ExploRLLM

Guiding Exploration in Reinforcement Learning with Large Language Models

Ma, Runyu; Luijkx, Jelle; Ajanovic, Zlatan; Kober, Jens

DOI
10.1109/ICRA55743.2025.11127622

Publication date
2025

Document Version
Final published version

Published in
Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2025

2025 IEEE International Conference on Robotics and Automation (ICRA)
May 19-23, 2025. Atlanta, USA

ExploRLLM: Guiding Exploration in Reinforcement Learning with Large Language Models

Runyu Ma*[1], Jelle Luijkx*[1], Zlatan Ajanović[2], and Jens Kober[1]

***Abstract*— In robot manipulation, Reinforcement Learning (RL) often suffers from low sample efficiency and uncertain convergence, especially in large observation and action spaces. Foundation Models (FMs) offer an alternative, demonstrating promise in zero-shot and few-shot settings. However, they can be unreliable due to limited physical and spatial understanding. We introduce ExploRLLM, a method that combines the strengths of both paradigms. In our approach, FMs improve RL convergence by generating policy code and efficient representations, while a residual RL agent compensates for the FMs' limited physical understanding. We show that ExploRLLM outperforms both policies derived from FMs and RL baselines in table-top manipulation tasks. Additionally, real-world experiments show that the policies exhibit promising zero-shot sim-to-real transfer. Supplementary material is available at https://explorllm.github.io.**

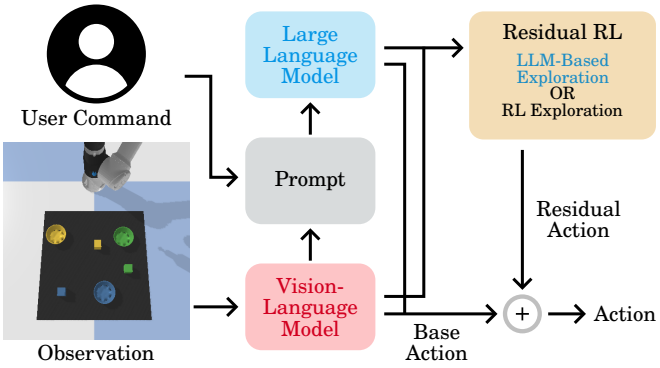


Fig. 1: Graphical overview of ExploRLLM.

I. Introduction

Foundation Models (FMs) [1], which refer to models trained on large-scale data, have shown great potential in robotics. In particular, language-based FMs, such as Large Language Models (LLMs) and Vision-Language Models (VLMs), are increasingly used in the field. Large Language Models, such as GPT-4 [2], can generate commonsense-aware reasoning in various scenarios. For instance, LLMs have demonstrated zero-shot planning capabilities [3], breaking down complex tasks into detailed step-by-step plans without additional training. When integrated with VLMs, LLMs leverage cross-domain knowledge for robot perception and planning in manipulation tasks [4]. This synergy allows for extracting environmental affordances and constraints, forming a foundation for subsequent robotic planning [5]. Despite the impressive results of FMs, unpredictable failures in LLM predictions can still lead to robotic errors, and LLMs generally do not learn from past experiences [6], [7].

On the other hand, Reinforcement Learning (RL) offers a powerful framework for learning decision-making and control policies through interaction with the environment [8]. However, RL struggles with the "curse of dimensionality," where large observation and action spaces slow down exploration and convergence. To address this, we propose combining FMs and RL by using FMs to guide the RL agent's exploration as depicted in Figure 1. While actions generated by FMs may be suboptimal or fail, they can highlight meaningful regions in the action space for exploration. Traditional RL exploration strategies (e.g., ϵ-greedy, Boltzmann exploration [9]) are stochastic, focusing on exploration-exploitation trade-offs, but lack mechanisms to incorporate prior knowledge for faster convergence. Instead, we use LLMs as few-shot planners, generating actions that serve as exploration steps in RL, increasing the likelihood of successful states and gathering more relevant state-action pairs for off-policy RL agents.

Our method, ExploRLLM, improves performance by compensating for FMs' sub-optimality and biases through RL, while FMs accelerate RL training by reducing observation spaces and guiding exploration. To summarize, our main contributions are the following.

1) We propose ExploRLLM, which employs an RL agent with a) residual action and observation spaces based on affordances identified by FMs and b) LLM-guided exploration.
2) We introduce a prompting method for LLM-based exploration using hierarchical language-model programs, leading to faster convergence.
3) We show that ExploRLLM outperforms policies derived solely from LLMs and VLMs and generalizes to unseen scenarios, tasks, and real-world settings without additional training.

