current memory state M_t and some general hints G_t. The proposed tasks are executed and verified in the simulator using another LLM's generated policy and success code respectively: $a_k, r_k = \texttt{LLMActorCritic}(\, l_k, s_{0,k}, M_t)$, where $s_{0,k}$ the initial simulator state of proposal k. Successful tuples are appended to the experience memory M_t, which helps the agent to recover more relevant context throughout exploration. The process is repeated until an iteration threshold is met, or the LLM decides that it has completed all objectives denoted in the hints. The acquired experiences are stored in a replay buffer $B_t = \left\{ < s_{0,k}, l_k, a_k, r_k > \mid r_k = 1 \right\}$.

Sleep Phase During sleep, the agent reflects on its acquired experiences to compose new skills. To achieve this, we first represent each experience as an abstract syntax tree of its policy code. Experiences are clustered such that codes that have the same tree structure modulo variable and constant names are grouped together for a total of M clusters $C_{1:M}$. We then feed each cluster into an LLM that uses the experiences as examples to define new functions that will update the library $L_{t+1} = L_t \cup \{\texttt{LLMAbstract}(C_m)\}_{m=1}^{M}$. The demo policies are refactored based on the proposed functions $\tilde{X}^{(i)} = \texttt{Refactor}(X^{(i)}, L_{t+1})$, and the wake phase is replayed from scratch, starting only with the refactored demos. When the agent fails in a previously succeeded task, its policy-success is also refactored and appended to the memory: $M_{t+1} = M_t \cup \left\{ \tilde{X}^{(i)} \in B_t \mid r^{(i)} = 0 \right\}$ This process ensures that by the end of the cycle, the memory will contain the minimum number of experiences needed to replicate the performance of the wake phase.

B. System Components

Initial Vision & Action Primitives Following previous works [15], [16], we employ frozen pretrained vision-language models for zero-shot vision-language grounding In particular, we use MDETR [45] for referring expression grounding and CLIP [43] for open-vocabulary classification. For control, as we wish to demonstrate the capability of LRLL to progressively build complex skills from simpler ones, we start with the most basic primitives: moving the arm to a certain pose and opening / closing the gripper. Motion planning is performed via inverse kinematics from end-effector space.

Experience Memory & Retrieval Agent experiences are formalized as tuples of task instruction, action and success code, either provided by the human teacher or "imagined" by the LLM exploration module $X^{(i)} =< l^{(i)}, a^{(i)}, r^{(i)} >$. When appended to the memory, each experience is indexed by its instruction embedding, provided by an encoder-based LM [59], [60] $\mathbf{z}_i = F^{LM}(l^{(i)})$. In order to retrieve experiences to serve as prompts, the query instruction q is embedded by the same model $\mathbf{z}_q$ and the experiences of the top-k most similar instructions based on maximum marginal relevance search [61] are returned:

$$\texttt{argmax}_{i \in M_t \setminus S} \left[\lambda (\texttt{cos}(\mathbf{z}_i, \mathbf{z}_q) - (1-\lambda) \texttt{max}_{j \in S} \texttt{cos}(\mathbf{z}_i, \mathbf{z}_j)) \right]$$

where S the set of already selected retrievals, $\texttt{cos}$ the cosine distance metric and λ a diversification hyper-parameter. This rule is applied k times to retrieve diverse experiences.

Skill Library Each agent skill corresponds to a function, implemented as a Python API. The library maintains skill information such as their names and descriptions, and is able to trace skill dependencies from a given code snippet.

Before learning begins, the library is initialized with the initial primitives. A wrapper around the library and the agent converts the newly acquired skills into API modules that are executable in a robot simulator.

Replay Buffer The replay buffer is a replica of the experience memory but only for the explored experiences of the current cycle. Additionally, for each experience, the simulator states are saved. The replay buffer is reset at the beginning of each new cycle.

