

BLAZER: Bootstrapping LLM-based Manipulation Agents with Zero-Shot Data Generation

Rocktim Jyoti Das^{*1}, Harsh Singh^{*1}, Diana Turmakhan^{†1}, Muhammad Abdullah Sohail^{†1}, Mingfei Han¹, Preslav Nakov¹, Fabio Pizzati¹, Ivan Laptev¹

Abstract—Scaling data and models has played a pivotal role in the remarkable progress of computer vision and language. Inspired by these domains, recent efforts in robotics have similarly focused on scaling both data and model size to develop more generalizable and robust policies. However, unlike vision and language, robotics lacks access to internet-scale demonstrations across diverse robotic tasks and environments. As a result, the scale of existing datasets typically suffers from the need for manual data collection and curation. To address this problem, here we propose BLAZER, a framework that learns manipulation policies from *automatically generated training data*. We build on the zero-shot capabilities of LLM planners and automatically generate demonstrations for diverse manipulation tasks in simulation. Successful examples are then used to finetune an LLM and to improve its planning capabilities without human supervision. Notably, while BLAZER training requires access to the simulator’s state, we demonstrate direct transfer of acquired skills to sensor-based manipulation. Through extensive experiments, we show BLAZER to significantly improve zero-shot manipulation in both simulated and real environments. Moreover, BLAZER improves on tasks outside of its training pool and enables downscaling of LLM models. Our code and data will be made publicly available on the project page [1].

I. INTRODUCTION

“A teacher is one who makes himself progressively unnecessary.”
— Thomas Carruthers

Learning-based methods are attracting increasingly growing attention in robotics. In particular, extending large language models (LLMs) and vision-language models (VLMs) to robotic tasks promise to empower resulting policies with strong reasoning capabilities and generalization to diverse tasks and environments [2], [3], [4], [5]. However, training generic robotic policies requires large-scale data in the form of paired actions and observations. To address this challenge, recent efforts attempt to collect real-robot demonstrations [6], adopt human videos [7], and leverage simulated environments [8], [9], [10]. Manual collection of robot demonstrations, however, is slow and expensive, while human videos are missing low-level control data and face an embodiment gap between robots and humans. Hence, scaling robotic demonstrations remains a major bottleneck.

Recent work on language models brought significant progress improving reasoning capabilities of LLMs [12],

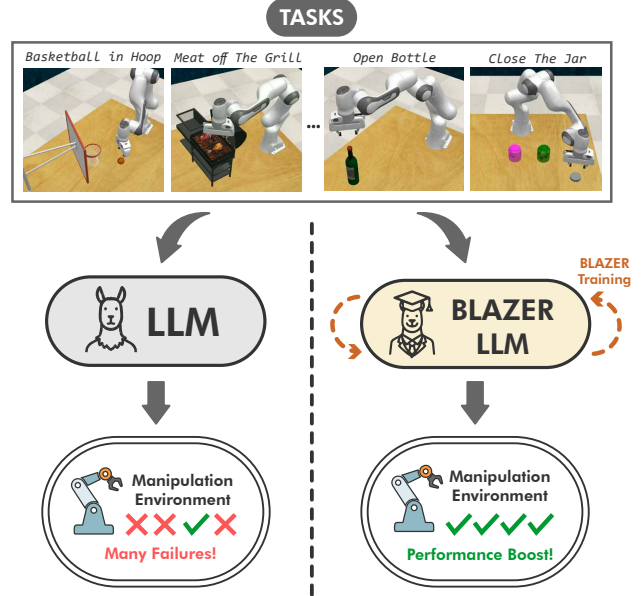


Fig. 1: **BLAZER overview.** Previous approaches such as Code As Policies (CAP) [11] (left) use LLMs to produce interaction plans and to solve manipulation tasks in a zero-shot manner. Such methods rely on careful prompt engineering and often lead to suboptimal performance. In contrast, BLAZER (right) uses a fully automatic pipeline, where successful LLM-generated demonstrations are used to train improved LLM-based manipulation agents with no manual supervision.

[13], [14]. One successful line of work in this direction is based on bootstrapping [13], where pretrained LLMs are first used to generate high-quality rationales and are then finetuned to yield improved performance while using generated rationales as training examples. Bootstrapping and self-improvement have been shown to be particularly effective for problems that require non-trivial solutions and offer simple verification, as for the case of many problems in mathematics and common sense reasoning [15]. We note that robotic manipulation tasks frequently belong to such class of problems since their success can often be certified by merely verifying the end states of manipulated objects.

Inspired by the idea of bootstrapping, we propose BLAZER, a framework that bootstraps LLM-based manipulation agents using automatically generated and verified demonstrations. Given language-defined tasks, such as “stack blocks” or “open wine bottle”, we follow previous

^{*}Co-first author.

[†]Co-second author.

¹Mohamed bin Zayed University of Artificial Intelligence, UAE
email: firstname.lastname@mbzuai.ac.ae,

Project Page: <https://blazer-llm-agent.github.io/>

work [11], [16] and use general-purpose LLM with strong reasoning and coding capabilities to generate executable manipulation plans. Next, we execute such plans in a simulator and evaluate their success. Successful plans for diverse tasks form a training set for supervised finetuning (SFT) of a smaller-scale LLM, which learns to improve manipulation abilities using no human supervision, see Fig. 1.

The BLAZER training uses privileged information in the form of object locations, orientations, and dimensions provided by the simulator. To deploy BLAZER in the real world, we design a vision pipeline using Molmo [17] and M2T2 [18] for object state estimation. Notably, our LLaMA-8B model trained with BLAZER significantly outperforms its initial and larger teacher model LLaMA-70B used to generate training demonstrations. Moreover, despite being trained in simulation, we demonstrate off-the-shelf transfer of BLAZER to real-world manipulation tasks where LLaMA-8B (47.8%) significantly outperforms the success rate of LLaMA-70B (33.3%). We perform extensive experimental evaluation and demonstrate consistent improvements of BLAZER in both the real world and in simulation for a variety of tasks. BLAZER outperforms state-of-the-art zero-shot MALMM [16], it generalizes to tasks unseen during training and requires no manual demonstrations at any stage of the learning process.

In summary, our work makes the following contributions:

- We introduce BLAZER, the framework that bootstraps a generic LLM and enables self-improvement of zero-shot manipulation agents using automatically generated training demonstrations.
- Through extensive experiments and ablations, we demonstrate BLAZER to result in significant improvements for a range of manipulation tasks.
- We further demonstrate off-the-shelf transfer of simulator-trained BLAZER to real-world manipulation tasks while using no manual demonstrations in any part of the training process.

II. RELATED WORK

We discuss related work on policy learning, self-improving models and scaling training data for robotics below.

A. Language and Vision Foundational Models for Robotics

Visuomotor policies [19], [20], [21] trained on manually collected demonstrations have shown great success in robotic manipulation. However, such models trained from scratch lack generalization to new tasks and environments. The incorporation of vision-language models (VLMs) into end-to-end robotic control, in the form of Vision-Language Action Models [22], [2], [3], [4], has enhanced generalization and enabled emergent semantic reasoning. The success of these methods, however, still relies on the collection of large-scale human demonstrations, which is expensive. ManipLLM [23] explores VLM finetuning with chain-of-thought reasoning for robotic manipulation tasks. This approach, however, requires test-time adaptation to overcome sim-to-real gap. In contrast, our framework enables self-improvement using no

human supervision. Moreover, our LLM agents, combined with our vision pipeline, directly transfer to real-world tasks without additional training.

