Prompted to Explore: Training-Time-Only LLM Proposals for Sample-Efficient Pushing Grasping Policies

Meiyuan Gong[1], Yan Gao[2], Ze Ji[3,1]
[1]School of Engineering, Cardiff University, Cardiff, United Kingdom
[2]College of Mechanical and Electrical Engineering, Harbin Engineering University, Harbin, China
[3]College of Mechanical and Electrical Engineering, Hohai University, Changzhou, China
gongm3@cardiff.ac.uk, gaoyan07@hrbeu.edu.cn, jiz1@cardiff.ac.uk

***Abstract*—Learning long-horizon robot manipulation remains difficult and time-consuming, especially under sparse rewards due to inefficient exploration and reward assignment. We present a minimal integration of large language models (LLMs) with reinforcement learning (RL) in which the LLM is used strictly as an *online action proposer* during early training to help with the RL agent. Given a task description and a compact scene abstraction, the LLM outputs a single parameterised primitive (pushing or grasping) that is validated by lightweight safety checks and mixed with policy actions via an epsilon-greedy scheduler. Besides, we study two simple policies that preserve the same low-level learner (grasping-biased model): (i) *pushing-first reward shaping* that temporarily increases the relative reward of pushing early in an episode, and (ii) a *Proximal Policy Optimization (PPO) high-level switch* that explicitly selects between push and grasp while Soft Actor-Critic (SAC) handles low-level parameterisation. In simulation, LLM-guided exploration accelerates early learning, and the proposed methods help further mitigate grasping overuse in the cluttered environment. Besides, the LLM-guided RL agent could achieve a higher success rate at 89.5% compared with the baseline method at 68%.**

***Index Terms*—Robotic manipulation, Large Language Models, Reinforcement Learning, Grasping and pushing.**

I. Introduction

Learning effective robot manipulation policies in cluttered, partially occluded scenes is difficult due to *complicated reward setting* and long-horizon reward assignment. Even when a task can be solved by composing a small set of parameterised primitives—notably *pushing* to expose a target and then *grasping* to complete the pickup task—pure reinforcement learning (RL) often fails to discover reward-producing behaviours early in training, leading to noisy value estimations and also results in inefficient exploration and slow convergence [1]–[3]. Traditional grasping/pushing pipelines based on geometry and hand-crafted heuristics require extensive engineering and often lack adaptability in cluttered scenes [4], [5].

Recent work has shown that LLMs provide a promising source of high-level priors—such as goal structure, object relations, and commonsense physical constraints—that can bias exploration and reduce the search space during the early stages of the learning process [6]. Recent systems demonstrate how vision-language(-action) models can bridge perception, reasoning, and control [7]–[9], and how LLMs can guide RL via structured prompts or pretraining signals [10]. However, many prior integrations either rely on complex planning/tool interfaces or depend on offline demonstrations and behaviour cloning, making deployment and attribution of gains more difficult [6]. What is needed is a *minimal, training-time-only* integration that improves the *data distribution* collected during early learning *without* altering the RL objective or requiring additional supervision.

In our work, we adopt a minimalist approach where LLM is used strictly as an *online action proposer* during the early phase of training. Given a task description and a compact scene abstraction, it outputs a single parameterised primitive (grasping or pushing) to guide the agent to conduct actions. A lightweight, rule-based validator enforces hard safety and feasibility constraints (workspace bounds, basic reachability, height limits). The proposed action is then mixed with the policy's own action via an ε-greedy scheduler; ε_t is decreased to zero as training progresses, after which the learned policy assumes full control. Crucially, the RL policy objective and updates remain unchanged, and the LLM is disabled at evaluation. Thus, the LLM influences only the *distribution of early experience* for the replay buffer.

Applying this scheme to long-horizon tabletop manipulation with sparse rewards reveals a consistent *grasp bias*: policies over-select grasps while under-utilising pushes in cluttered scenes—likely because successful grasps yield immediate reward whereas pushes are instrumentally valuable only when they enable later grasps. We therefore study two simple remedies that preserve the same low-level RL controller: (i) a *push-first reward shaping* schedule that temporarily increases the relative value of pushing early in an episode and then decays so that grasping is favoured once occlusions are cleared; and (ii) a *Proximal Policy Optimization (PPO) high-level switch* that explicitly selects between *push* and *grasp*, while low-level execution and parameterisation remain with RL plus LLM-guided exploration. These variants directly target the discrete mode-selection problem underlying grasp overuse.