C. LLM Prompts

We implement three LLM modules:

Actor-Critic The actor-critic comprises of two parallel LLM calls, one for policy and one for success code generation. Both prompt templates are instantiated after retrieving experiences from the memory for a given query, and follow the same general structure:

- A comment indicating a **general purpose** of the code (e.g. *"### Python robot control script"*).
- Information about the **API**, aiming to present the building blocks for code generation. For the actor, all modules are extracted from the retrieved examples' policies and rendered as import statements [15]. For the critic, a fixed code snippet describing simulator utilities via docstrings is provided [58].
- A sequence of task-code pairs from the **retrieved experiences**. Each pair is rendered as a comment of the task description followed by a policy or success code snippet, for the actor and critic respectively.
- Throughout demonstration code, **chain-of-thoughts** [62] are provided as in-line comments to guide the LLM's reasoning before producing a next line, which is especially useful for explaining perspective conventions (e.g. *"left"* correspond to x-axis).

For inference, the query is appended to the prompt as a comment and the LLM fills the corresponding policy or success code. We find that this code-based completion format works robustly also for chat-based LLMs [63].

Exploration The exploration module proposes the next tasks to complete in the simulator. The goal is to use the demos as guidance and introduce both task *variations*, i.e. alter concepts present in the instruction (e.g. color, spatial direction etc.), as well as task *compositions*, i.e. combinations of concepts present in the demos (e.g. desired destinations for placing objects). The prompt contains:

- A system message that includes **general directives** that condition the LLM for the task, encourages diverse responses and provides the required response format.
- **Hints** provided by the teacher, aimed to provide objectives of a specific cycle.
- Information about the current **state**, represented as a list of appearing object names.
- A list of completed and failed tasks so far, reflecting the agent's current **progress** towards completing all objectives mentioned in the hints. The completed tasks are initialized with the provided demos.
- A set of two **exemplar** generations, containing input demo-hints and output task proposals. We provide one manual and set one more exemplar from the LLM's first response in the previous cycle.

Before proposing tasks, we ask the LLM to reason about its proposals [51], which significantly helps in responding better to the provided hints. We find that decomposing exploration to a chain of two LLM calls, prompted separately for compositions and variations, leads to faster completion. Task variations, proposed by the second LLM call, are not included in the progress prompt field. A temperature parameter of 0.1 is set at successive iterations to encourage diverse responses.

Skill Abstraction This module leverages LLMs' capabilities to define Python functions out of examples. The goal is dual: a) maintain the same code logic, but abstract code variations such as target objects and destination regions as arguments to the new function, and b) extract boilerplate code snippets and abstract them to new functions. This is achieved by prompting the LLM in two rounds. The prompt contains:

- A **general purpose** system message that primes the LLM for function generation and imposes constraints.
- An **API** field, rendered from the dependencies of input code snippets as an import statement.
- A set of two **exemplar** function definitions. The exemplars are different for each round of abstraction. As in exploration, one exemplar is manual and the other is selected from the first LLM response of the last cycle.
- The input code snippets with their instruction as comments. We first provide **definition examples**, which are the raw instruction-code pairs from each cluster of the experience memory, and then **abstraction examples**, which correspond to the LLM-generated functions from the first round.

We also ask the LLM to provide a docstring, which is used as the skill description, as well as to re-write the given examples based on the generated function, which is used to refactor the memory and move to the replay stage of the sleep phase.

IV. EXPERIMENTS

The focus of our experimental evaluation is threefold: a) compare our method against previous baselines for tabletop manipulation in simulation (Sec.IV-B), b) evaluate the impact of each our method's proposed contributions (Sec.IV-C), and c) demonstrate the transferability of our approach to the real world (Sec.IV-D).

A. Evaluation Setup

Implementation We leverage OpenAI's `gpt-3.5-turbo` [63] engine for all LLM generations, and `text-embedding-ada-002` [60] as the memory embedding model. Our system is built using the LangChain library [64]. Our simulator environment is built on Pybullet [65] and it is based on the Ravens [66] manipulation suite, with the blocks-and-bowls setup replicated from previous works [15], [34]. We introduce more tasks and language variations, for a total of 41 task templates, organized in a

TABLE I: Averaged success rates (%) over seen instructions with seen/unseen attributes (SA/UA) and unseen instructions (UI), organized in a 4-cycle curriculum with 10 trials per instruction. Best results are in bold.