To overcome the need of large-scale training data and to facilitate generalization to new tasks, recent work explores LLMs and VLMs for zero-shot robotic manipulation. Code as Policies [11] and MALMM [16] deploy LLM for writing robot policy code, while Voxposer [24] and GenSim [8], [9] couple code generation capabilities of LLM with multimodal reasoning capabilities of VLM to generate 3D value map for trajectory estimation and to automatically create simulator tasks, respectively. AHA [25] demonstrates that VLMs can be used for detecting and adapting to failures. Our approach is closely related to MALMM [16] and Code as Policies [11]. While these methods remain dependent on careful prompt engineering and require much computation at test time, our approach enables self-improvement using automatically generated training data and can run using relatively small and efficient LLMs at inference time.

B. Self-improving models

Recent work improves the reasoning capabilities of LLMs by generating few-shot rationales [26], [12], [27], bootstrapping [13], and reinforcement learning [28], [29], [15]. Among these approaches, self-training [13] stands out as a particularly scalable and successful strategy. In this approach, general-purpose LLMs are first used to generate high-quality rationales that are later deployed to train either improved versions of original models or smaller models [30].

The idea of self-improvement has also been explored in robotics e.g., to learn low-level visuomotor policies from a large dataset of grasping attempts [31] or from hours-long object poking interactions [32]. More recent methods focus on correcting manipulation failures at test time by reasoning about past experience [33] or detecting misalignment between planned and executed actions [34]. Another work, ReFineVLA [35], augments manipulation datasets with action rationales using Gemini [36], thereby enabling VLAs to reason about their actions. Related to our approach, SC-VLA [37] enables self-correction of VLA models from successful task executions. This parallel work, however, is mostly focused on low-level pushing and pulling actions, and requires explicit failure detection. In contrast, our bootstrapping approach only relies on success verification at the task level and addresses a variety of complex tasks that require high-level reasoning.

C. Data Generation for Manipulation

Many efforts have been dedicated to scale up training data for manipulation tasks. Some works collect real-world grasping data [31], [6] by deploying random trials followed by verification as well as collection of human demonstrations. KALIE [38] curated human-annotated affordance data as an alternative to robot demonstrations, whereas PhysObjects [39] collected an object-centric dataset to train a VLM with the physical concepts of common household objects. Another approach, LLaRA [40], adopted behavior cloning

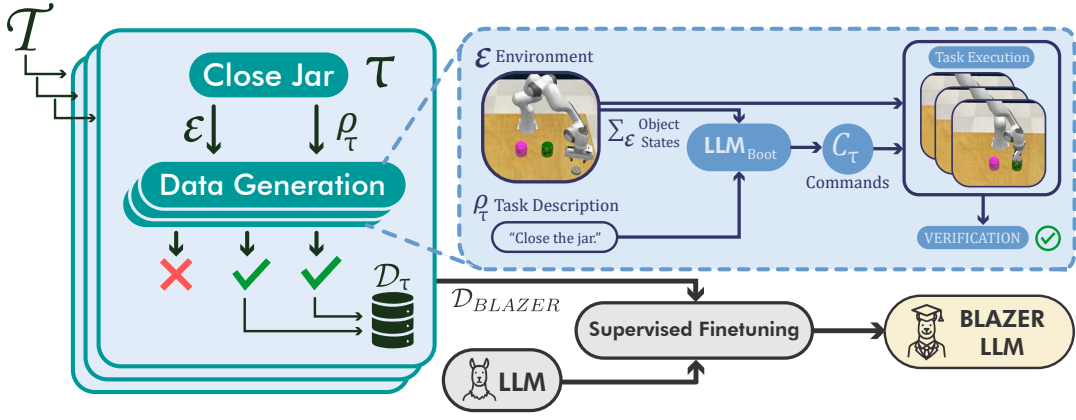


Fig. 2: **Overview of BLAZER.** Given a set of manipulation tasks $\tau \in \mathcal{T}$, we use LLM to automatically generate executable commands \mathcal{C}_τ for solving τ . The resulting solutions are automatically verified by executing \mathcal{C}_τ in a simulator and successful solutions are added to the task database \mathcal{D}_τ . Task databases for all training tasks \mathcal{T} are merged into $\mathcal{D}_{\text{BLAZER}}$ and are used for supervised finetuning of BLAZER LLM.

datasets for instruction tuning tailored for manipulation tasks. This approach relies on external demonstrations and uses manually curated templates to generate question-answer pairs.

Another popular direction is to collect demonstrations in simulation. Nevertheless, this approach is still challenging as it requires significant human efforts to create diverse simulation tasks and scenes to allow generalization of the learned policies. To address this issue, recent work [8], [9] has leveraged coding LLMs with multimodal reasoning capabilities to generate complex and realistic simulation tasks without human supervision. DemoGen [41] adapted a single human trajectory to novel object configurations using 3D point cloud editing, yielding synthetic demonstrations that improve manipulation policies. Unlike prior work, our work focuses on scaling up demonstrations for a given set of tasks, while preserving generalization capabilities of LLMs. We propose a framework where we can generate an arbitrary number of data in simulation, benefiting from the reasoning and coding capabilities of LLMs to synthesize verification data with no human intervention. We subsequently use the generated data for the training of our agent tailored for manipulation.

III. METHOD

Here, we introduce the BLAZER methodology for training specialized LLM agents for robotic manipulation. In short, we aim to finetune an existing lightweight LLM on synthetically-generated data, tuning it for the generation of manipulation-oriented robotics policies. An overview of our method is shown in Fig. 2. This section first introduces the formalization and the problem setup (Section III-A), and we further describe how the BLAZER training works in Section III-B. Since our training procedure only exploits automatic annotations generated by a simulator, we also introduce a vision-based pipeline for the deployment in the real world of the trained LLM agent (Section III-C).

A. Formalization and background

We assume a manipulation environment \mathcal{E} , where a robotic gripper \mathcal{G} interacts with objects to solve a task τ that we can automatically verify in simulation. To achieve this, \mathcal{G} needs to manipulate a set of K objects $\{o_1, o_2, \dots, o_K\}$ in \mathcal{E} , using a sequence of I control functions sampled from a primitives set $\mathcal{F} = \{\text{open_gripper}, \text{close_gripper}, \text{move_gripper}\}$. Primitives in \mathcal{F} are combined to define a policy composed of control commands \mathcal{C}_τ . Formally, this is written as follows:

$$\mathcal{C}_\tau = \{f_i\}_{i=1}^I, \forall f_i \in \mathcal{C}_\tau, f_i \in \mathcal{F}. \quad (1)$$

The primitives *open_gripper* and *close_gripper* can be executed without additional knowledge of the environment. In contrast, *move_gripper* requires as input the final desired gripper position and orientation. During execution, the gripper joints' positions are computed with inverse kinematics.