Our work can be concluded into three contributions:

1) **Minimal LLM–RL integration.** A training-time-only mechanism that uses an LLM as an online proposer of parameterised primitives, validated by simple rules

and mixed via ε-greedy, leaving the RL learning rule untouched and avoiding dependence on test-time LLM calls.

2) **Diagnosis and mitigation of grasp bias.** We identify a recurrent failure mode under sparse rewards (overuse of grasps, underuse of pushes) and propose two complementary remedies: push-first reward shaping and a PPO high-level switch for discrete primitive selection.
3) **Data-centric gains without extra supervision.** We show that performance improvements arise from reshaping early experience rather than adding auxiliary losses or priors, isolating the contribution of LLM-guided exploration.

II. Related Work

Robotic manipulation has seen significant progress with the integration of machine learning and, more recently, LLMs show strong capability in action planning and reward shaping. Traditional methods for grasping and pushing often rely on geometric analysis and hand-crafted heuristics [4] [5], but these approaches require extensive engineering and often lack adaptability in complex or cluttered scenes. End-to-end deep learning methods, including RL and imitation learning, have demonstrated robust performance for manipulation in diverse environments [1], [2]. However, these methods are typically data-hungry and difficult to interpret.

Recently, LLMs such as GPT-4 [11] have shown promise in bridging perception, reasoning, and control for robots. A line of work investigates LLMs for high-level task planning, scene reasoning, and instruction following [7], [8], [12]. Others explore leveraging LLMs for low-level action generation, mapping structured scene descriptions directly to parameterised actions for tasks such as grasping and pushing [13], [14].

LLM-assisted action planning. A variety of recent systems, including RT-2 [7] and VoxPoser [8], use LLMs or vision-language models (VLMs) as intermediaries to interpret robot scene observations and generate manipulation commands. These works primarily focus on multi-step or language-driven long-horizon planning, utilising the LLM to break down complex tasks into primitives, and to guide RL or imitation learning policies.

Single-step manipulation with LLMs. While most prior research emphasises long-horizon planning, there is increasing interest in using LLMs for single-step, state-conditioned action generation [10], [15]. Such approaches prompt the LLM with structured observations of the robot and environment, and output a continuous or discrete action (e.g., 7D control vector for grasping/pushing). This enables robots to make context-aware decisions in a data-efficient manner, but challenges remain in action validity, robustness, and safety.

Prompt design and action validity. The performance of LLM-based policies heavily depends on prompt design, including how scene information, task goals, and action examples are structured [6], [8]–[10]. Recent work improves prompt engineering and interfaces to increase the rate of valid, physically executable actions—e.g., value-map grounding and typed primitive specifications—thereby boosting successful graspings and reducing spurious actions [7], [8], [13]. Nevertheless, it remains an open question how best to encode state, object geometry, and dynamic constraints to maximise LLM utility for manipulation, as surveyed in [6].

LLM-guided RL training. Tan et al. propose an LLM-guided policy modulation framework that targets training bottlenecks by detecting *critical states* from a suboptimal agent's trajectories and using the LLM to provide action suggestions and implicit reward signals—without extra model training or human labels [16]. Hao et al. further present *LLM-Explorer*, a plug-in module that samples the agent's learning trajectory and periodically prompts an LLM to produce task-specific exploration distributions, adaptively enhancing policy exploration across Atari and MuJoCo benchmarks [17].

Unlike prior work that uses LLMs for long-horizon planning, test-time decision making, or offline imitation, our approach employs an LLM *only during training* as a lightweight proposer of immediate push/grasp actions. The RL objective and updates remain unchanged, and no test-time LLM calls or offline demonstrations are required. This design isolates gains to improved early exploration and avoids deployment overhead while directly addressing grasp–push mode selection.

III. Method

We propose a minimal two-stage pipeline (see Fig. 1) in which a large language model (LLM) is used solely as an *online* action proposer to aid early exploration under sparse rewards, while RL is responsible for policy learning.