Tasks	**CLIPort** [27]			**LLM-static** [15]			**LRLL** (ours)		
	SA	UA	UI	SA	UA	UI	SA	UA	UI
Spatial Coord/on	-	-	-	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Visual Reasoning	-	-	-	90.0	83.3	66.6	**91.7**	**94.0**	**85.1**
Object Manip/on	**98.3**	37.1	4.1	95.0	94.1	80.0	98.1	**98.9**	**90.4**
Rearrangement	70.8	13.9	0.3	93.0	90.0	60.6	**97.0**	**95.4**	**70.9**
Average	84.5	25.5	2.2	94.5	91.9	76.8	**96.9**	**97.1**	**86.6**

TABLE II: Averaged success rates (%) over unseen task combinations (FT) and a subset of previous task combinations (BT) for each cycle and baseline. Best results are in bold.

Tasks	**LLM-static** [15]		**LRLL-no-wake**		**LRLL-no-sleep**		**LRLL**	
	FT	BT	FT	BT	FT	BT	FT	BT
Spatial Coord/on	**100.0**	**100.0**	68.7	60.0	**100.0**	**100.0**	**100.0**	**100.0**
Visual Reasoning	60.0	**100.0**	40.0	55.0	**80.0**	**100.0**	**80.0**	**100.0**
Object Manip/on	55.3	60.0	46.6	40.4	78.4	80.0	**94.0**	**80.0**
Rearrangement	45.7	55.3	50.0	46.6	64.0	78.4	**70.2**	**94.0**
Average	65.1	78.9	51.3	50.5	80.6	89.6	**88.1**	**93.5**

curriculum of 4 cycles: a) *Spatial Coordination*, i.e. precise motions relative to objects/regions, b) *Visual Reasoning*, i.e. determining attributes, resolving spatial relations and counting/enumerating objects, c) *Object Manipulation*, i.e. single picking, releasing and placing tasks, and d) *Rearrangement*, i.e. long-horizon tasks that involve multiple objects and destinations.

Evaluation For conducting generalization experiments, we generate task instances in three splits [15]: seen instructions with either seen (SA) or unseen (UA) attributes, and unseen instructions with unseen attributes (UI). For studying learning in the lifelong setup, we also propose two more splits: a) a *forward-transfer (FT)* split, which contains unseen compositions of tasks from the present cycle with all previous tasks (with seen attributes), and b) *backward-transfer (BT)*, which contains the FT tasks from the previous cycle. These splits are meant to study whether the agent can learn to transfer knowledge between tasks (FT), and to what extent it "forgets" or improves on previous tasks (BT).

Baselines We consider four baselines: a) learning end-to-end multi-task policies with *CLIPort* [27], adapted as in [34] (not applicable in all tasks), b) prompting LLMs for primitive-based policy code using a static prompt, as in *CaP* [15] (without hand-crafted routing between LLM subsystems), c) *LRLL-no-sleep*, where we remove the sleep phases from our *LRLL* and only retrieve examples from the memory without abstraction, and d) *LRLL-no-wake*, where we attempt to synthesize new skills directly from the human demonstrations, without the exploration of the wake phase.

B. Tabletop Manipulation in Simulation

We first wish to evaluate the performance of *LRLL* compared to established baselines in our simulated tabletop domain. To that end, we developed a simulated teacher that samples tasks from a set of predefined templates. The teacher generates up to 5 demonstration (1 per SA template) and multiple test (10 per UA, UI template) tasks in the beginning and end of each of our 4 cycles. For end-to-end learning with CLIPort [27], we sample $1k$ trajectories per task template using a scripted expect for each cycle and train the model incrementally. For our LLM-static baseline [15], we append demos from each new cycle in the LLM's prompt. The same demos are provided to *LRLL* at the beginning of each cycle's wake phase. Agents are tested at the end of each cycle. We repeat our experiments three times with different teacher seeds and report averaged success rates in Table I.