Following the open-loop single-agent setup introduced in MALMM [16], \mathcal{C}_τ can be obtained by prompting large-scale LLMs such as GPT-4 [42]. Logically, in this case the LLM must also generate all the code necessary for a correct execution. To achieve this, we define the current state of the environment as $\Sigma_\mathcal{E} = \{\mathbf{o}_i\}_{i=1}^K$, where each $\mathbf{o}_i = [\xi_i, \phi_i, l_i, w_i, h_i]$, $\forall i \in [1, K]$ encodes the 3D center position ξ_i of the object o_i , the orientation on the z-axis ϕ_i , and the metric dimensions l (length), w (width) and h (height). We also define ρ_τ as a textual description of τ , to enable LLM prompting. We then derive \mathcal{C}_τ by prompting the LLM with ρ_τ and a textual representation of $\Sigma_\mathcal{E}$. For more details, refer to [16]. In short, the control commands generation is expressed as follows:

$$\mathcal{C}_\tau = \text{LLM}(\rho_\tau, \Sigma_\mathcal{E}). \quad (2)$$

B. Bootstrapping manipulation agents with simulation

In many scenarios, naively prompting an LLM for \mathcal{C}_τ as in Eq. 2 results in planning or coding errors [16]. While MALMM mitigates the problem with an expensive multi-agent pipeline, our goal is instead to train a single LLM

agent LLM_{BLAZER}, with specific capabilities tuned ad hoc on manipulation tasks.

Our idea is to obtain LLM_{BLAZER} by finetuning a pre-trained lightweight language model without requiring human supervision, as we show in Figure 2. To do so, we use synthetic examples tailored for \mathcal{C}_τ generation, obtained within the interactions of an LLM with a manipulation-oriented simulator such as CoppeliaSim. We first define a set of T representative tasks $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_T\}$ that we use for data generation. Each task is processed by a language model LLM_{boot} to generate \mathcal{C}_τ as described in Section III-A. Importantly, since we operate in a simulated environment, we can automatically verify if \mathcal{C}_τ successfully solves τ , thanks to the automatic verification criteria in manipulation-oriented simulators. Hence, for given commands \mathcal{C}_τ and a task τ , we associate a verification operator V defined as follows:

$$V(\mathcal{C}_\tau, \tau) = \begin{cases} \checkmark & \text{if } \mathcal{C}_\tau \text{ is an acceptable solution for } \tau, \\ \times & \text{otherwise.} \end{cases} \quad (3)$$

Now, for each task τ we automatically collect a dataset \mathcal{D}_τ of N successful solutions provided by LLM_{boot}. While this would be extremely challenging with real demonstrations, in a simulated environment, we can easily randomize the initial state $\Sigma_\mathcal{E}$ to obtain an arbitrary number of object configurations for each task in \mathcal{T} , and automatically verify the correctness of proposed commands. We illustrate this process in Fig. 2. For any $\tau \in \mathcal{T}$,

$$\mathcal{D}_\tau = \{\text{LLM}_{\text{boot}}(\rho_\tau, \Sigma_\mathcal{E}^i)\}_{i=1}^N, \quad (4)$$

$$\text{if } V(\text{LLM}_{\text{boot}}(\rho_\tau, \Sigma_\mathcal{E}^i), \tau) = \checkmark.$$

In Eq. (4), we call $\Sigma_\mathcal{E}^i$ a random state configuration sampled in the simulated environment. Note that it is impossible to only generate $\Sigma_\mathcal{E}^i$ such that \mathcal{C}_τ is successful; hence, part of the generated commands resulting in unacceptable \mathcal{C}_τ will be discarded by our simulator-in-the-loop verification strategy. Finally, we train LLM_{BLAZER} using Supervised Finetuning (SFT) on a dataset $\mathcal{D}_{\text{BLAZER}}$ resulting from the aggregation of all \mathcal{D}_τ :

$$\text{LLM}_{\text{BLAZER}} \leftarrow \text{SFT}(\text{LLM}, \mathcal{D}_{\text{BLAZER}}), \quad (5)$$

$$\text{where } \mathcal{D}_{\text{BLAZER}} = \{\mathcal{D}_\tau^i\}_{i=1}^T$$

Note that our training uses a generic target LLM, and hence we do not impose the finetuning of LLM_{boot} as in self-refinement strategies [13]. This allows to benefit from larger LLMs for data generation, easing the generation of successful commands. Ultimately, our simulated training with verification allows filtering of any wrong solution, automatically curating a dataset for SFT with no human intervention.

C. Vision pipeline

While in simulation we can extract the ground truth space $\Sigma_\mathcal{E}$, in a real-world deployment, we must instead estimate its approximation $\tilde{\Sigma}_\mathcal{E}$ purely from visual data. Hence, to enable the deployment of LLM_{BLAZER} outside the simulator, we introduce a vision pipeline, based on existing foundation

models and *no training*. This allows us to mitigate the distribution shift that would occur if we trained perception components directly on a simulated environment [43].

We assume a multi-view setup that can use any number of calibrated RGB-D cameras. As a preliminary step, we prompt GPT-4o for a list of elements present in the scene, conditioned on a given task τ . Then, in all views, we first prompt Molmo [17] to extract the 2D coordinates of the center of each object o_i in a task τ . Employing Molmo for prompt interpretation enables contextual and spatial reasoning, going beyond simple semantics [17]. Next, we use the 2D center coordinates as a prompt for Segment Anything [44], obtaining a segmentation mask of the object in 2D for that view. By combining the segmentation mask with the RGB-D data, we derive a 3D bounding box for the object, which provides its estimated dimensions. Finally, we use M2T2 [18] to predict the object’s 3D center position and grasping orientation. We aggregate the outputs from multiple views with a simple median filter to remove outliers, obtaining the final $(\tilde{l}, \tilde{w}, \tilde{h})$ from the bounding box and $\tilde{\xi}_i, \tilde{\phi}_i$ from M2T2. We then construct $\tilde{\mathbf{o}} = \{\tilde{\xi}_i, \tilde{\phi}_i, \tilde{l}, \tilde{w}, \tilde{h}\}$. Repeating this for each object in the space gives $\tilde{\Sigma}_\mathcal{E} = \{\tilde{\mathbf{o}}_i\}_{i=1}^K$. We use $\tilde{\Sigma}_\mathcal{E}$ as input to LLM_{BLAZER} in real-world deployment.

IV. EXPERIMENTS

Our experiments are designed to assess the LLM agent trained using our BLAZER framework against existing vision and language foundation model-based baselines.

A. Experimental details

Tasks and environment. For comparison with baselines, we follow the MALMM setup and evaluate on nine simulated tasks from RL-Bench [45], illustrated in Fig. 3: *Basketball in Hoop* (BH), *Close Jar* (CJ), *Empty Container* (EC), *Insert in Peg* (IP), *Meat off the Grill* (MG), *Open Bottle* (OB), *Put Block* (PB), *Rubbish in Bin* (RB), and *Stack Blocks* (SB). Detailed task descriptions and success conditions are provided in Appendix-B. We train BLAZER by setting these tasks as \mathcal{T} . During testing, we randomize each test episode configuration, preventing overlaps with those used in training. We use CoppeliaSim and interface it with PyRep. In the real-world setup, we test generalization on 12 tasks on a tabletop using a 7-DOF Franka Emika Panda Research 3 robot equipped with a parallel jaw gripper. We use three Intel RealSense D435i RGB-D cameras to capture the frontal, right, and left views and the panda-py [46] library to control the robot arm.