Stage I, given a task description g and a structured scene abstraction $\xi(s)$, the LLM outputs a single parameterised primitive (grasping or pushing). The proposal is passed through a light, rule-based filter enforcing hard constraints (workspace bounds, basic reachability, height limits, and a fixed clearance margin). Only proposals that satisfy these checks are executed to conduct the task.

In Stage II, we train an RL agent off-policy on transitions collected during interaction. The LLM is mixed into data *collection* via an ε-greedy controller during the first T_{guide} environment steps and is then fully disabled; the RL objective and update rules remain unchanged. This design isolates the contribution of LLM-proposed actions to the *early data distribution* without introducing demonstration losses or offline seeding.

A. LLM Action Proposer

a) Role and scope.: We employ the LLM as a one-step action proposer in a parameterised primitive space. In each decision state s, the agent provides a brief task description g and a compact scene abstraction $\xi(s)$; the LLM returns a single manipulation primitive (grasping or pushing) with 5D parameters, in which the last value indicates the action type. The LLM is not used for the implementation of policies, planning over horizons or evaluation, and it does not produce learning signals beyond the proposed action itself.

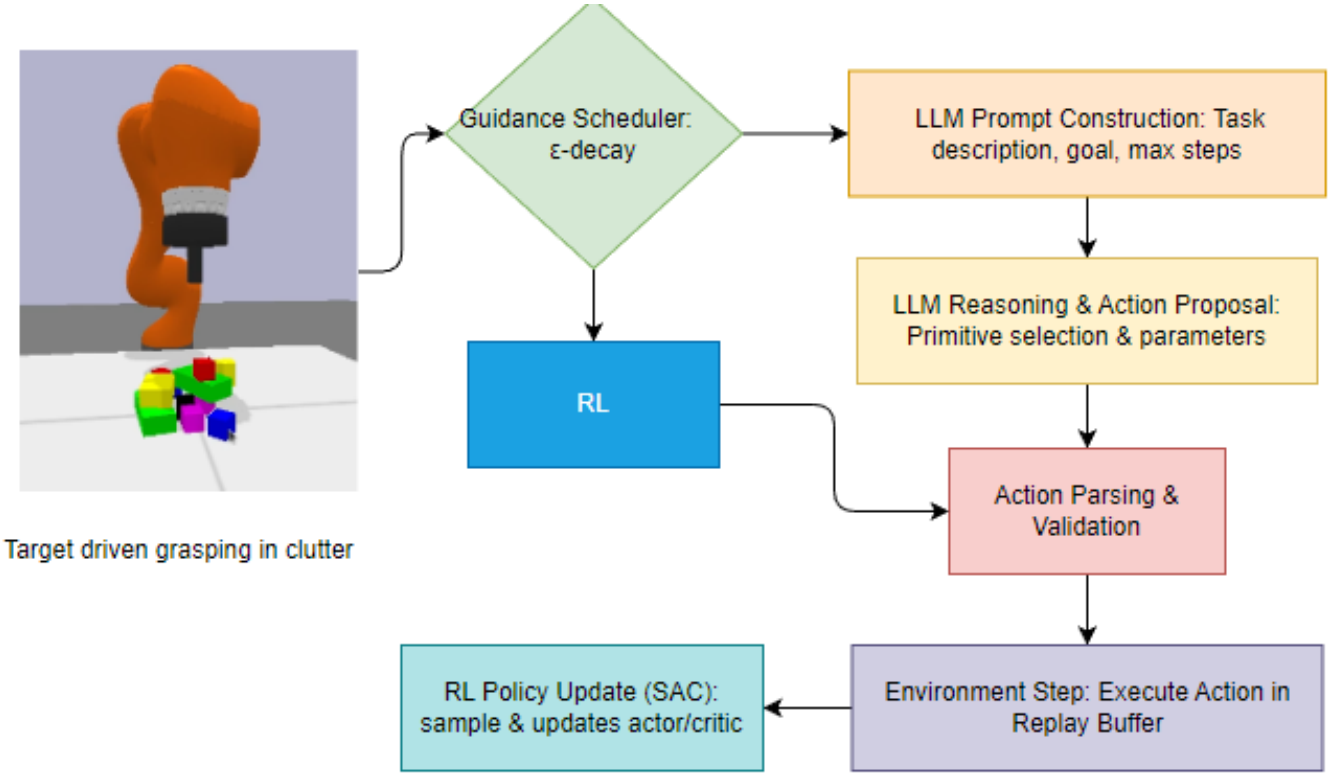

Fig. 1: **System Overview:** An LLM–RL pipeline in which an LLM proposes pushing/grasping primitives from a compact scene summary, a rule-based validator enforces feasibility, and an ε-greedy mixer blends LLM suggestions with Soft Actor-Critic (SAC) policy actions to accelerate the learning process.