We observe that CLIPort struggles with unseen attributes and its performance degrades drastically with unseen instructions. LLM-static is robust to unseen attributes (2.9% average drop) and can generalize significantly better in unseen task instructions, with an average success rate of 76.8% in all cycles. We find that this baseline's main limitation is producing non-executable code in cases of unseen instructions at later cycles, which we attribute to its inability to interpret and compose multiple skills from a limited demonstration context. Such skill compositions are (partially) already explored during the wake phase of our *LRLL*, and abstracted to functions during the sleep phase, resulting in policy code that is much shorter and functional in style. This robustifies *LRLL*'s generated policies, which translates to an average increase of $\sim 6\%$ in unseen attribute and $\sim 10\%$ in unseen instructions compared to LLM-static.

C. Ablation Studies

Our ablations focus on exploring the effect of each of our proposed components and discussing options for implementation.

Forward/Backward Transfer We compare the averaged success rates of all baselines in FT/BT instructions. Results are reported in Table II. First, we assess that all baselines are robust in tasks from previous cycles, showcasing the immunity of LLM's in-context learning to forgetting. However, no actual increase in backward task's performance is reported in any baseline. For forward transfer, we see a large increase in averaged success between LLM-static and *LRLL*. Even without refactoring code (*LRLL-no-sleep* baseline), retrieving explored experiences leads to better compositional abilities, with a $\sim 15\%$ delta from static.

Static Prompts vs. Retrieval When prompted with a few examples, the difference between a static and a retrieved prompt is marginal. In the late cycles of the curriculum, we observe the effect of *prompt saturation* [15], [34] kicking in the static baseline, leading to several instabilities in the LLM responses, such as ignoring the first examples in favour of more recent ones or referring to variable names outside the current scope. Retrieval-based baselines tackle such issues by ensuring a fixed context length for the LLM actor.

Effect of Exploration The contribution of the exploration module is vital, as the performance of *LRLL-no-wake* is consistently much lower across cycles. This is due to the high difficulty of abstracting skills from one-shot demos, which usually leads to a one-to-one mapping of instructions to functions, without adding any actual refactoring. When adding

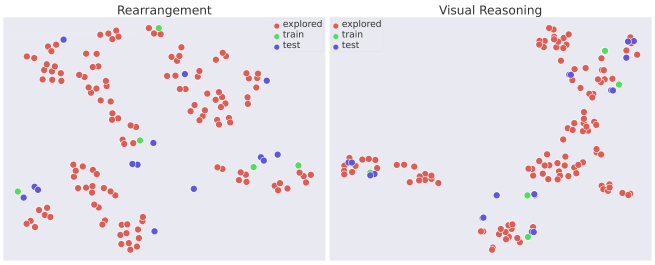

Fig. 3: *tSNE* projections of train (SA), test (UA+UI) and explored task instruction embeddings, for two of our curriculum cycles: a) *Visual Reasoning* (right), and b) *Rearrangement* (left). The exploration module augments the agent's experiences with task variations that cover a broad range of skill compositions from the demos. *(Best viewed in color).*

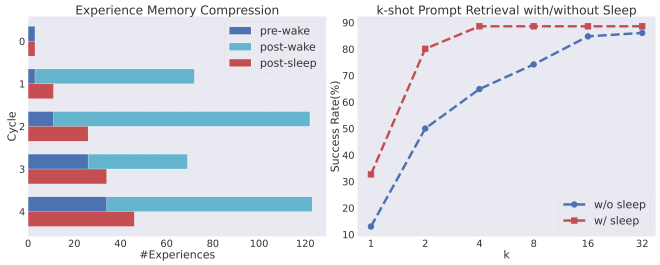

Fig. 4: (Left): Number of stored experiences per cycle before and after the sleep phase of *LRLL*. Sleep helps to compress the number of experiences needed to reach the same performance. (Right): Averaged success rates in all unseen instructions vs. number of retrieved experiences. Sleep phase refactors experiences, which leads to sufficient context with fewer examples.

exploration, the abstractor has significantly more examples to define new skills. To evaluate the breadth of variance in the explored tasks, we visualize the *tSNE* projections of their instruction embeddings [60] compared to demonstration and test tasks within a cycle (see Fig. 3).