II. Related Work

A. Foundation Models for Planning in Robotics

Researchers have shown that LLMs can exhibit reasoning capabilities and generate plans in zero-shot or few-shot settings [3], [10], which is crucial for high-level planning in robotics. These models facilitate task-level planning by integrating environmental groundings, such as affordance value scores [11] or feedback [12], with their language groundings. Furthermore, LLMs can generate robot-centric

* Equal Contribution. [1] Cognitive Robotics, Delft University of Technology, The Netherlands (e-mail: {j.d.luijkx, j.kober}@tudelft.nl). [2] RWTH Aachen University, Germany (e-mail: zlatan.ajanovic@ml.rwth-aachen.de).

2025 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3315-4139-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICRA55743.2025.11127622

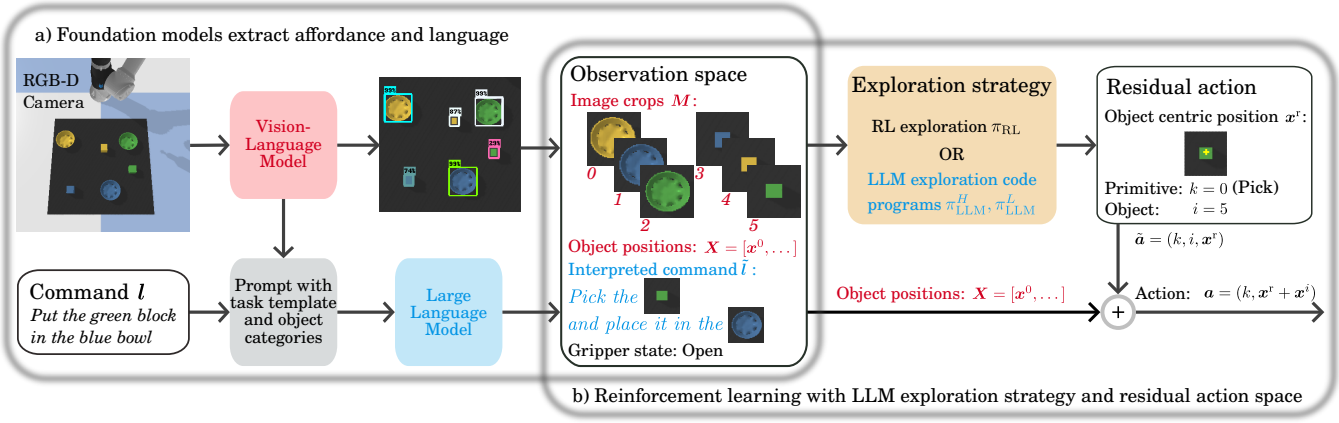


Fig. 2: Implementation structure of ExploRLLM for tabletop manipulation, combining the strengths of RL and FMs.

code programs as representations for both task-level [13] and skill-level planning [14]. Additionally, VLMs are increasingly integrated into robotics as a perception module of environmental context. The integration of knowledge from LLMs and VLMs can facilitate the creation of perception-planning pipelines [4] and the construction of 3D value maps for zero-shot planning frameworks [5]. However, due to real-world uncertainty, directly applying VLMs and LLMs to zero-shot tasks may not guarantee success or safety. Therefore, in our research, we treat these actions as exploratory behaviors within an RL framework.

B. Foundation Models and Reinforcement Learning

Incorporating FMs into RL frameworks has notably improved RL's effectiveness. In [15], the authors have implemented LLMs as proxy reward functions, demonstrating their utility in RL. In the context of RL for robotics, LLMs are also capable of generating reward signals for robot actions by connecting commonsense reasoning with low-level actions [16], self-refinement [17] and evolutionary optimization over reward code to enable complex tasks such as dexterous manipulation [18]. Regarding exploration, authors in [19] reward RL agents toward human-meaningful intermediate behaviors by prompting an LLM. LLMs are also utilized as an intrinsic reward generator to guide exploration for long horizon manipulation tasks [20]. Contrary to these studies, our approach employs LLM-generated code policies as exploratory actions rather than focusing on reward shaping. Simultaneously with our study, [21] introduced a method for improving the sample efficiency of reinforcement learning with LLM-generated rule-based controllers. In [21], the RL policy is regularized towards replay data generated with the LLM policies. Our method instead uses LLM-generated policies for exploratory actions and does not promote the RL agent to be close to the LLM-generated policies.