b) Prompt composition.: At each decision step t, the current observation s_t is converted into a compact scene abstraction $\xi(s_t)$ and combined with the task goal g_t and the action/output constraints to instantiate a structured prompt for the LLM. Fig. 2 schematically illustrates this online prompt-construction process rather than merely an example prompt template.

Specifically, the goal specification g_t defines the designated target object and the success condition. The scene abstraction $\xi(s_t)$ provides a compact object-centric description of the current scene, including object poses and sizes, gripper state, and recent action context, while omitting unnecessary low-level sensory details. The third component specifies the action schema and output requirements, including the available primitives, reasoning budget, and required JSON format. The resulting prompt is then passed to the LLM to predict the next primitive action and its associated parameters.

c) Output contract.: For deterministic execution, the LLM is required to emit a *single* primitive with fully instantiated numeric parameters, rather than a free-form response. The controller ingests only the primitive type and its parameters; any accompanying textual justification, if present, is ignored for control purposes. To ensure unambiguous parsing, the primitive type is encoded as a binary indicator $u \in \{0, 1\}$, with $u = 1$ denoting a grasping and $u = 0$ denoting a pushing. This contract constrains output variability and eliminates format ambiguity.

d) Rule-based filtering.: Before execution, the proposal passes a light, deterministic filter that makes sure the proposed actions are within workspace bounds and height limits relative to the support plane. Proposals violating any rule are discarded without execution. Let $\mathcal{V}$ denote a deterministic validator and define the valid-action set $\mathcal{A}_{\text{valid}} = \{\, a \in \mathcal{A} \mid \mathcal{V}(a) = \text{valid} \,\}$. We retain only $a \in \mathcal{A}_{\text{valid}}$. The filter is deliberately minimal and policy-agnostic, contributing to LLM proposals transparent and reproducible.

B. LLM-Guided Exploration for RL

During the early phase of training, we use the LLM *online* as an exploration guide. At each decision state s, an ε-greedy controller selects either (i) the LLM-proposed primitive a_{LLM} with probability ε_t, or (ii) the policy action $a_\pi \sim \pi_\theta(\cdot \mid s)$ with probability $1-\varepsilon_t$. The schedule ε_t is decreased to zero over the first T_{guide} training steps, after which control is fully decided by the learned policy. All executed actions—regardless of their source—are stored as standard transitions (s, a, r, s') in the replay buffer.

This design leaves the RL objective and update rules unchanged; the LLM only influences the *data distribution* collected early in training. By injecting feasible, task-aligned