Effect of Abstraction *LRLL-no-sleep* never abstracts the explored tasks into new skills, and so needs to retrieve a lot of examples in order to obtain sufficient context. This effect bottlenecks the agent to the quality of the retriever. Besides performance gains, sleep leads to other practical benefits (see Fig. 4). First, the amount of experiences required to maintain the same success within each cycle is drastically reduced, leading to a $\times 8$ decrease in RAM required to store experience embeddings. Second, *LRLL* with refactored memory requires much less retrieved experiences to maintain high performance in UI tasks. Additionally, as the experiences are refactored to be simple function calls (to the abstracted skills), the retrieved code is itself smaller, which leads to smaller prompt lengths and hence cost gains for using GPT.

LLM and Embedding Models We find no significant difference between `text-davinci-003` and `gpt-3.5-turbo` in the quality of generated policy code or function abstraction. For exploration, we find that both models provide rich variance in the proposed tasks, but the chat model tends to be less responsive to the hints signal. This effect can be ameliorated by running more exploration iterations. The choice of the embedding model is more important, as experiments with smaller encoder LMs such as Sentence-BERT [59] showed a tendency to retrieve instructions that are similar lexically (e.g. same object noun appears), but not necessarily convey the same task.

D. Zero-Shot Sim-to-Real Transfer

We repeat our curriculum with *LRLL* in a dual-arm robot setup with two UR5e arms and a Kinect sensor. We provide vision APIs for open-vocabulary detection with MDETR [45] and attribute recognition with CLIP [43]. To assist in articulated grasping, we also integrate GR-ConvNet [67] for 4-DoF grasp synthesis as a vision API. The motion primitives for moving the arm and opening/closing the fingers are parameterized by the left or right arm. We include a catalog of 12 household objects, including fruits, soda cans, juice boxes etc. We first train LRLL using our default 4-cycle curriculum in the Gazebo simulator [68] and then test the agent in the real robot. We demonstrate that the robot is able to perform long-horizon rearrangement tasks that combine precise spatial positioning with reasoning about object attributes, without any further adaptation from simulation. Errors were observed mostly at motion execution due to collisions, as well as perception errors due to CLIP misclassifications.

V. CONCLUSION & LIMITATIONS

In this work, we introduce *LRLL*, an agent and learning algorithm for lifelong robot manipulation. *LRLL* exploits the emergent capabilities of modern LLMs to: a) generate policies as code, b) interact with the environment to explore new tasks, and c) distill the acquired experiences into new skills over time. *LRLL* replaces tedious prompt engineering with retrieval from memory, and a static library of skills with an expandable codebase, written and verified by the agent itself. Empirical evaluation shows that our agent learns a library of composable, generalizable, and interpretable skills that can be transferred to the real world, while its dynamic and gradient-free nature prevents it from prompt saturation and forgetting phenomena of stationary-LLM and end-to-end approaches respectively.

LRLL comes not without limitations. First, perception is restricted by the choice of vision APIs, which currently support only referring expressions and attribute classification. In the future, we would like to look at multimodal LLMs [16], [69] for open-ended vision-language grounding. Second, the current human demonstration input to *LRLL* is language, limiting its scalability to skills that can be expressed symbolically as primitive compositions. For articulated, contact-rich manipulation tasks, we would like to augment demonstration input to support video or kinesthetic teaching. Third, the initial prompts to exploration/abstraction modules need to be refined when changing domains or LLM engines. Finally, exploration with commercial LLMs is constrained by latency and price factors. In the future, we would like to investigate the gap between GPT and open-source alternatives [70].

