Novelty Adaptation Through Hybrid Large Language Model (LLM)-Symbolic Planning and LLM-guided Reinforcement Learning

Hong Lu[1], Pierrick Lorang[1,2], Timothy R. Duggan[1], Jivko Sinapov[1], Matthias Scheutz[1]

[1] Department of Computer Science, Tufts University, Medford, MA, USA
[2] Austrian Institute of Technology GmbH, Vienna, Austria

***Abstract*— In dynamic open-world environments, autonomous agents often encounter novelties that hinder their ability to find plans to achieve their goals. Specifically, traditional symbolic planners fail when the robot's planning domain lacks the operators to interact appropriately with novel objects. We propose a neuro-symbolic architecture that integrates symbolic planning, reinforcement learning, and a large language model (LLM) to learn how to handle novel objects. In particular, we leverage the common sense reasoning capability of the LLM to identify missing operators, generate plans with the symbolic AI planner, and write reward functions to guide the reinforcement learning agent in learning control policies for newly identified operators. Our method outperforms the state-of-the-art methods in operator discovery as well as operator learning in continuous robotic domains. Our webpage and code can be access here: helenlu66.github.io/hybridLLMguided/**

I. Introduction

Integration of high-level task reasoning with low-level motion planning within a Task and Motion Planning (TAMP) framework improves the ability of a robotic system to solve complex problems. Using its knowledge of the environment and task structure encoded in the Planning Domain Definition Language (PDDL) [1], a high-level symbolic task planner can generate a sequence of operator executions using the operator's pre- and post-conditions to achieve the desired goal [2] while a low-level motion planner can generate the robot motion for each operator execution (e.g., [3]). Alternatively, each operator could have a policy that was learned through Reinforcement Learning (RL) (e.g., [4]).

However, planning failures can occur due to novelties when the robot cannot find an appropriate sequence of operators to achieve its goal because the solution is outside the robot's planning domain [5], [6]. In fact, common everyday objects, when introduced into a robot's controlled environment, often disrupt the robot's ability to achieve its usual goals because the robot's predefined planning domain lacks the operators necessary for interacting with these objects. In this work, we focus on the introduction of common objects and the challenge of adapting to them, i.e., for the robot to learn how to manipulate them. As such objects might appear within the training data of LLMs, an LLM could effectively reason about interactions involving them [7]. We thus utilize an LLM to perform common sense reasoning to structurally define missing plan operators in the PDDL format. In addition, to efficiently guide the exploration of the RL algorithm for learning the associated operator policies in a continuous space, we prompt the LLM to write dense reward function candidates that are used in a sub-goal curriculum. We launch multiple RL agents—one per dense reward function candidate for each sub-goal—and periodically prune the worst performing candidate until a policy is found that can generate robot motion from the operator's pre-conditions to its post-conditions. Figure 1 illustrates our approach. Our contributions are as follows:

1) We propose a method that leverages the common-sense reasoning of LLMs to solve the missing operator identification problem—a challenge that is intractable for traditional operator discovery methods in the hybrid planning and learning literature.
2) We augment a hybrid planning and learning architecture with LLM generated dense reward functions, thereby significantly speeding up the learning process.
3) We perform evaluations in continuous robotic manipulation domains in the mimicgen simulation environment [8].

*This work has been in part funded by ONR grant N00014-25-1-2479.

II. Related Works

A. Hybrid Planning and Reinforcement Learning

To handle the novelties that might arise in an agent's environment, Goel et al. and Sarathy et al. propose hybrid planning and learning frameworks that extend the definition of MacGyver problems [5], [4], [9]. In these frameworks, when the agent fails to plan due to novelties, RL agents are spawned to bridge the gap by discovering plannable states from which the goal is reachable. In addition, the structured knowledge in the symbolic reasoner provides guidance to the RL agent during its exploration [10], [11], [12], [13], [14], [15]. However, these works often evaluate their approaches in discrete action domains. Recent works in robotics have applied hybrid planning and learning methods to continuous domains [16], [17], [18], [19]. Notably, LEAGUE [18] integrates task planning with skill learning for long-horizon manipulation. These works typically rely on RL agents to discover the missing operator or recover from operator failure, spending considerable time during exploration. To structure this exploration, reward machine-based approaches have proposed decomposing the reward function according to task structure [20], [21]. Nevertheless, even when guided