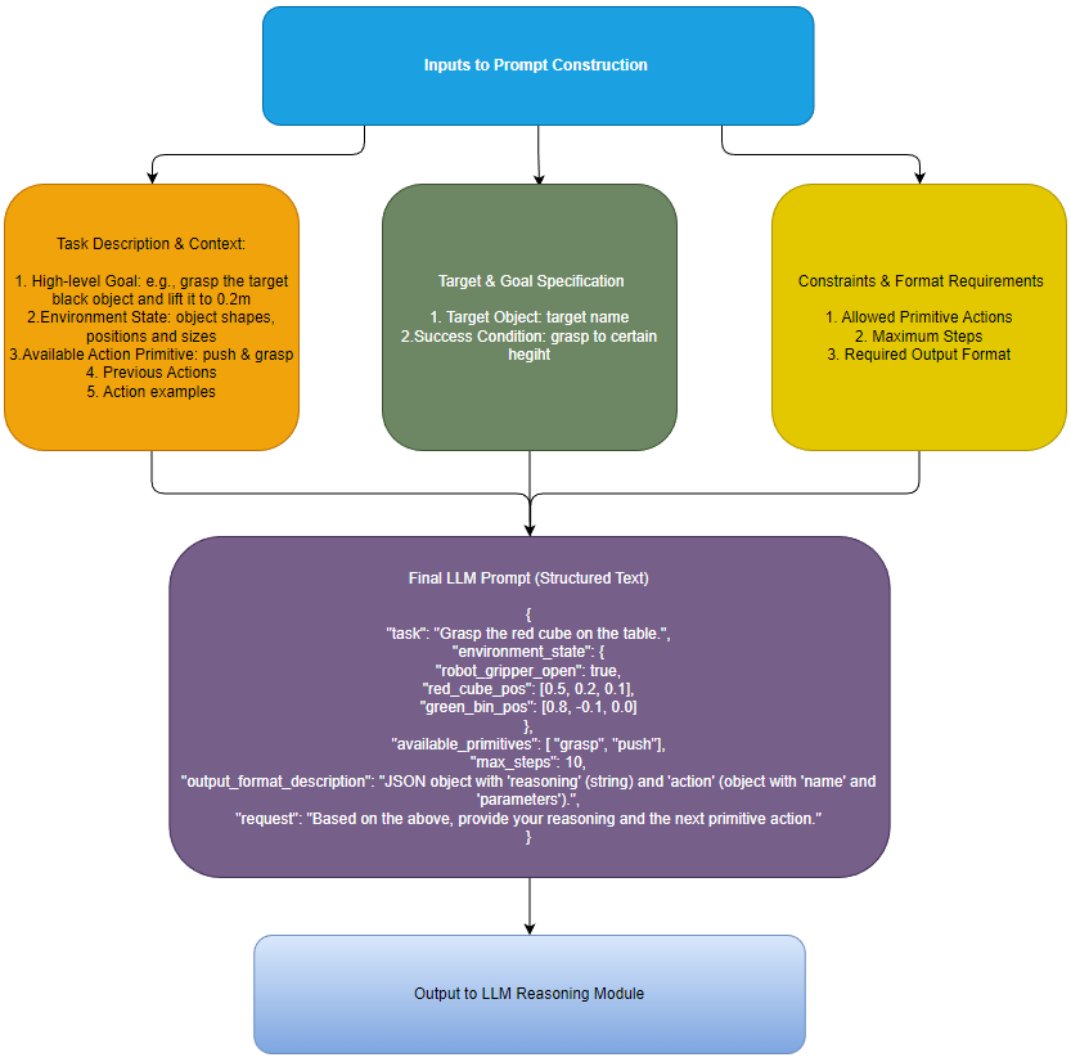

Fig. 2: **Prompt design.** The prompt is instantiated online from three components: (a) the compact state abstraction $\xi(s_t)$, which summarizes the current scene state, gripper status, available primitives, and recent action history; (b) the goal specification g_t, including the designated target object and the success condition; and (c) action constraints and output requirements, including the primitive set, reasoning budget, and JSON action schema. These inputs are fused by the prompt constructor into a structured text prompt, which is sent to the LLM to predict the next action primitive and its parameters. Thus, the figure illustrates the runtime prompt-generation pipeline rather than only an example prompt template.

primitives (e.g., well-aligned graspings or obstacle-clearing pushinges) with nonzero probability, the agent is more likely to reach rewarding states under the long-horizon task, which in turn stabilises value estimates and accelerates policy improvement.

C. RL Setup

State (observation): we use a *state-based* representation $s \in \mathcal{S}$ that concatenates: (i) robot state (end-effector pose and gripper opening); (ii) a compact scene abstraction (blocks state and sizes); The low-level controller acts in a *primitive* space. An action is $a = (\theta, c)$ with $c \in \{\texttt{pushing}, \texttt{grasping}\}$ θ is 6D parameters. During learning, the actor emits a scalar $u \in (0, 1)$ and two parameter heads; we set $c = \texttt{grasping}$ if $u \geq 0.5$ (otherwise $\texttt{pushing}$) and execute θ_c. This matches the LLM's output contract.

Reward and success: The environment provides a *sparse* task reward $r_{\text{task},t} > 0$ only upon success, i.e., when the target object is grasped and lifted above the threshold. All other steps receive zero reward, except for small environment penalties when applicable (e.g., leaving the workspace). When *pushing-first* shaping is enabled (Sec. III-F), the agent is trained with a shaped reward defined in that section, which augments the sparse task reward with additional guidance favouring useful

pushing behaviour during training.