VI. ACKNOWLEDGMENTS

We thank the Center for Information Technology of the University of Groningen for providing access to the Hábrók high-performance computing cluster.

REFERENCES

[1] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Kumar Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *ArXiv*, abs/1712.05474, 2017.

[2] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4495–4504, 2021.

[3] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9338–9346, 2019.

[4] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annu. Rev. Control. Robotics Auton. Syst.*, 3:25–55, 2020.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Krzysztof Choromanski, Tianli Ding, Danny Driess, Chelsea Finn, Peter R. Florence, Chuyuan Fu, and et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *ArXiv*, abs/2307.15818, 2023.

[6] Danny Driess, F. Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Ho Vuong, Tianhe Yu, and Wenlong Huang et al. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning*, 2023.

[7] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi (Jim) Fan. Vima: General robot manipulation with multimodal prompts. *ArXiv*, abs/2210.03094, 2022.

[8] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert K. Baruch, Travis Armstrong, and Peter R. Florence. Interactive language: Talking to robots in real time. *ArXiv*, abs/2210.06407, 2022.

[9] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz Rodríguez. Continual learning for robotics. *ArXiv*, abs/1907.00182, 2019.

[10] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks : the official journal of the International Neural Network Society*, 113:54–71, 2018.

[11] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *ArXiv*, abs/2302.00487, 2023.

[12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, and Amanda Askell et al. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[13] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, and Shruti Bhosale et al. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023.

[14] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, and Sebastian Gehrmann et al. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311, 2022.

[15] Jacky Liang, Wenlong Huang, F. Xia, Peng Xu, Karol Hausman, Brian Ichter, Peter R. Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2022.

[16] Siyuan Huang, Zhengkai Jiang, Hao-Wen Dong, Yu Jiao Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *ArXiv*, abs/2305.11176, 2023.

[17] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2022.

[18] Sai Vemprala, Rogerio Bonatti, Arthur Fender C. Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *ArXiv*, abs/2306.17582, 2023.

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

[20] Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and R M Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268 5214:1158–61, 1995.

[21] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke B. Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021.

[22] Dipendra Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35:281 – 300, 2014.

[23] Jun Hatori, Yuta Kikuchi, Sosuke Kobayashi, K. Takahashi, Yuta Tsuboi, Yuya Unno, Wilson Kien Ho Ko, and Jethro Tan. Interactively picking real-world objects with unconstrained spoken language instructions. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3774–3781, 2017.

[24] Mohit Shridhar and David Hsu. Interactive visual grounding of referring expressions for human-robot interaction. *ArXiv*, abs/1806.03831, 2018.

[25] Hanbo Zhang, Yunfan Lu, Cunjun Yu, David Hsu, Xuguang Lan, and Nanning Zheng. Invigorate: Interactive visual grounding and grasping in clutter. *ArXiv*, abs/2108.11092, 2021.

[26] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. *ArXiv*, abs/2202.02005, 2022.

[27] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. *ArXiv*, abs/2109.12098, 2021.

[28] Jesse Thomason, Shiqi Zhang, Raymond J. Mooney, and Peter Stone. Learning to interpret natural language commands through human-robot dialog. In *International Joint Conference on Artificial Intelligence*, 2015.

[29] Renhao Wang, Jiayuan Mao, Joy Hsu, Hang Zhao, Jiajun Wu, and Yang Gao. Programmatically grounded, compositionally generalizable robotic manipulation. *ArXiv*, abs/2304.13826, 2023.

[30] Giorgos Tziafas and Hamidreza Kasaei. Enhancing interpretability and interactivity in robot manipulation: A neurosymbolic approach. 2022.

[31] Wenlong Huang, P. Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ArXiv*, abs/2201.07207, 2022.

[32] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Jayant Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego M Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, F. Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022.

[33] Wenlong Huang, F. Xia, Ted Xiao, Harris Chan, Jacky Liang, Peter R. Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2022.

[34] Andy Zeng, Adrian S. Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Peter R. Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *ArXiv*, abs/2204.00598, 2022.