Implementation Details. For our experiments, we choose LLaMA-3.1-8B (LLaMA-8B) as LLM_{BLAZER}, and LLaMA-3.3-70B (LLaMA-70B) as LLM_{boot}. The prompt for data bootstrapping is adopted from Single Agent (SA) setup of MALMM [16], and is included in Appendix-A. Both the LLMs, LLM_{BLAZER} and LLM_{boot} generate 3D waypoints for the gripper, while trajectories are computed and executed using a motion planner, which is a common approach in RL-Bench [45]. During SFT, the models are trained with a prompt completion loss, using $N = 2000$ examples per task.

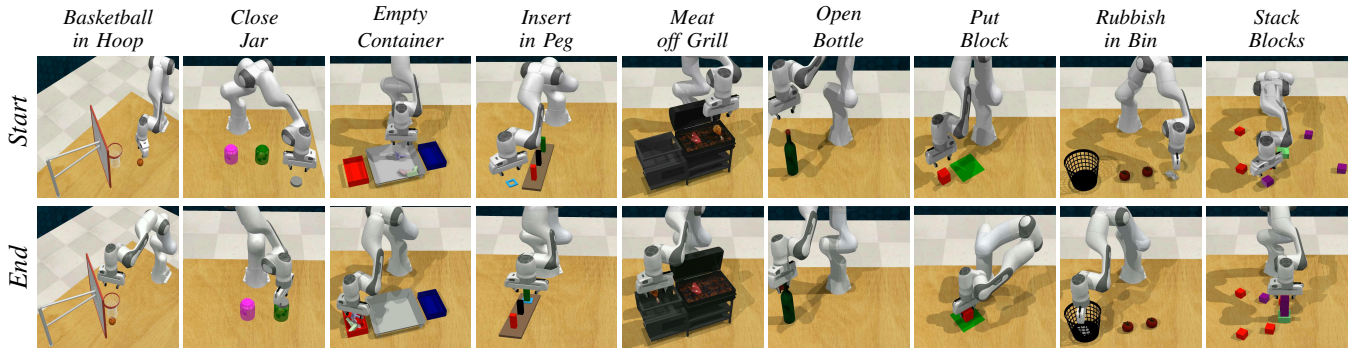


Fig. 3: **Tasks in simulation.** We consider 9 pick-and-place tasks from RLbench simulator [45]. For each task we display a starting condition (top) and the desired final state (bottom).

| Method | Basketball in Hoop | Close Jar | Empty Container | Insert in Peg | Meat off Grill | Open Bottle | Put Block | Rubbish in Bin | Stack Blocks | Average |
|---------------|-----------------------|--------------|--------------------|------------------|-------------------|----------------|--------------|-------------------|-----------------|-------------|
| CAP [11] | 0.0 | 0.0 | 0.0 | 8.0 | 0.0 | 0.0 | 76.0 | 0.0 | 0.0 | 9.3 |
| VoxPoser [24] | 20.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 36.0 | 64.0 | 32.0 | 16.9 |
| MALMM [16] | 88.0 | 84.0 | 64.0 | 68.0 | 92.0 | <u>96.0</u> | 100 | 80.0 | <u>56.0</u> | <u>80.9</u> |
| LLaMA-70B | 70.0 | 99.0 | <u>40.0</u> | <u>86.0</u> | 66.0 | <u>93.0</u> | 93.0 | 97.0 | 49.0 | 77.0 |
| LLaMA-8B | 53.0 | 16.0 | 1.0 | 15.0 | 16.0 | 57.0 | 60.0 | 10.0 | 0.0 | 25.3 |
| → w/ BLAZER | 95.0 | <u>88.0</u> | 39.0 | 97.0 | <u>76.0</u> | 98.0 | <u>98.0</u> | <u>95.0</u> | 63.0 | 83.2 |

TABLE I: **Comparison with zero-shot baselines.** We report the task success rate (%) for different methods applied to the nine manipulation tasks from RLbench [45]. With a small LLaMA-8B model finetuned with BLAZER, we are able to achieve the best performance. Note how LLaMA-8B with BLAZER outperforms considerably LLaMA-70B, that was used as LLM_{boot}. This implies that BLAZER can yields LLMs that outperform their teacher models on manipulation tasks. The table highlights the best-performing method for each task in **bold** and the second-best method is underlined.

We train for 5 epochs with an effective batch size of 24. We adopted parameter-efficient finetuning via LoRA with a rank of 64 and a scaling factor (α) of 16. The learning rate is $2e-5$, with a cosine learning rate scheduler applied during training.

Baselines. We compare BLAZER to other methods using no manual supervision: CAP [11], VoxPoser [24] and MALMM [16]. All baseline results are reported following MALMM [16] and use GPT-4 Turbo [42] as the underlying LLM model. Furthermore, we also test two zero-shot LLM baselines, which include LLaMA-70B and LLaMA-8B. To do so, we query each LLM with the prompt used for $\mathcal{D}_{\text{BLAZER}}$ generation, and directly apply the obtained \mathcal{C}_τ as policy. Note that in BLAZER we use LLaMA-70B as LLM_{boot} and LLaMA-8B as LLM_{BLAZER}, so those baselines serve as comparison with respect to the base version of the LLMs used in our framework.

B. Performance analysis

Comparison to baselines In Table I, we present the performance of our LLM_{BLAZER} model on 100 episodes for each task in \mathcal{T} , along with results of baselines introduced in Sec. IV-A. Here we test all models assuming the knowledge of ground truth states $\Sigma_\mathcal{E}$ provided by the simulator.

Our LLaMA-8B model trained with BLAZER *outperforms all baselines* on average (**83.2%** success). This shows that the bootstrapping of reasoning data for manipulation tasks

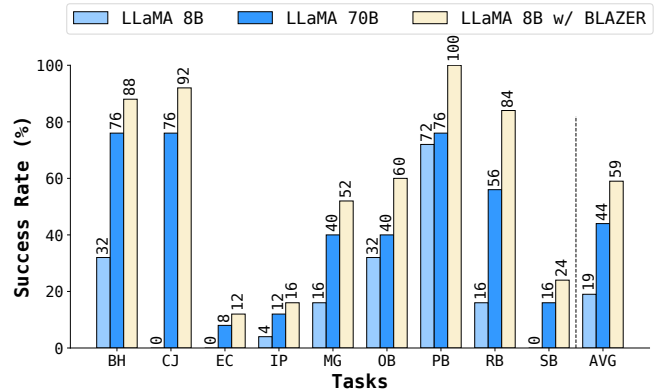
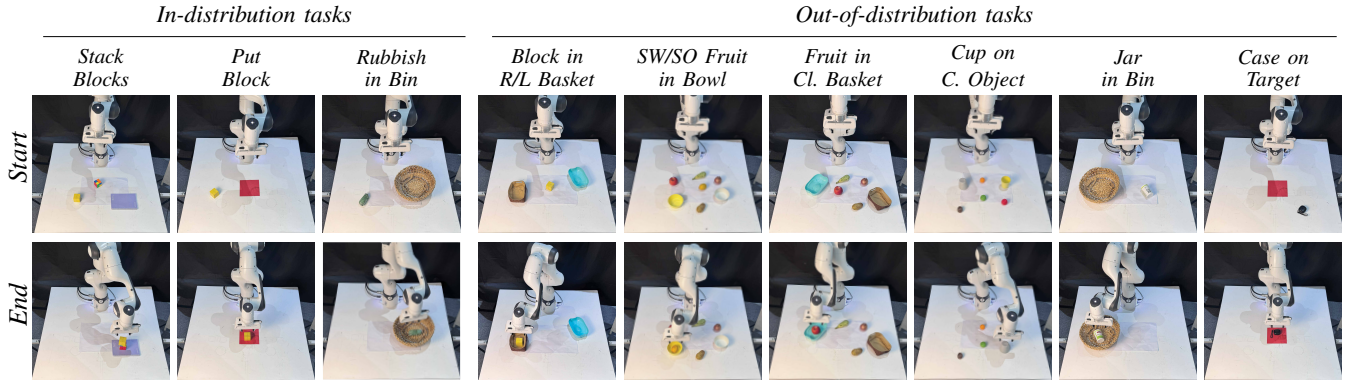