III. Problem Formulation

In this study, we focus on language-conditioned tabletop manipulation tasks and a detailed overview of the method is shown in Figure 2. Each manipulation task begins at timestep $t = 0$ with a linguistically described goal, denoted by $\boldsymbol{l}_t$. The agent receives an observation $\boldsymbol{o}_t$, consisting of an overhead RGB-D image and the state of the end-effector. Similar to existing methods (e.g., Transporter [22]), the action space involves a pick and a place primitive, denoted as $\{\mathcal{P}_{\text{pick}}, \mathcal{P}_{\text{place}}\}$, with each action parameterized by pick and place positions in a top-down view. We simplify this to a single motion primitive—either pick or place. This simplification makes the RL problem more tractable by eliminating the need to learn a feature representation for each primitive individually. The pick or place action is defined as a tuple containing the primitive index k (0 for pick, 1 for place) and a top-down view position, expressed as $\boldsymbol{x}$, i.e., $\boldsymbol{a}_t = (k_t, \boldsymbol{x}_t)$. At each time step, the agent receives a reward r_t consisting of a dense reward component r_t^d and a sparse reward r_t^s.

IV. Framework: ExploRLLM

A. Observation and Action Spaces

Our method leverages the strengths of LLMs and VLMs to reduce the observation space used for the RL framework. First of all, the LLM reformulates user-provided language commands into predefined templates and highlights the objects within these templates to form an interpreted command vector $\tilde{\boldsymbol{l}}_t$. An example is shown in Figure 2a, where *"Put the green block in the blue bowl"* is interpreted into the template *"Pick the [pick_object] and place it in the [place_object]"*. It is important to note that, within a given task setting, the number and category of objects do not change. Utilizing VLMs as open-vocabulary object detectors, our system identifies and encloses objects relevant to the task within bounding boxes from the image, represented by their locations $\boldsymbol{X}_t = [\boldsymbol{x}_t^0, \boldsymbol{x}_t^1, ...]$. RGB-D visual inputs are segmented into crops based on bounding box positions, denoted as $\boldsymbol{M}_t = [\boldsymbol{m}_t^0, \boldsymbol{m}_t^1, ...]$. This method improves the system's robustness to detection-inaccuracies and varying object shapes. The interpreted commands $\tilde{\boldsymbol{l}}_t$, the positional data $\boldsymbol{X}_t$ and the image patches $\boldsymbol{M}_t$ are then integrated into the reformulated RL observation $\boldsymbol{s}_t$ together with the robot gripper state (open/closed).

Algorithm 1: Exploration strategy π_{EXP}

Input: state $\boldsymbol{s}_t$, high-level LLM policy π^H_{LLM}, low-level LLM policy π^L_{LLM}, RL policy π_{RL}
Output: action $\tilde{\boldsymbol{a}}_t$
Parameter: threshold ϵ
$j \sim U_{[0,1)}$ `// Uniform sampling`
if $j \leq \epsilon$ **then**
  $\boldsymbol{a}^H_t = (k_t, i_t) \leftarrow \pi^H_{\text{LLM}}(\boldsymbol{s}_t)$ `// High-level`
  $\boldsymbol{x}^{\text{r}}_t \leftarrow \pi^L_{\text{LLM}}(\boldsymbol{s}_t, \boldsymbol{a}^H_t)$ `// Low-level`
  $\tilde{\boldsymbol{a}}_t = (k_t, i_t, \boldsymbol{x}^{\text{r}}_t)$
else
  $\tilde{\boldsymbol{a}}_t \leftarrow \pi_{\text{RL}}(\boldsymbol{s}_t)$ `// RL policy`
return $\tilde{\boldsymbol{a}}_t$

As the VLM already extracts each object's position $\boldsymbol{x}^i_t$, the action space is converted into an object-centric residual action space (see Figure 2b). The reformulated action space consists of a primitive index k, an object index i and a residual position $\boldsymbol{x}^{\text{r}}$, expressed as $\tilde{\boldsymbol{a}}_t = (k_t, i_t, \boldsymbol{x}^{\text{r}}_t)$. This residual position is then added to the position of object i, i.e., $\boldsymbol{x}_t = \boldsymbol{x}^i_t + \boldsymbol{x}^{\text{r}}_t$. This residual action allows the agent to pick or place objects at specific locations. This is, for example, needed when picking the letter O, and $\boldsymbol{x}^i_t$ denotes the center of the bounding box. In this case, the residual action $\boldsymbol{x}^{\text{r}}_t$ is needed to prevent picking the letter O at its empty center.

B. LLM-Based Exploration

Traditional deep RL algorithms (e.g., SAC [23], PPO [24]) do not inherently promote frequent visits to high-value states in high-dimensional state-action spaces, making vision-based tabletop manipulation tasks particularly challenging. In such cases, RL agents may struggle when successful outcomes are rare. Leveraging the planning capabilities of LLMs and the perception strengths of VLMs can help guide the exploration process more effectively by tapping into the rich prior knowledge within these FMs. The LLM-based exploration strategy, denoted as π_{EXP} in Algorithm 1, draws inspiration from the ϵ-greedy strategy. Specifically, during the rollout collection at each timestep, the off-policy RL agent employs the LLM-based exploration technique if a sampled random variable falls below the threshold ϵ. Otherwise, the action is selected according to the current RL agent's policy, π_{RL}, as detailed in Algorithm 1.