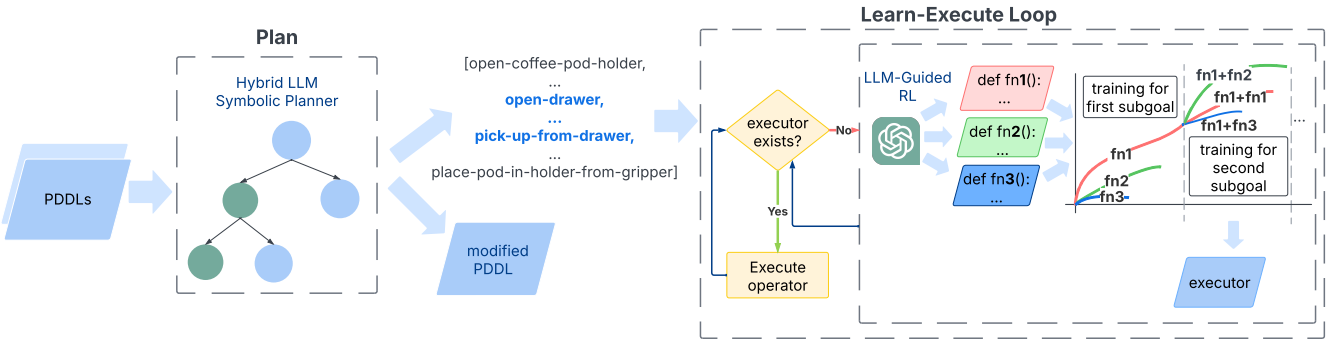

Fig. 1: The plan-learn-execute loop. The Hybrid LLM Symbolic planner parses the domain and problem PDDL files, prompts the LLM to define the missing operator(s) in PDDL, and finds a plan with grounded operators. It lifts the grounded operators, and outputs the modified domain PDDL file with the added lifted operator definitions and the plan. The LLM's newly defined operators are in blue while the existing ones are in black. The learn-execute loop starts by executing the operators in the plan. When it encounters a new operator for which an executor policy does not yet exist, it prompts the LLM to generate dense reward shaping function candidates and launches RL agents to learn the policy. The effects of the operator are treated as sub-goals for the RL agents. One agent is launched per dense reward function candidate and the worst performing agent is eliminated periodically based on sub-goal success rates. The sub-goals are trained in phases. Once training is complete, the best performing policy is saved as the operator's executor. The pseudocode for the algorithm is shown in Algorithm 1.

with symbols, these works never explicitly reason about missing operators, instead relying on RL agents to discover them incidentally through exploration, a strategy that scales poorly in continuous domains where plannable states are difficult to reach by chance. In this work, we explore whether the common sense encoded in LLMs can be utilized to structurally define the missing operators and guide the RL agents via reward shaping, thereby saving exploration time.

B. LLM for Planning

Recent works in robotics planning seek to leverage the semantic understanding, common sense knowledge and the code generation capability of LLMs. For example, Silver et al. propose a prompting pipeline and evaluate LLMs' ability to write code that solves general problems in a given PDDL domain [22], Singh et al. prompt LLMs to write functions that call available robot actions to achieve a goal [23]. Some works aim to increase the executability of the LLM's proposed plan by grounding the plan in the robot's executable actions [24], [25], [26], [23]. To plan in dynamic and partially observable environments, some works introduce interactive methods of prompting the LLMs to incorporate environment feedback and self-correct [27], [28], [29], [30]. Hybrid approaches combine the capabilities of the LLMs with the reliability of symbolic planners to overcome the lack of flexibility of symbolic planners and the tendency to hallucinate of the LLMs. For example, Liu et al. leverage the LLMs' natural language understanding to translate natural language problems into a PDDL problem based on the given PDDL domain and use a symbolic planner to generate the plan [31], and Dagan et al. leverage the common sense encoded in LLMs to make assumptions about the truth value of predicates in partially observable environments [29]. A common assumption made in these works is that the existing operators are enough to reach the goal state. Unlike these prior works that focus on plan synthesis, our approach focuses on domain expansion and leverages the LLM to infer the missing operator(s) when a novel object is injected.