Fig. 4: **Tasks in simulation using visual observations.** We evaluate task success rate for LLaMA-70B, LLaMA-8B and LLaMA-8B w/ BLAZER in simulation using our vision pipeline that assuming no ground truth knowledge about object states. Consistently with results in Table I, BLAZER outperform other methods.

and the training of LLMs leads to stronger agents for robotic manipulation, proving the effectiveness of BLAZER. Importantly, our LLaMA-8B model with BLAZER surpasses considerably (+6.2% on average) LLaMA-70B, which we used as LLM_{boot}, using only a fraction of the parameters. Moreover, it substantially increases the capabilities of the



(a) Real world tasks visualization

| | In-distribution tasks | | | Out-of-distribution tasks | | | | | | |
|--------------------|-----------------------|-------------------|-------------------|---------------------------|-------------------|---------------------|-------------------|-------------------|-------------------|--------------|
| | Stack Blocks | Put Block | Rubbish in Bin | Block in R/L Basket | SW/SO Fruit in B. | Fruit in Cl. Basket | Cup on C. Object | Jar in Bin | Case on Target | Average |
| LLaMA-70B | 0.3 (3/10) | 0.4 (4/10) | 0.2 (2/10) | 0.6 (6/10) | 0.4 (4/10) | 0.5 (5/10) | 0.3 (3/10) | 0.3 (3/10) | 0.1 (1/10) | 0.333 |
| LLaMA-8B w/ BLAZER | 0.4 (4/10) | 0.6 (6/10) | 0.4 (4/10) | 0.5 (5/10) | 0.4 (4/10) | 0.7 (7/10) | 0.5 (5/10) | 0.4 (4/10) | 0.4 (4/10) | 0.478 |

(b) Quantitative evaluation.

Fig. 5: **Real world results.** We compare LLaMA-8B with BLAZER against LLaMA-70B on real-world tasks depicted in (a). From quantitative results in (b), we outperform the baseline, both on In-distribution tasks (similar to \mathcal{T}) and Out-of-distribution tasks, showcasing the generalization capability of BLAZER.

base LLaMA-8B (+58.4%). Even in long-term tasks such as *Stack Blocks* and *Empty Container*, LLaMA-8B trained with BLAZER outperforms LLaMA-70B in *Stack Blocks* by 14% and is almost on par with it on the *Empty Container* task. We notice that LLaMA-70B shows remarkable results with our detailed prompt, outperforming Code-as-Policy and VoxPoser, and proving the strength of our baseline. The MALMM multi-agent framework still surpasses LLaMA-70B due to its multistep failure detection and correction approach, although it still falls short of our agent.

Interestingly, we also note that zero-shot usage of LLaMA-8B results in unsatisfactory performance (avg. **25.3%** success). In particular, for *Stack Blocks* and *Empty Container*, LLaMA-8B often fails to generate valid control commands, even for a single episode in *Stack Blocks*. In contrast, LLaMA-70B successfully handles most tasks. This result is consistent with recent studies on long-context generation in large language models [47], [48]. Ultimately, the ability of LLaMA-70B to provide more successful examples for $\mathcal{D}_{\text{BLAZER}}$ justifies its usage as LLM_{boot} .

Perception impact. We now evaluate the robustness of BLAZER to a more realistic setup with visual observations replacing ground truth object states. We employ our vision-based pipeline (Section III-C) to process input views for simulated tasks in Table I, and obtain $\tilde{\Sigma}_{\mathcal{E}}$, which we use as input for LLMs. For fair comparison, we equip LLaMA-70B and the base LLaMA-8B as well as BLAZER with the same vision pipeline and report results in Figure 4. We observe that even with the noisy state estimation of our vision pipeline, LLaMA-8B trained with BLAZER still outperforms alternatives. In particular, the gain with respect to LLaMA-70B (+15%) is even higher than when providing the ground truth

$\Sigma_{\mathcal{E}}$ (+6.2%, see Table I). This shows the robustness to noise in the policies generated with LLMs trained with BLAZER.

C. Robot experiments

We next deploy BLAZER in the real robot setup and compare its performance to LLaMA-70B baseline while using the same vision pipeline for both methods.

Generalization capabilities. We use 9 additional tasks, shown in Figure 5a, to assess the transferability of BLAZER to real manipulation scenarios. We first consider *In-distribution* tasks that resemble the *Stack Blocks*, *Put Block*, and *Rubbish in Bin* tasks in Figure 3. We use these tasks to quantify the transfer of tasks in \mathcal{T} in real-world deployment. We also propose 6 new *Out-of-distribution* tasks, different from those in \mathcal{T} : *Block in Right/Left Basket*, *Sweet/Sour Fruit in Bowl*, *Fruit in Closest Basket*, *Cup on Colored Object*, *Jar in Bin*, and *Case on Target*. By testing these tasks, we aim to demonstrate the ability of BLAZER to solve tasks *beyond* those in \mathcal{T} . We average results over 10 episodes for each task, replacing generic attributes in the task description (e.g. “colored”, “sweet/sour”) with precise values (e.g. “red”, “sour”), randomized for each episode. From the results in Table 5b, we show that *our LLM agent trained with BLAZER still outperforms the baseline*, proving that the capabilities of tasks in \mathcal{T} transfer successfully to the real world. In particular, for *in-distribution* tasks, we notice a higher *Stack Blocks* performance, that was suboptimal in Figure 4. We speculate that the performance in simulation could have been influenced mainly by the presence of distractor elements (see Figure 3). For out-of-distribution tasks, we beat LLaMA-70B on 5 tasks out of 6, ultimately proving that the benefits of BLAZER training transfer to new task in real world settings.