Inspired by Code-as-Policy (CaP) [14], our method employs the LLM to generate hierarchical language model programs, which are executed during the training phase as exploratory actions. The hierarchical language model programs include high-level π^H_{LLM} and low-level π^L_{LLM} policy code programs. A high-level plan primarily involves selecting robot action primitives and the objects to interact with based on the current state of the robot and the objects.

In contrast to high-level tasks, instructing low-level actions poses a more significant challenge because high-level states and actions are more accessible and can be represented as language. When dealing with low-level actions, the complexity of the state becomes considerably more intricate, particularly for image-based problems. Therefore, instead of a deterministic code policy, we instruct the LLM to produce a code policy π^L_{LLM} for generating an affordance map according to the input image. The low-level exploration behavior is derived from a stochastic policy that relies on the values within this affordance map. Although the code generated by LLMs lacks guaranteed feasibility and accuracy in robot environments, these models can generate potentially useful policy candidates, with the one exhibiting the highest success rate being selected as shown in Figure 3.

V. Implementation

A. RL Agent

We use the Soft Actor-Critic (SAC) algorithm with modifications in the collecting rollout phase, detailed in Algorithm 1. Other implementation aspects remain consistent with the standard SAC approach in stable-baselines3 [25]. We employ two convolutional layers to transform every image patch into a vector $\boldsymbol{\phi} \in \mathbb{R}^{n \times d}$, where n is the number of objects captured by VLM and d the dimension of each patch as encoded by the CNN. The vector is subsequently concatenated with the position, robot gripper state, and the extracted episodic language goal $\tilde{\boldsymbol{l}}$ to form a new vector $\boldsymbol{\phi}' \in \mathbb{R}^{n \times d'}$, where d' denotes the dimension of each patch's vector following encoding and concatenation. It then goes to a self-attention layer. The output features from this layer then go into a two-layer MLP. The structure mentioned above is consistently utilized across all actor and critic networks.

B. VLM Detection

Utilizing an open-vocabulary object detector ViLD [26], objects in the environment can be identified by given specific labels. However, implementing this model online during training is time-consuming, so ViLD is utilized solely in the evaluation phase. In the training phase, the ground truth in the simulation is used to determine the center positions of the bounding boxes. It is important to note that ViLD's position detection in real-world scenarios is not always flawless. To simulate this imperfection, Gaussian noise with a standard deviation equal to half the radius of the image crop is applied to the ground truth positions.

C. LLM Code Policy Generation

The policy code for executing high-level behavior is obtained using a few-shot prompt in GPT-4 [2]. It includes a list of available robot motion primitives to demonstrate the robot's actions. A custom API is also provided to aid the LLM in reasoning, such as determining whether an object is held in the robot's gripper or understanding the relationships between different objects. Following the approach demonstrated by [14], where LLMs have been shown capable of generating novel policy codes with example codes and commands, our prompt also includes examples. They are designed to guide the LLM in formulating plans

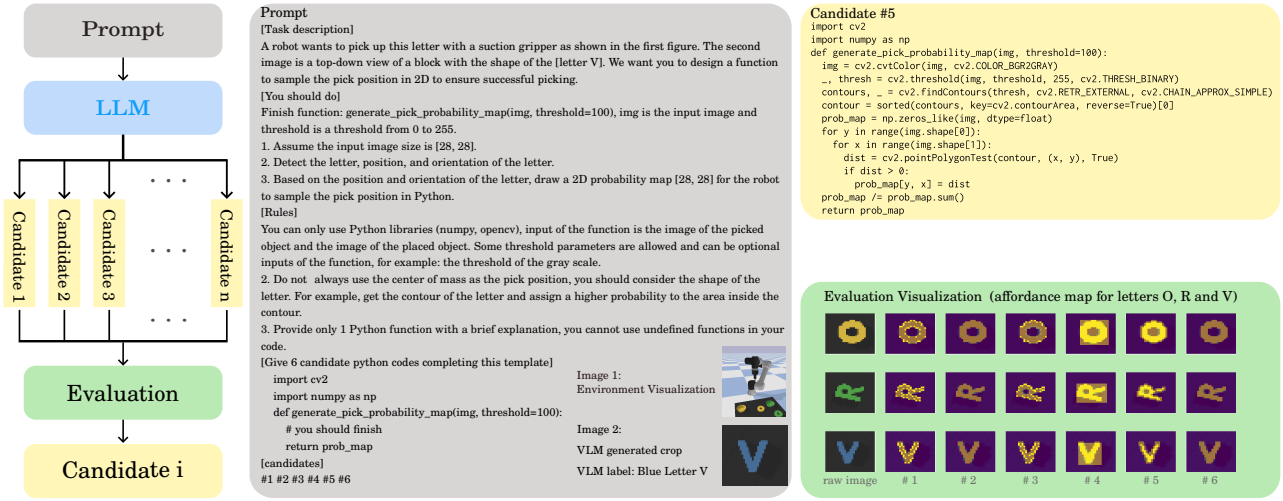


Fig. 3: Based on an exploration prompt, candidate policy code is generated. The exploration policy is selected after evaluation.

and conducting geometric reasoning for our specific task scenarios.