D. RL Objective and LLM-Guided Data Collection

We adopt SAC (stable-baselines3 [18]). Critic targets and actor updates are computed from replayed transitions (s, a, r, s'), where s denotes the current state, a the executed action, r the resulting reward, and s' the next state. These updates are *independent* of the action source (LLM or policy). Denoting the temperature by α, the policy by π_θ, and critics Q_{ϕ_1}, Q_{ϕ_2} with target networks, losses follow the baseline SAC formulation; no demonstration losses or priors are added.

During the first T_{guide} environment steps, an ε-greedy scheduler mixes validated LLM proposals with the policy's own actions; thereafter, the LLM is disabled, and the learned policy acts alone. Under sparse rewards, purely random/entropy-driven exploration rarely reaches rewarding states, creating noisy targets and slow convergence. Online LLM guidance injects feasible, goal-consistent primitives early on, increasing the incidence of non-zero returns without changing the learning rule; this stabilises value estimates and shortens cold start.

E. Failure Mode under Sparse Rewards: Grasp Bias

In cluttered scenes, we repeatedly observe a *grasping bias*: policies over-select grasps and under-utilise pushing, likely because successful grasping produces immediate reward, whereas pushing is valuable mainly when they enable later grasps. To address this mode-selection imbalance *without* altering the low-level learner, we evaluate two possible solutions.

F. Pushing-First Reward Function

We employ a training-only shaping scheme that biases the agent toward *pushing first* and *grasping later*. Let $c_t \in \{\texttt{pushing}, \texttt{grasping}\}$ denote the chosen primitive, τ the within-episode step index (reset at the beginning of each episode), $\text{cnt}^{\text{grasping}}$ the number of grasping attempts, and $\text{cnt}^{\text{pushing}}$ the number of early-pushing bonuses already assigned. The shaped reward is

$$\begin{aligned} r_t = r_{\text{task},t} &+ w_{\text{pushing}}(\tau)\,\mathbf{1}\{c_t = \texttt{pushing}\}\,\mathbf{1}\Big\{\text{cnt}^{\text{pushing}} < N_{\text{pushing}}\Big\} \\ &- \Big(\kappa_0 + \kappa_{\text{tax}}\,\mathbf{1}\Big\{\text{cnt}^{\text{grasping}} \geq B_{\text{grasping}}\Big\}\Big)\,\mathbf{1}\{c_t = \texttt{grasping}\} \\ &+ w_{\text{time}}\Big(-\lambda_{\text{time}}\Big(1 + \eta\,\tfrac{\tau}{T_{\max}}\Big)\Big) + w_{\text{ws}}\,\phi_{\text{ws}}(s_{t+1}). \end{aligned} \tag{1}$$

Here, $r_{\text{task},t}$ is the sparse task reward, $\phi_{\text{ws}}(\cdot) \leq 0$ penalises leaving the manipulable workspace, and $\mathbf{1}\{\cdot\}$ denotes the indicator function, which equals 1 if the condition holds and 0 otherwise. The early-pushing weight decays over a short within-episode window:

$$w_{\text{pushing}}(\tau) = \lambda_0 \max\left(0,\, 1 - \frac{\tau}{\tau_{\text{cut}}}\right).$$

Counters $\text{cnt}^{\text{pushing}}$ and $\text{cnt}^{\text{grasping}}$ are updated online and reset at the end of each episode. Thus, only the first N_{pushing} pushes receive a *front-loaded* bonus, with earlier pushes receiving a larger reward. Grasping carries a small cost and an additional soft-budget tax after B_{grasping} attempts, while mild time and workspace penalties regularise behaviour. Shaping is applied only during training; evaluation uses the unshaped reward.

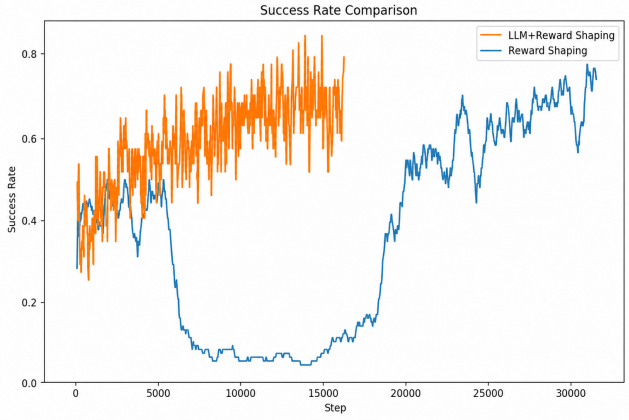

Fig. 3: **Training success rate comparison.** LLM+Reward Shaping reaches a success rate of around 0.7 within approximately 10k–15k training steps, whereas Reward Shaping alone requires roughly 24k–30k steps to achieve a comparable level. This suggests that LLM guidance improves the speed at which the agent attains similar policy performance during training.