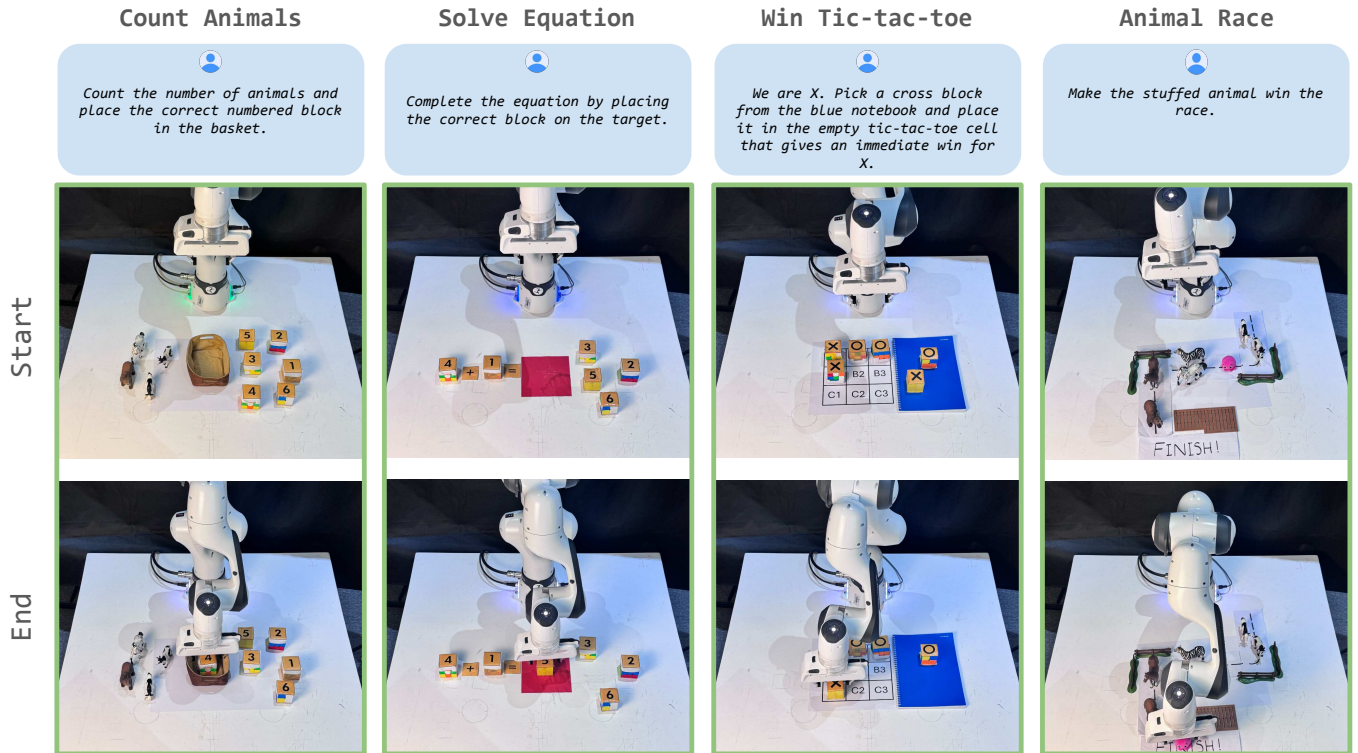


Fig. 6: **Real-world results on reasoning tasks.** We show examples of four tasks that illustrate high-level reasoning and planning capabilities of BLAZER and demonstrate its generalization to new tasks. We also provide the prompt to BLAZER in the blue box.

Qualitative examples. To further evaluate the generalization abilities of BLAZER, we present it with four tasks that require high-level reasoning. As in previous experiments in Fig. 5b, we use BLAZER trained in simulation on \mathcal{T} tasks together with the vision pipeline described in Section III-C. Specifically, we employ numbered blocks to test math capabilities in the *Count Animals* and *Solve Equation* tasks, game strategic planning in *Win Tic-tac-toe*, and contextual awareness in *Animal Race*. We provide qualitative results of BLAZER for these tasks in Figure 6. As can be seen, BLAZER can successfully solve tasks that require high-level reasoning while being substantially different from the training tasks \mathcal{T} . Videos illustrating execution of these tasks can be found on the project website [1].

D. Ablation studies

In this section, we propose ablation studies. First, we study the impact of changing the number of per-task training samples N that we use for training $\text{LLM}_{\text{BLAZER}}$. Second, we analyze if smaller models are compatible with BLAZER.

Training dataset size. In BLAZER, we can generate arbitrarily large datasets of training samples. To understand the impact of the data scale, we train LLaMA-8B with BLAZER on 4 different datasets generated automatically, containing 500, 1000, 2000, and 4000 samples per task (N). We report in Figure 7 (left) the average results across the 9 tasks in simulation that we use in Table I. From our results,

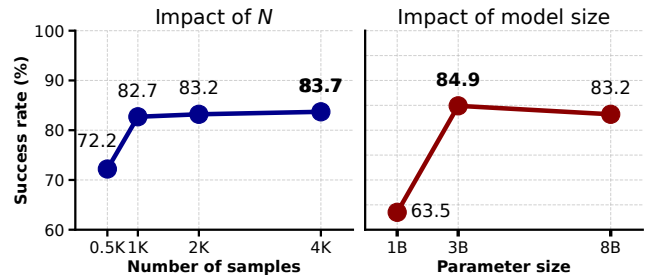


Fig. 7: **Ablation studies.** On the left, we study the impact of the number of samples N . We notice diminishing returns for our trained model and stop at 4K generated samples. On the right, we study the impact of the parameters of the model, proving that also smaller models can benefit from BLAZER.

it is evident that the size of N significantly impacts the efficacy of the trained $\text{LLM}_{\text{BLAZER}}$, with a substantial loss of performance when $N = 500$ (success rate 72.2%). However, we also noticed diminishing returns at the increase of N . In particular, we noticed that the performance doubling N from 2000 to 4000 is only +0.5%, so we have chosen $N = 2000$ in the paper to shorten training times. In conclusion, we advocate that although BLAZER is effective in exploiting the synthetically generated data, more strategies may be needed to increase accuracy beyond saturation.

Model size. Within BLAZER, we finetune a small model (LLaMA-8B) and use a larger one for data generation (LLaMA-70B). We aim to understand whether even smaller models can be trained with BLAZER, to enable applications on edge devices with low computational resources. To do so, we trained LLaMA-3.2 1B and LLaMA-3.2 3B [49] with BLAZER and compared it with our finetuned LLaMA-8B used for all other experiments. All models use data generated by LLaMA-70B as LLM_{boot} . We present results in Figure 7 (right). The usage of the 3B and 1B models still results in remarkable performance. Interestingly, LLaMA-3.2 3B achieves **84.9%** as average success rate, even marginally higher than LLaMA-8B (**83.2%**). Please note that LLaMA-3.2 is a different release from LLaMA-8B (3.1), therefore, we attribute the higher performance to the superior data quality used for the 3.2 LLaMA release [49]. This shows that even a compact model used as $\text{LLM}_{\text{BLAZER}}$ can result in competitive performance with state-of-the-art zero-shot manipulation methods based on LLMs. Conversely, the 1B model yields lower successes (63.5%), but still beats some baselines in Table I with a very limited number of parameters. This shows further flexibility of BLAZER in model size for applications with significant computational constraints.

V. CONCLUSIONS

In this paper, we introduced BLAZER, a method for finetuning standard LLMs to obtain specialized agents for robotic manipulation. We demonstrated the efficacy of BLAZER in both simulated and real environments and evaluated its generalization performance on different tasks beyond those used for training. We believe that our work will encourage further research on the usage of pretrained LLMs for robotics-oriented tasks. While BLAZER currently exploits only positive demonstrations for supervised finetuning, future work can benefit from both positive and negative samples to learn more accurate manipulation models. Indeed, unsuccessful episodes generated by our method in simulation could be used for more advanced post-training techniques such as Direct Preference Optimization (DPO) [50], that are able to exploit preference pairs including negative examples.