For low-level exploration actions, we employ GPT-4 with Vision [2], which generates code using prompts that combine example images with language descriptions, enriching the context with visual information, as shown in Figure 3. The provided example images include a depiction of the environmental setup featuring the robot, a simulated background, objects, and a specific example of image patches inside VLM bounding boxes. The prompt describes the requirements and guidelines, enabling generated code to create a probability affordance heatmap for the specified image patch, utilizing external libraries like OpenCV and NumPy. However, as indicated in Figure 3, there are instances where the generated affordance map may not be optimal. For example, the optimal pick position for the letter O should be at its rim, whereas the heatmap suggests the center.

To address sub-optimality, we use a stochastic policy based on the affordance map instead of a deterministic one that selects the point of highest affordance. Since RL improves through rewards from environmental interactions, sub-optimal exploration policies can be corrected via learning. This approach also allows for the generation of counter-examples during replay buffer collection.

VI. Experimental Setups

A. Simulation Setup

We evaluated the proposed method on a simulated tabletop pick-and-place task, as shown in Figure 2. Similar to [22] and [27], we use a UR5e, and the input observation is a top-down RGB-D image. Inspired by [27], we increased the task difficulty by replacing simple blocks with various objects, such as letters. We assess our method in two tasks: a short-horizon (SH) task, *"Pick the [pick_letter] and place it in the [place_color] bowl"*, and a long-horizon (LH) task, *"Put all letters in the bowl of the corresponding color"*, as shown in Figure 6a. In the SH task, each episode starts with three letters and three bowls randomly placed on the table, with pick-and-place actions generated from random language commands. The task is completed when the robot places the chosen letter in the specified bowl. In the LH task, all letters and bowls are randomly arranged, and the task is completed when each letter is placed in a bowl that matches its color.

B. Real-World Setup

We validated our approach on a Franka Panda robot equipped with a suction gripper and an RGB-D camera, as shown in Figure 6a, implementing our policy and code in the EAGERx [28] framework. Given the potential risks to hardware and the time-intensive nature of direct training, we completed training in simulation, with real-robot applications limited to evaluation. We used ViLD to identify bounding boxes based on object names. To simulate real-world conditions more accurately, we introduced noise to the bounding box center's position during the training phase in the simulation, mimicking the positional uncertainty inherent in VLM detection. We also added noise to bounding box positions and image inputs, simulating VLM detection uncertainty and camera noise, including lighting variations.

VII. Results

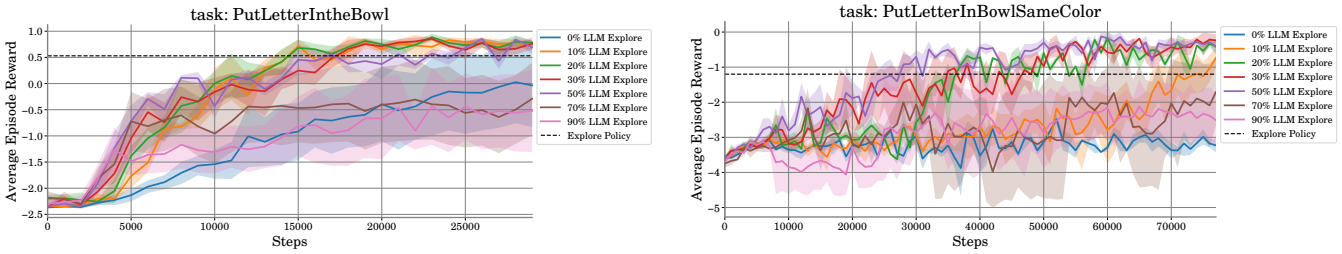
A. Simulation Results

We investigated the effect of varying LLM-based exploration frequencies on training convergence, using $\epsilon \in \{0.0, 0.1, \ldots, 0.9\}$, as shown in Figure 4. An ϵ of 0 corresponds to standard SAC. We trained the agents with six random seeds per frequency, and each session began with a 20,000-step warm-up phase without LLM exploration, as no significant policy improvements were observed during this phase. Post-warm-up results, shown in Figure 4 and detailed in Table II for both short- and long-horizon tasks, indicate that ExploRLLM consistently outperforms LLM-only policies across various exploration frequencies.