[35] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim

Rocktäschel. A survey of reinforcement learning informed by natural language. *ArXiv*, abs/1906.03926, 2019.

[36] Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. *ArXiv*, abs/2007.15543, 2020.

[37] Yiding Jiang, Shixiang Shane Gu, Kevin P. Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *Neural Information Processing Systems*, 2019.

[38] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *ArXiv*, abs/2010.12083, 2020.

[39] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, 2021.

[40] Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In *North American Chapter of the Association for Computational Linguistics*, 2017.

[41] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. *ArXiv*, abs/2204.05186, 2022.

[42] Yao Mu, Qinglong Zhang, Mengkang Hu, Wen Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Y. Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *ArXiv*, abs/2305.15021, 2023.

[43] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.

[44] Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. In *International Conference on Learning Representations*, 2021.

[45] Aishwarya Kamath, Mannat Singh, Yann LeCun, Ishan Misra, Gabriel Synnaeve, and Nicolas Carion. Mdetr - modulated detection for end-to-end multi-modal understanding. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1760–1770, 2021.

[46] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022.

[47] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. Replug: Retrieval-augmented black-box language models. *ArXiv*, abs/2301.12652, 2023.

[48] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Li-Yu Daisy Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *ArXiv*, abs/2305.06983, 2023.

[49] Reiichiro Nakano, Jacob Hilton, S. Arun Balaji, Jeff Wu, Ouyang Long, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332, 2021.

[50] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *ArXiv*, abs/2304.03442, 2023.

[51] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi (Jim) Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *ArXiv*, abs/2305.16291, 2023.

[52] Difei Gao, Lei Ji, Luowei Zhou, Kevin Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *ArXiv*, abs/2306.08640, 2023.

[53] Alexander Broad, Jacob Arkin, Nathan D. Ratliff, Thomas M. Howard, and Brenna Argall. Real-time natural language corrections for assistive robotic manipulators. *The International Journal of Robotics Research*, 36:684 – 698, 2017.

[54] Yuchen Cui, Siddharth Karamcheti, Raj Palleti, Nidhya Shivakumar, Percy Liang, and Dorsa Sadigh. No, to the right: Online language corrections for robotic manipulation via shared autonomy. *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 2023.

[55] Arthur Fender C. Bucker, Luis F. C. Figueredo, Sami Haddadin, Ashish Kapoor, Shuang Ma, and Rogerio Bonatti. Reshaping robot trajectories using natural language commands: A study of multi-modal data alignment using transformers. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 978–984, 2022.

[56] Arthur Fender C. Bucker, Luis F. C. Figueredo, Sami Haddadin, Ashish Kapoor, Shuang Ma, Sai Vemprala, and Rogerio Bonatti. Latte: Language trajectory transformer. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7287–7294, 2022.

[57] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Manfred Otto Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and F. Xia. Language to rewards for robotic skill synthesis. *ArXiv*, abs/2306.08647, 2023.

[58] Huy Ha, Peter R. Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. *ArXiv*, abs/2307.14535, 2023.

[59] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*, 2019.

[60] New and improved embedding model, openai, https://openai.com/blog/new-and-improved-embedding-model. 2022.

[61] Jaime G. Carbonell and Jade Goldstein-Stewart. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998.

[62] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

[63] Introducing chatgpt, openai, https://openai.com/blog/chatgpt. 2022.

[64] Oguzhan Topsakal and Tahir Cetin Akinci. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. *International Conference on Applied Engineering and Natural Sciences*, 2023.

[65] Benjamin Ellenberger. Pybullet gymperium. `https://github.com/benelot/pybullet-gym`, 2018–2019.

[66] Andy Zeng, Peter R. Florence, Jonathan Tompson, Stefan Welker, Jonathan M. Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, 2020.

[67] Sulabh Kumra, Shirin Joshi, and Ferat Sahin. Antipodal robotic grasping using generative residual convolutional neural network. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9626–9633, 2019.

[68] Nathan P. Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154 vol.3, 2004.

[69] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *ArXiv*, abs/2304.08485, 2023.

[70] Hugo Touvron et. al. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023.