C. LLM for Reinforcement Learning

LLMs can be used to either suggest goals for exploration or provide reward signals. For example, LLMs can suggest the next goal to explore when the RL agent is faced with a new task [32], suggest sequences of sub-goals as potential paths to the goal [33], or suggest auxiliary tasks for learning to maximize the use of previous experiences collected during RL training [34]. In addition, LLMs can provide reward signals either by writing reward function codes or by directly evaluating an episode's outcome [35], [36], [37], [38]. Closest to our work is LEAGUE++ [39], which shares our motivation of leveraging LLM-generated reward functions to guide the acquisition of new robot skills. Despite this conceptual proximity, two important limitations set it apart from our approach. First, LEAGUE++ assumes a fully specified planning domain, meaning it has no mechanism to reason about or recover from the absence of operators needed to handle novel objects — a critical shortcoming for open-world deployment. Second, it commits to a single LLM-generated reward function per skill, leaving the agent exposed to the risk of investing substantial training time following a fundamentally misguided signal. Our work addresses both gaps: we introduce an LLM-guided symbolic search to explicitly identify missing operators, decompose their effects into an ordered sub-goal curriculum, and generate multiple reward function candidates in parallel, applying a genetic algorithm-inspired elimination strategy to progressively retain the most effective reward signal throughout training.

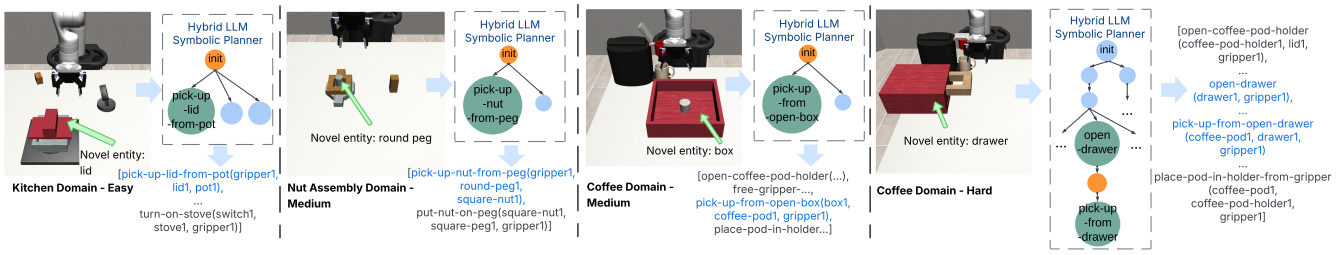

Fig. 2: Problem domains and hybrid LLM-symbolic planner outputs. Domains are ordered by the difficulty of discovering a plannable state via random exploration. Green arrows indicate injected novelties: a lid (Kitchen), a round peg (Nut Assembly), and a drawer or box (Coffee). In the planning graph, green nodes represent states reached via LLM-suggested operators, blue nodes denote existing operators, and orange nodes indicate where the search-ahead algorithm finds a valid plan. Identified missing operators (shown in blue text) include: *pick-up-lid-from-pot* (Kitchen), *pick-up-nut-from-peg* (Nut Assembly), *pick-up-from-open-box* (Coffee-Box), and *open-drawer* and *pick-up-from-open-drawer* (Coffee-Drawer).

D. Prompting Techniques for Reasoning

Wei et al. demonstrate that reasoning can be elicited from LLMs when they emulate the chain-of-thought process shown in their prompts [40]. Kojima et al. point out that similar thought process can be elicited by simply adding "let's think step by step" to the prompt [41]. The self-consistency approach demonstrates that the accuracy of the output can be further improved if multiple chains-of-thoughts are sampled and the most frequent output is chosen [42]. Tree-of-thoughts (ToT) takes this one step further by prompting the LLM to generate multiple next states in a given problem state and evaluate the progress each thought makes. Inspired by the exploratory nature of ToT, we employ self-consistency prompting to sample the most frequently proposed missing operator in each planning state, and we use the symbolic planner as a binary state evaluator that searches ahead and reports whether a plan can be found with the added operator.