In the short-horizon task (Figure 4a), training without LLM-based exploration is often unstable, resulting in either a successful policy or failure to converge within the duration

TABLE I: Results of 50 evaluation episodes for short-horizon (SH), long-horizon (LH), and different initialization methods: no object overlap (NO) and allowed overlap (AO). ExploRLLM standard deviations are shown for 6 seeds.

Method	Overall success rate				Low-level error rate			
	SH NO	SH AO	LH NO	LH AO	SH NO	SH AO	LH NO	LH AO
ExploRLLM (20%)	0.86±0.05	0.80±0.06	0.70±0.11	0.54±0.09	0.14±0.05	0.20±0.06	0.18±0.10	0.22±0.9
ExploRLLM (0%)	0.56±0.40	0.48±0.36	–	–	0.32±0.24	0.42±0.30	–	–
CaP*	0.60	0.48	0.38	0.30	0.38	0.52	0.42	0.48
Socratic Models + CLIPort	0.78	0.64	0.50	0.36	0.22	0.28	0.22	0.28
Inner Monologue + CLIPort	0.82	0.72	0.58	0.42	0.18	0.26	0.20	0.24



(a) *Pick the [pick letter] and place it in the [place color] bowl* (SH). (b) *Put all letters in the bowl of the corresponding color* (LH).

Fig. 4: Training curves for varying exploration rates in SH and LH tasks. ExploRLLM outperforms the exploration policies (dashed lines) and RL without LLM-based exploration ($\epsilon = 0$). In the LH task, LLM-based exploration is crucial for success.

TABLE II: ExploRLLM training returns for varying ϵ.

Explore ϵ (%)	SH Task (25k steps)	LH Task (75k steps)
0	-0.03 ± 1.13	-3.22 ± 0.29
10	0.74 ± 0.13	-0.73 ± 0.40
20	0.79 ± 0.06	-0.42 ± 0.31
30	0.76 ± 0.16	-0.23 ± 0.26
50	0.70 ± 0.17	-0.40 ± 0.23
70	-0.29 ± 0.98	-1.71 ± 1.38
90	-0.52 ± 1.12	-2.51 ± 1.09
Exploration Policy	0.53	-1.2

TABLE III: Success rate (%) of SH ExploRLLM with [4].

Task Settings	Seen	Unseen Color	Unseen Letters
Socratic Models + ExploRLLM	74	68	56
Socratic Models + CLIPort	72	50	34

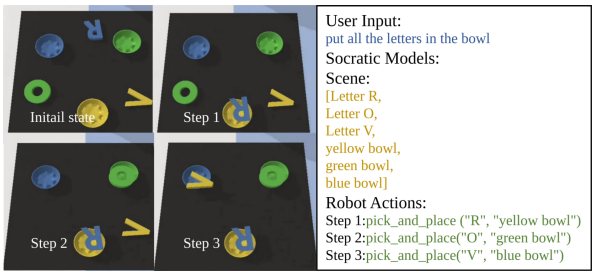


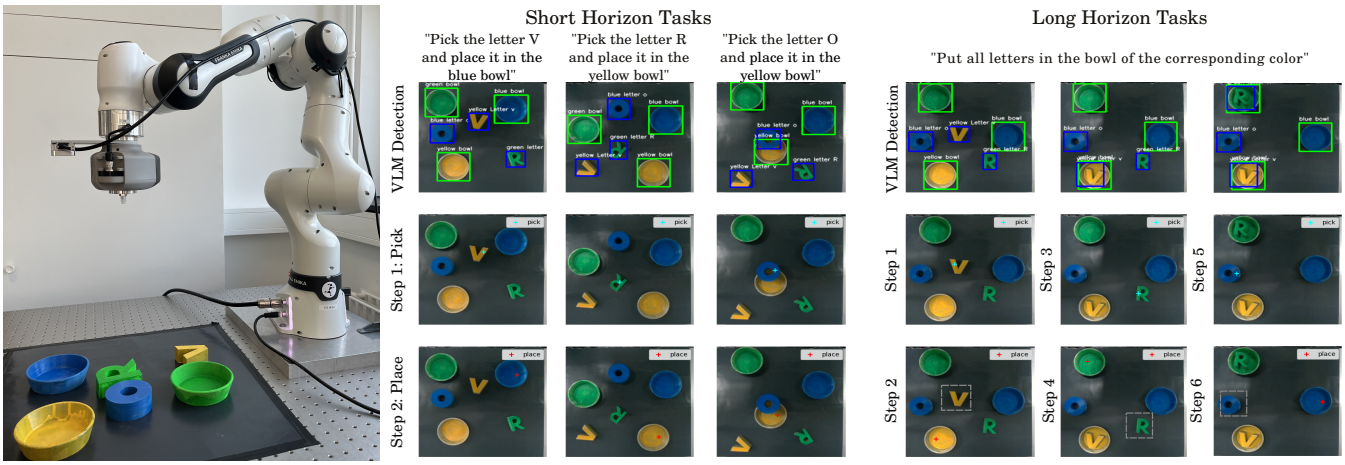
Fig. 5: Short-horizon ExploRLLM policies can be used in long-horizon tasks with zero-shot LLM planners, e.g., [4].