G. PPO High-Level Action Selection (Primitive Gating)

We also use PPO as a *high-level* categorical policy $\pi_\omega(c \mid s)$ that selects $c \in \{\texttt{pushing}, \texttt{grasping}\}$; a pretrained low-level controller (the SAC above) then parameterises and executes the chosen primitive. The gate is trained with a minimal-shaped signal.

$$\begin{aligned} \tilde{r}_t = r_{\text{task},t} &+ k_{\text{prog}}\big(\gamma\,\Phi(s_{t+1}) - \Phi(s_t)\big) \\ &- k_{\text{grasping}}\,\mathbf{1}\{c_t = \texttt{grasping}\} - \lambda_t\,\mathbf{1}\{c_t = \texttt{grasping}\}, \end{aligned}$$

where $\Phi(s)$ is a progress potential, the progress term is potential-based shaping (policy-invariant if it uses the same γ), k_{grasp} deters premature grasping, and λ_t is a Lagrange multiplier that regulates grasping frequency.

Both variants use the same LLM+RL scheme: during an initial guidance window, the LLM provides *online* pushing/grasping proposals that are validated and mixed with the low-level controller via an ε-greedy scheduler; after this window, the LLM is disabled and all evaluations proceed without test-time LLM access.

IV. Experiments and Results

A. Experimental Setup

We evaluate whether an *online* LLM proposer based on GPT-4o mini [19] can accelerate learning in long-horizon tabletop manipulation using a KUKA *iiwa14* with a parallel-jaw gripper in PyBullet. Tasks are grasping-centric but typically require preparatory pushing under clutter and occlusion. A low-level SAC agent trained with sparse rewards

TABLE I: Main results: final success rate and efficiency (avg. steps per successful episode). Higher success and fewer steps are better.

Method	**Success rate (%) ↑**	**Avg. steps ↓**
LLM+SAC + *reward shaping*	89.5	2.66
LLM+SAC + *PPO gate*	78.5	4.23
SAC + *reward shaping* (no LLM)	68.0	3.45

exhibits a consistent *grasping bias* (over-selecting grasping and under-using pushing, since successful grasping comes with immediate positive return). To mitigate this while keeping the SAC learner unchanged, we employ the LLM only during an early guidance window of 5,000 environment steps: at state s, the controller executes the validated LLM proposal with probability ε_t and the policy action otherwise; ε_t is monotonically annealed to 0 over the window, after which the LLM is disabled.

Following methods are compared: **SAC + reward shaping (no LLM).** A low-level SAC agent trained under the same sparse task reward augmented with the push-first shaping in Sec. III-F. No LLM proposals are used at any time.

LLM+SAC + push-first shaping. Identical to the above SAC learner, but during the first T_{guide} env steps we mix validated LLM proposals with probability ε_t (annealed to 0) as in Sec. III-D. Shaping parameters $(\lambda_0, \tau_{\text{cut}}, N_{\text{push}}, \kappa_0, \kappa_{\text{tax}}, B_{\text{grasping}})$ are shared with the no-LLM baseline.

LLM+SAC + PPO gate. A high-level PPO policy $\pi_\omega(c \mid s)$ selects `pushing` vs. `grasping` at each step; the low-level SAC (same architecture as above) parameterises and executes the chosen primitive. The PPO gate is trained with the minimal signal in Sec. III-G (potential-based progress, grasping cost, grasping-rate Lagrangian). During the same guidance window, low-level parameters can still be mixed with LLM proposals via ε_t; after the window, the LLM is disabled.

All methods use the same simulator, validator, observation encoding, network sizes, optimisers, and training budgets; only the presence of LLM mixing and/or the high-level gate differs. Evaluation never uses the LLM, and success is judged by the same task criterion across methods.