III. Preliminaries

A. Symbolic Planning

Given a formal definition of a problem domain $\sigma = \langle \mathscr{E}, \mathscr{F}, \mathscr{S}, \mathscr{O} \rangle$, where $\mathscr{E}$ is a set of typed entities in the environment, $\mathscr{F}$ a set of detectable Boolean predicates applicable to entities, $\mathscr{S}$ a set of states represented as a set of grounded predicates, and $\mathscr{O}$ a set of operators, we seek to find a solution $\mathscr{P} = [o_1, \ldots, o_{|\mathscr{P}|}]$ for the planning task $\mathscr{T} = (\mathscr{E}, \mathscr{F}, \mathscr{O}, s_0, s_g)$. $\mathscr{P}$ is a sequence of grounded operators that when applied transforms the initial state s_0 into the goal state s_g. Each operator $o \in \mathscr{O}$ is defined as the entities it operates on, the preconditions ψ that must be satisfied before it can be applied, and the effects ω that become true once it is applied [1].

B. Neuro-symbolic Planning and Control

Neuro-symbolic planning and control retain the symbolic planning problem formulation above. For each operator $o_i \in \mathscr{P} = [o_1, \ldots, o_{|\mathscr{P}|}]$, a neural policy $\pi_i \in \Pi$ is learned. Each skill π_i is executable when the preconditions ψ_i of the operator are met, and interacts with the environment to realize the operator's effects ω_i. This formulation captures the learning of low-level controls and the abstraction of tasks in high-level planning.

C. Problem Formulation

Given a planning task $\mathscr{T}' = (\mathscr{E}', \mathscr{F}, \mathscr{O}, s_0, s_g)$ after injection of a novel entity e where $\mathscr{E}' = \mathscr{E} \cup e$, obtain an augmented set of operators $\mathscr{O}'$ by adding the missing operators $\mathscr{O}_{missing}$ that can be applied to e so that a plan $\mathscr{P}'$ can be found from the initial state s_0 to the goal state s_g. Furthermore, learn the neural skill π_i associated with each new grounded operator $o_i \in \mathscr{O}_{missing}$. We assume that the available predicates are enough to define the missing operators and the robot is able to detect and categorize the novel object. For example, since the robot can detect the truth value of the predicate *open(container)*, we assume that it can recognize that a drawer is a container type and detect whether the drawer is open after it is injected as a novelty.

IV. The Plan-Learn-Execute Loop

The loop of plan-learn-execute shown in Figure 1 and Algorithm 1 begins by calling the search-and-prompt algorithm of the hybrid planner to solve the missing operator identification problem. The search and prompt algorithm augments the problem domain PDDL with definitions of $O_{missing}$ and returns a plan that contains grounded operators of $O_{missing}$ for which the robot does not know how to execute. The plan-learn-execute algorithm learns the neural control policy for each operator in $O_{missing}$ through the LLM guided sub-goal learning process.

A. Hybrid LLM and Symbolic Planning

The Hybrid LLM Symbolic Planner takes a PDDL domain specifying the available operators $\mathscr{O}$, predicates $\mathscr{F}$, object types and a PDDL problem file specifying the set of entities $\mathscr{E}'$ including the novel entity e, the initial state s_0 and the goal state s_g as grounded predicates over entities. The hybrid planner goes through a breadth-first search-and-prompt process to find a plan to the goal as illustrated by Figure 2 and Algorithm 2. In the search-and-prompt algorithm, the LLM is prompted in each search state to suggest up to one missing