of our experiments. Training stabilizes and converges faster when the exploration frequency is within $0 < \epsilon \leq 0.5$, with minimal variation across different ϵ values. However, increasing ϵ beyond 0.5 reduces the proportion of online data, slowing progress and introducing greater instability into the training. For long-horizon tasks, Figure 4b shows that higher frequencies of LLM-based exploration ($0 < \epsilon \leq 0.5$) correlate with faster training. These results highlight the importance of LLM-based exploration in navigating complex tasks by guiding experience toward the optimal region, thereby mitigating challenges from large observation and action spaces. However, similar to the short-horizon tasks, excessive exploration rates introduce instability and fail to converge within the duration of the experiments.

To evaluate the effectiveness of ExploRLLM, we benchmark its performance against four baselines: ExploRLLM without the LLM-based exploration policy, the CaP-style policy [14] (our exploration policy), Socratic Models [4], and Inner Monologue [12]. Our Socratic Models and Inner Monologue implementations use ViLD [26] as the object detector and GPT-4 [2] as a multi-step planner. The individual steps are executed by a pre-trained CLIPort [27] model with 500 demonstrations. The key difference between Socratic Models and Inner Monologue is that Inner Monologue features a success detector that can identify mistakes.

During evaluation, the letter colors range from seen to unseen colors. Tasks and initialization methods vary, with "NO" indicating no overlap between the initial positions of letters and bowls and "AO" allowing overlaps. These configurations assess each method's robustness in handling complex object relationships.

For short-horizon tasks, as shown in Table I, ExploRLLM maintains stable performance. In contrast, versions without the exploration policy have not all converged and exhibit high variance in success rates and low-level errors. Our method surpasses other methods for LLM-generated policies in success rates, reduces robot behavior errors, and minimizes the performance gap between NO and AO scenarios, emphasizing the exploration policy's role in correcting FMs' inaccuracies. In contrast, CLIPort-based methods struggle



(a) Real-world experimental setup.

(b) Visualization of VLM detections and pick and place actions.

Fig. 6: ExploRLLM can be practically applied using a sim-to-real approach with transfer due to VLM object detections.

with novel scenarios or complex geometric object relationships. For long-horizon tasks, RL agents without LLM-based exploration fail to converge within the duration of the experiment. As shown in Table I, ExploRLLM outperforms Socratic Models, Inner Monologue, and LLM-generated policies, achieving superior results in long-horizon tasks.

Although our short-horizon agent is trained specifically for a pre-defined pick-and-place task, our approach can transfer to unseen long-horizon tasks in similar environments. This is made possible by integrating a zero-shot planner framework, such as Socratic Models [4]. This framework effectively breaks down user-provided input into individual action steps, each serving as a distinct language command for our single-step RL agent, as illustrated in Figure 5. Following the execution of each command, the task space is reset, allowing for the subsequent command to be executed. Apart from unseen colors, unseen letters are also included to evaluate the generalization capabilities of unseen scenarios. Table III demonstrates that the short-horizon ExploRLLM adapts to these settings, surpassing earlier Socratic Models versions. Using VLMs to provide bounding boxes and positions, our approach reformulates the observation space, enabling RL to focus on learning the physical attributes of objects, which is crucial for precise pick-and-place tasks. This strategy minimizes distractions from variations in colors and shapes.

B. Real-World Results

We evaluated ExploRLLM in two real-world scenarios: one replicating all letters from the simulation and another introducing the unseen letter 'C'. Each scenario was tested over 15 episodes. The short-horizon ExploRLLM achieved success rates of 66.6% for seen letters and 53.3% for the unseen letter scenario. In comparison, the long-horizon ExploRLLM recorded success rates of 40% for seen letters and 33.3% for unseen letters. Despite the sim-to-real gap, our approach shows promising results without additional real-world training. As the VLM extracts the observation space, the RL agent trained in simulation is less distracted by real-world noise. Figure 6b illustrates the adaptability of our method in handling diverse object orientations, understanding logical relationships between objects, and executing long-horizon tasks in real-world settings. However, challenges remain with noise in the color and depth perception of objects, which hampers the RL agent's ability to manipulate objects. Using a photorealistic simulator with extensive domain randomization is expected to improve performance.