REFERENCES

- [1] “BLAZER project webpage,” <https://blazer-llm-agent.github.io/>. 1, 7
- [2] M. J. Kim *et al.*, “OpenVLA: An open-source vision-language-action model,” *arXiv*, 2024. 1, 2
- [3] P. Intelligence *et al.*, “ $\pi 0.5$: a vision-language-action model with open-world generalization,” *arXiv*, 2025. 1, 2
- [4] T. L. Team *et al.*, “A careful examination of large behavior models for multitask dexterous manipulation,” *arXiv*, 2025. 1, 2
- [5] J. Lee, J. Duan, H. Fang *et al.*, “MolmoAct: Action reasoning models that can reason in space,” *arXiv*, 2025. 1
- [6] A. Padalkar *et al.*, “Open X-Embodiment: Robotic learning datasets and rt-x models,” *arXiv*, 2023. 1, 2
- [7] R. McCarthy, D. C. Tan, D. Schmidt, F. Acero, N. Herr, Y. Du, T. G. Thuruthel, and Z. Li, “Towards generalist robot learning from internet video: A survey,” *JAIR*, 2024. 1
- [8] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang, “GenSim: Generating robotic simulation tasks via large language models,” *arXiv*, 2023. 1, 2, 3
- [9] P. Hua, M. Liu, A. Macaluso, Y. Lin, W. Zhang, H. Xu, and L. Wang, “GenSim2: Scaling robot data generation with multi-modal and reasoning llms,” in *CoRL*, 2024. 1, 2, 3
- [10] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, Z. Erickson, D. Held, and C. Gan, “RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation,” *ICML*, 2024. 1
- [11] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. R. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” *ICRA*, 2022. 1, 2, 5
- [12] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. Chi, F. Xia, Q. Le, and D. Zhou, “Chain of thought prompting elicits reasoning in large language models,” *NeurIPS*, 2022. 1, 2
- [13] E. Zelikman, Y. Wu, and N. D. Goodman, “Star: Bootstrapping reasoning with reasoning,” in *NeurIPS*, 2022. 1, 2, 4
- [14] H. Lin, Z. Sun, Y. Yang, and S. Welleck, “Lean-STaR: Learning to interleave thinking and proving,” *ICLR*, 2025. 1
- [15] S. R. Motwani, C. Smith, R. J. Das, M. Rybchuk, P. Torr, I. Laptev, F. Pizzati, R. Clark, and C. S. de Witt, “MALT: Improving reasoning with multi-agent llm training,” *COLM*, 2025. 1, 2
- [16] H. Singh, R. J. Das, M. Han, P. Nakov, and I. Laptev, “MALMM: Multi-agent large language models for zero-shot robotics manipulation,” *IROS*, 2025. 2, 3, 4, 5
- [17] J. Lee, J. Duan, H. Fang, Y. Deng, S. Liu, B. Li, B. Fang, J. Zhang, Y. R. Wang, S. Lee, W. Han, W. Pumacay, A. Wu, R. Hendrix, K. Farley, E. VanderBilt, A. Farhadi, D. Fox, and R. Krishna, “Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models,” *CVPR*, 2024. 2, 4
- [18] W. Yuan, A. Murali, A. Mousavian, and D. Fox, “M2T2: Multi-task masked transformer for object-centric pick and place,” in *CoRL*, 2023. 2, 4
- [19] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *JMLR*, 2015. 2
- [20] P. R. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. S. Wong, J. Lee, I. Mordatch, and J. Tompson, “Implicit behavioral cloning,” in *CoRL*, 2022. 2
- [21] P.-L. Guhur, S. Chen, R. G. Pinel, M. Tapaswi, I. Laptev, and C. Schmid, “Instruction-driven history-aware policies for robotic manipulations,” in *CoRL*, 2023. 2
- [22] A. Brohan *et al.*, “RT-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv*, 2023. 2
- [23] X. Li *et al.*, “ManipLLM: Embodied multimodal large language model for object-centric robotic manipulation,” *CVPR*, 2023. 2
- [24] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, “VoxPoser: Composible 3d value maps for robotic manipulation with language models,” *CoRL*, 2023. 2, 5
- [25] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, A. Mandlekar, and Y. Guo, “AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation,” *ICLR*, 2025. 2
- [26] M. Nye, A. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena, “Show your work: Scratchpads for intermediate computation with language models,” *arXiv*, 2021. 2
- [27] W. Chen, X. Ma, X. Wang, and W. W. Cohen, “Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks,” *TMLR*, 2022. 2
- [28] DeepSeek-AI, D. Guo *et al.*, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv*, 2025. 2
- [29] A. R. Setlur, S. Garg, X. Geng, N. Garg, V. Smith, and A. Kumar, “R1 on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold,” *NeurIPS*, 2024. 2
- [30] N. Ho, L. Schmid, and S.-Y. Yun, “Large language models are reasoning teachers,” in *ACL*, 2022. 2
- [31] L. Pinto and A. K. Gupta, “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours,” *ICRA*, 2015. 2
- [32] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, “Learning to poke: Experiential learning of intuitive physics,” *NeurIPS*, 2016. 2
- [33] Z. Liu, A. Bahety, and S. Song, “Reflect: Summarizing robot experiences for failure explanation and correction,” *CoRL*, 2023. 2
- [34] Y. Guo, Y.-J. Wang, L. Zha, Z. Jiang, and J. Chen, “DoReMi: Grounding language model by detecting and recovering from plan-execution misalignment,” in *IROS*, 2024. 2
- [35] T. V. Vo, T. Q. Nguyen, K. Nguyen, D. H. M. Nguyen, and M. N. Vu, “ReFineVLA: Reasoning-aware teacher-guided transfer fine-tuning,” *arxiv*, 2025. 2

- [36] G. Team, “Gemini: A family of highly capable multimodal models,” 2025. [2](#)
- [37] C. Li, J. Liu, G. Wang, X. Li, S. Chen, L. Heng, C. Xiong, J. Ge, R. Zhang, K. Zhou, and S. Zhang, “A self-correcting vision-language-action model for fast and slow system manipulation,” *arxiv*, 2024. [2](#)
- [38] G. Tang, S. Rajkumar, Y. Zhou, H. R. Walke, S. Levine, and K. Fang, “Kalie: Fine-tuning vision-language models for open-world manipulation without robot data,” *ICRA*, 2024. [2](#)
- [39] J. Gao, B. Sarkar, F. Xia, T. Xiao, J. Wu, B. Ichter, A. Majumdar, and D. Sadigh, “Physically grounded vision-language models for robotic manipulation,” *ICRA*, 2023. [2](#)
- [40] X. Li, C. Mata, J. S. Park, K. Kahatapitiya, Y. S. Jang, J. Shang, K. Ranasinghe, R. Burgert, M. Cai, Y. J. Lee, and M. S. Ryoo, “LLaRA: Supercharging robot learning data for vision-language policy,” *ICLR*, 2025. [2](#)
- [41] Z. Xue, S. Deng, Z. Chen, Y. Wang, Z. Yuan, and H. Xu, “Demogen: Synthetic demonstration generation for data-efficient visuomotor policy learning,” *RSS*, 2025. [3](#)
- [42] OpenAI, “GPT-4 technical report,” 2023. [3](#), [5](#)
- [43] A. Torralba and A. A. Efros, “Unbiased look at dataset bias,” in *CVPR*, 2011. [4](#)
- [44] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. B. Girshick, “Segment anything,” in *ICCV*, 2023. [4](#)
- [45] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, “Rlbench: The robot learning benchmark & learning environment,” *RA-L*, 2019. [4](#), [5](#)
- [46] J. Elsner, “Taming the panda with python: A powerful duo for seamless robotics programming and integration,” *SoftwareX*, 2023. [4](#)
- [47] C. An, J. Zhang, M. Zhong, L. Li, S. Gong, Y. Luo, J. Xu, and L. Kong, “Why does the effective context length of llms fall short?” *ICLR*, 2025. [6](#)
- [48] C.-P. Hsieh, S. Sun, S. Krizan, S. Acharya, D. Rekesch, F. Jia, and B. Ginsburg, “Ruler: What’s the real context size of your long-context language models?” *COLM*, 2024. [6](#)
- [49] Meta AI, “Llama 3.2: Revolutionizing edge ai and vision with open, customizable models,” 2024. [8](#)
- [50] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” *NeurIPS*, 2023. [8](#)

APPENDIX

This appendix provides additional details about our BLAZER framework, including LLM Agent Prompt in Appendix-A, and RL Bench tasks in Appendix-B.