We report (a) success rate and (b) average steps per successful episode; we also compare the *time-to-success*, i.e., the number of training steps required to reach a fixed success threshold (Fig. 3).

B. Results

Fig. 3 reports the training steps required by **LLM-guided SAC** versus **SAC** to reach the same success rate, showing improved sample efficiency with LLM guidance.

Overall, as shown in Table I, **LLM-guided** variants improve both time-to-success and final performance. Reward shaping with LLM attains the highest success (89.5%) and the fewest steps (2.66), indicating efficient *push-then-grasp* behaviour. The PPO gate reaches 78.5% with more steps (4.23), consistent with executing more purposeful pushes before grasping. Removing LLM guidance (SAC with shaping only) reduces success to 68.0% (3.45 steps), underscoring the benefit of LLM-guided early experience.

The improvements are greatest in medium to high-occlusion scenes, where clearing pushes are required; low-occlusion scenes show smaller but consistent improvements. LLM-guided variants also exhibit lower seed variance and a clear guidance-window effect—too short under-explores, too long delays takeover—so a small early window yields the best trade-off and faster attainment of target success.

V. Conclusion

We presented a minimal, training-time-only integration of LLMs with RL for tabletop manipulation. The LLM acts solely as an *online* proposer of parameterised pushing/grasping primitives during an early guidance window, mixed with policy actions based on an ε-greedy scheme and validated by simple safety checks; the RL objective (SAC) remains unchanged, and no LLM is used at test time. Empirically, LLM-guided exploration improves both time-to-success and final performance. We utilise a *grasping bias* model, which we mitigate with two complementary solutions: *pushing-first reward shaping* and a *PPO high-level switch* for discrete primitive selection. Together, these achieve higher success and more purposeful pushing–then–grasping sequences than SAC baselines.

For the current limitations and future work. Our validator is minimalist and may reject feasible but unconventional actions; tighter feasibility checks or learned filters could admit more useful proposals without sacrificing safety. The scene abstraction is compact; adding richer yet structured geometry may further improve proposal quality. Finally, real-world transfer will require robustness to sensing noise, latency-aware guidance schedules, and uncertainty-aware filtering—directions we leave for future work.

References

[1] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv preprint arXiv:1806.10293*, 2018.

[3] K. Lee, L. Smith, and P. Abbeel, "Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training," *arXiv preprint arXiv:2106.05091*, 2021.

[4] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transactions on robotics*, vol. 30, no. 2, pp. 289–309, 2013.

[5] M. Kiatos, I. Sarantopoulos, L. Koutras, S. Malassiotis, and Z. Doulgeri, "Learning push-grasping in dense clutter," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8783–8790, 2022.

[6] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, "Large language models for robotics: A survey," *arXiv preprint arXiv:2311.07226*, 2023.

[7] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*. PMLR, 2023, pp. 2165–2183.

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

[9] 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," *arXiv preprint arXiv:2209.07753*, 2022.

[10] 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*. PMLR, 2023, pp. 8657–8677.

[11] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[12] M. Ahn, A. Brohan *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning (CoRL)*, 2022.

[13] H. Zhou, M. Ding, W. Peng, M. Tomizuka, L. Shao, and C. Gan, "Generalizable long-horizon manipulations with large language models," *arXiv preprint arXiv:2310.02264*, 2023.

[14] X. Yuan, Y. ZHENG, F. Zhang, D. ZHANG, H. Mao, and K. Gai, "Generate explorative goals with large language model guidance."

[15] Y. Yin, Z. Wang, Y. Sharma, D. Niu, T. Darrell, and R. Herzig, "In-context learning enables robot action prediction in llms," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 8972–8979.

[16] H. Tan, H. Yan, and Y. Yang, "Llm-guided reinforcement learning: Addressing training bottlenecks through policy modulation," *arXiv preprint arXiv:2505.20671*, 2025.

[17] Q. Hao, Y. Song, Q. Liao, J. Yuan, and Y. Li, "Llm-explorer: A plug-in reinforcement learning policy exploration enhancement driven by large language models," *arXiv preprint arXiv:2505.15293*, 2025.

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of machine learning research*, vol. 22, no. 268, pp. 1–8, 2021.

[19] OpenAI, "Gpt-4o mini: advancing cost-efficient intelligence," https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024, accessed: 2025-09-17.