VIII. Conclusion and Discussion

In this work, we presented ExploRLLM, a method that combines RL with FMs. ExploRLLM accelerates RL convergence by using actions informed by LLMs and VLMs to guide exploration, demonstrating the benefits of integrating the strengths of both RL and FMs. We evaluated our method on tabletop manipulation tasks, showing superior success rates compared to policies based solely on LLMs and VLMs. ExploRLLM also generalizes unseen colors, letters, and tasks better. Ablation experiments with varying levels of LLM-guided exploration indicated that extensive tuning of this parameter is unnecessary as values of $0 < \epsilon \leq 0.5$ showed convergence improvements. Additionally, we validated the method's ability to transfer learned policies from simulation to real-world scenarios without additional training through real robot experiments. Currently, our framework focuses on tabletop manipulation, but we plan to extend it to a broader range of robotic manipulation tasks. While the system can correct low-level robotic actions, it struggles with mitigating high-level errors that are less frequent in simulations. Future work will focus on addressing these high-level discrepancies.

IX. Acknowledgments

Research reported in this work was partially or completely facilitated by computational resources and support of the Delft AI Cluster (DAIC) [29] at TU Delft (RRID: SCR_025091), but remains the sole responsibility of the authors, not the DAIC team.

REFERENCES

[1] N. Di Palo, A. Byravan, L. Hasenclever, M. Wulfmeier, N. Heess, and M. Riedmiller, "Towards a unified agent with foundation models," in *Workshop on Reincarnating Reinforcement Learning at ICLR*, 2023.

[2] OpenAI, "GPT-4 technical report," 2023.

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

[4] A. Zeng, M. Attarian, B. Ichter, K. M. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. S. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, and P. Florence, "Socratic models: Composing zero-shot multimodal reasoning with language," in *International Conference on Learning Representations (ICLR)*, 2023.

[5] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "VoxPoser: Composable 3D value maps for robotic manipulation with language models," in *Conference on Robot Learning (CoRL)*. PMLR, 2023.

[6] T. Carta, C. Romac, T. Wolf, S. Lamprier, O. Sigaud, and P.-Y. Oudeyer, "Grounding large language models in interactive environments with online reinforcement learning," in *International Conference on Machine Learning (ICML)*. PMLR, 2023.

[7] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. P. Saldyt, and A. B. Murthy, "Position: LLMs can't plan, but can help planning in LLM-modulo frameworks," in *International Conference on Machine Learning (ICML)*, 2024.

[8] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[10] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[11] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian *et al.*, "Do as I can, not as I say: Grounding language in robotic affordances," in *Conference on Robot Learning (CoRL)*. PMLR, 2023.

[12] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, T. Jackson, N. Brown, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," in *Conference on Robot Learning (CoRL)*. PMLR, 2023.

[13] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "ProgPrompt: Generating situated robot task plans using large language models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[14] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[15] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[16] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," in *Conference on Robot Learning (CoRL)*. PMLR, 2023.

[17] J. Song, Z. Zhou, J. Liu, C. Fang, Z. Shu, and L. Ma, "Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics," *arXiv preprint arXiv:2309.06687*, 2023.

[18] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," in *International Conference on Learning Representations (ICLR)*, 2024.

[19] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas, "Guiding pretraining in reinforcement learning with large language models," in *International Conference on Machine Learning (ICML)*. PMLR, 2023.

[20] E. Triantafyllidis, F. Christianos, and Z. Li, "Intrinsic language-guided exploration for complex long-horizon robotic manipulation tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[21] L. Chen, Y. Lei, S. Jin, Y. Zhang, and L. Zhang, "Rlingua: Improving reinforcement learning sample efficiency in robotic manipulations with large language models," *IEEE Robotics and Automation Letters*, vol. 9, no. 7, pp. 6075–6082, 2024.

[22] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," in *Conference on Robot Learning (CoRL)*. PMLR, 2021.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*. PMLR, 2018.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[25] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12 348–12 355, 2021.

[26] X. Gu, T.-Y. Lin, W. Kuo, and Y. Cui, "Open-vocabulary object detection via vision and language knowledge distillation," in *International Conference on Learning Representations (ICLR)*, 2022.

[27] M. Shridhar, L. Manuelli, and D. Fox, "CLIPort: What and where pathways for robotic manipulation," in *Conference on Robot Learning (CoRL)*. PMLR, 2022.

[28] B. van der Heijden, J. Luijkx, L. Ferranti, J. Kober, and R. Babuska, "Engine agnostic graph environments for robotics (EAGERx): A graph-based framework for sim2real robot learning," *IEEE Robotics and Automation Magazine*, pp. 2–15, 2024.

[29] Delft AI Cluster (DAIC), "The Delft AI Cluster (DAIC), RRID:SCR_025091," 2024. [Online]. Available: https://doc.daic.tudelft.nl/