A. Prompts

The prompt used by the LLM Agent is illustrated in Figure 8. It contains four placeholders: *[INSERT TASK]* representing the task instruction, *[INSERT EE POSITION]* denoting the initial position of the end effector, *[INSERT EE ORIENTATION]* specifying its initial orientation, and *[INSERT CURRENT STATE ENVIRONMENT]* indicating the current state or observation of the environment.

LLM Agent Prompt

You are a sentient AI specialized in generating a sequence of steps and Python code for the robot arm end-effector to complete a given task. The end effector is in a environment and informations about the objects in the environment including the end-effector are provided in terms of their positions and orientations. You must remember that this conversation is a monologue, and that you are in control. I am not able to assist you with any questions, and you must output the plan and code yourself by making use of the common sense, general knowledge, and available information.

PLANNER:

ENVIRONMENT SET-UP:

The 3D coordinate system of the environment is as follows:

1. The x-axis is in the horizontal direction, increasing to the right.
2. The y-axis is in the depth direction, increasing away from you.
3. The z-axis is in the vertical direction, increasing upwards.
4. Workspace is in the table positioned at a z-level of 0.7519986. The workspace ranges from x: -0.075 to 0.575 and y: -0.455 to 0.455.

CONSTRAINTS:

1. **Orientation of the end effector** will always be **z orientation of object to grasp or destination to place**.
2. Negative rotation values represent clockwise rotation, and positive rotation values represent anticlockwise rotation. The rotation values should be in radians.
3. The **<safe distance>** in the z direction is 0.1 units.
4. The **<release distance>** in the z direction is 0.02 units above the top surface of the destination area or object, please estimate this distance.
5. **Remember you can only grasp the object from its *CENTER*. Not from any other position. So to grasp the object, end effector has to be lowered down to center of object**

COLLISION AVOIDANCE:

If there are multiple objects in the environment:

1. Make sure to consider the widths, lengths, and heights of other objects so that robot arm end effector does not collide with other objects or table.
2. This information may help to generate additional trajectories and add specific waypoints (calculated from the given objects' information) to avoid collision with other objects and the table.

COLLISION FREE OBJECT INTERACTION RULES:

1. Position the gripper **<safe distance>** above the target object.
2. Move to the ***CENTER*** of the target object. If ***CENTER*** position is (x,y,z) then grasping position will also be (x,y,z) Do not add any height margin.
3. Grasp the target object.
4. Raise the gripper **<safe distance>** above the target object.
5. Move to **<safe distance>** above the destination area.
6. Lower the gripper to the destination area.
7. Release the object (drop) at **<release distance>** **(*(0.02 units))** above the top surface of destination area.
8. Raise the gripper **<safe distance>** above the destination area.

PLANNING:

1. Describe the relative positions of all objects in the environment, including their spatial relationships, alignments, and groupings.
2. Provide a detailed, step-by-step plan for the given task.
3. Generate the all code corresponding to each high level plan.

CODE GENERATOR:

AVAILABLE FUNCTIONS:

You are able to call any of the following Python functions, if required, as often as you want:

1. `execute_trajectory(position: list[float], orientation: float) -> None`: This function will execute the trajectory on the robot arm end-effector based on position and orientation, and will also not return anything. It takes list position of 3 elements and one float orientation value as input.
2. `open_gripper() -> None`: This function will open the gripper on the robot arm, and will also not return anything.
3. `close_gripper(object_name: str) -> None`: This function will close the gripper on the robot arm, and will also not return anything. It takes the name of the object as input.
4. `task_completed() -> None`: Call this function only when the task has been completed. This function will also not return anything.

CODE GENERATION:

When generating the code for the trajectory, do the following:

1. Mark code clearly with the ````python` and ````` tags.
2. When mentioning the functions, specify the required parameters and clearly define them in the same code block before passing it to code executor. For `execute_trajectory`, define the position and orientation lists prior to it and mention object name in `close_gripper(object_name)` from `<CURRENT ENVIRONMENT STATE>`.
3. Orientation parameter will always be z orientation of object to grasp or destination to place.
4. *Generate the code all in one go for all the steps;*

Use the robot arm end effector to "[INSERT TASK]" in the environment.

The robot arm end-effector is currently positioned at [INSERT EE POSITION], with the orientation [INSERT EE ORIENTATION], and the gripper is open.

Remember you can only grasp the objects from its *CENTER*. Not from any other position. So to grasp the object, end effector has to be lowered down to center of object

The positions and orientations of all objects in the environment as follows:

`<CURRENT ENVIRONMENT STATE>`:

"[INSERT CURRENT STATE ENVIRONMENT]"

Fig. 8: Prompt for the LLM Agent.

B. RLBench Tasks

We experimented with nine tasks from RLBench, which are listed in Table II along with the task instructions and success criteria.

TABLE II: Details of the RLBench tasks used for evaluation.

| Task Instruction | Details |
|---------------------------|--|
| <i>Basketball In Hoop</i> | <i>Task Description:</i> Put basketball in hoop. <i>Success Criteria:</i> Basketball passes through hoop. |
| <i>Close Jar</i> | <i>Task Description:</i> Close the colored jar with a lid. <i>Success Criteria:</i> Lid is on top of the colored jar. |
| <i>Empty Container</i> | <i>Task Description:</i> Pick all the objects from the large container and put them into the colored container. <i>Success Criteria:</i> All objects from the large container are now in the colored container. |
| <i>Insert In Peg</i> | <i>Task Description:</i> Insert the square ring into the colored peg. <i>Success Criteria:</i> The square ring is in the colored peg. |
| <i>Meat Off Grill</i> | <i>Task Description:</i> Pick the meat (chicken or steak) from the grill and place it into the designated area. <i>Success Criteria:</i> Meat is on the designated area. |
| <i>Open Bottle</i> | <i>Task Description:</i> Remove the cap of the wine bottle. <i>Success Criteria:</i> Cap of the wine bottle is removed. |
| <i>Put Block</i> | <i>Task Description:</i> Put the block in the target area. <i>Success Criteria:</i> The block is in the target area. |
| <i>Rubbish In Bin</i> | <i>Task Description:</i> Put the rubbish in the bin. <i>Success Criteria:</i> Rubbish is in the bin. |
| <i>Stack Blocks</i> | <i>Task Description:</i> Stack a specified number of colored blocks on the target block. <i>Success Criteria:</i> Specified number of blocks are stacked on top of the target block. |