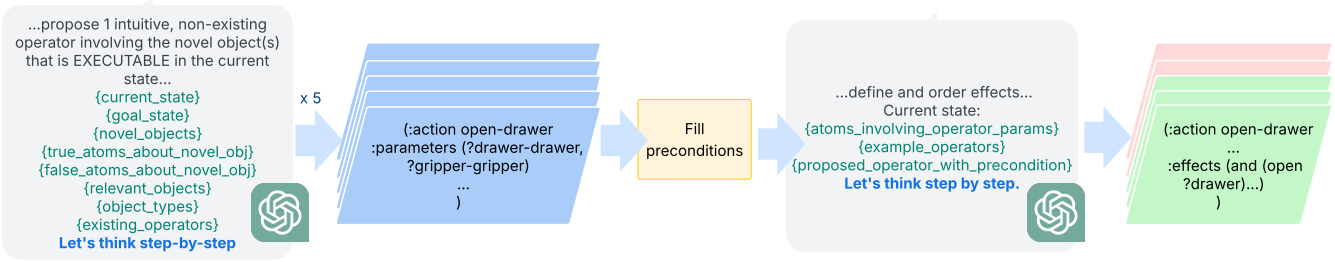

Fig. 3: The Prompt-LLM-for-New-Operator Pipeline. We sample five operator candidates and select the majority [42]. The prompt is dynamically filled in with the current state, goal, and existing operators. The LLM suggests names and parameters for missing operators. Preconditions are automatically filled with grounded predicates involving these parameters, while the LLM defines and orders the effects. For example, if preconditions for *open-drawer* include *not open drawer1* and *not grasped drawer1*, the LLM generates *grasped drawer1* and *open drawer1* as ordered effects, which subsequently serve as sub-goals for the guided learning stage. We observe that errors in effects ordering and operator generation are greatly reduced through self-consistency [42]. Finally, the grounded operator is lifted into a general PDDL definition by mapping specific entities to their variable types (e.g., drawer1 to ?d - drawer), enabling domain-wide generalization.

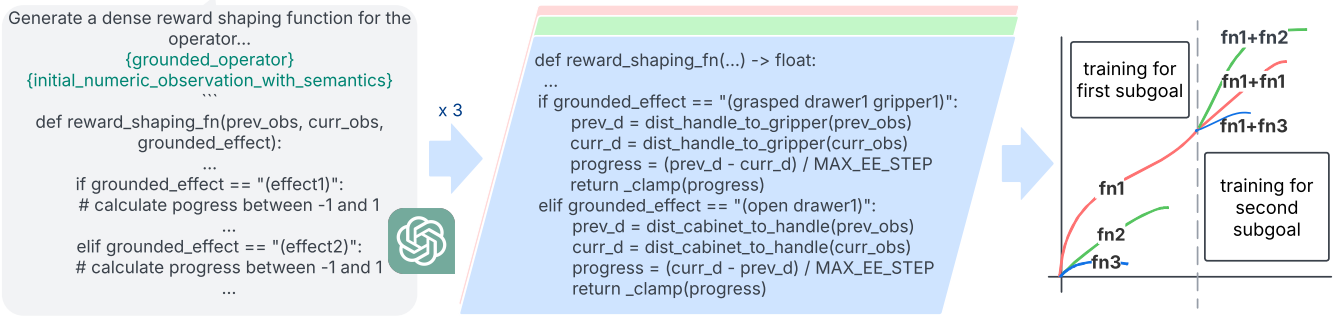

Fig. 4: The LLM Guided Sub-goal Learning Pipeline. Three reward shaping function candidates are sampled from the LLM. We use a temperature of 1.0 to encourage diversity in the samples. We filter out semantically identical candidates by passing in test values to the candidates and eliminating duplicates that output the same reward. The prompt contains the template for a reward shaping function. The definition of the grounded operator and the observation space of the robot are dynamically filled into the prompt. The LLM writes a function to compute a velocity based progress using the relevant observations in the robot's observation space. For example, the function computes the progress for *open drawer1* using the distance between the drawer handle and the cabinet. During each sub-goal's training phase, the reward shaping for the sub-goal is unlocked. The worst performing candidate is periodically eliminated.

executable operator shown in Figure 3. We employ the self-consistency prompting technique to increase the accuracy of the output [42]. To minimize LLM calls, a purely symbolic search-ahead algorithm evaluates whether a plan can already be found from the current state forward with the addition of the suggested operator. The search-ahead algorithm is a breadth-first search that searches for the goal state given the available operators. If no plan can be found, the operators whose preconditions are satisfied in the current state are applied to reach next states, and the reachable next states are added to the search-and-prompt tree. The search-ahead algorithm terminates the search-and-prompt process early if a plan with the currently available operators is found. The search-and-prompt tree grows until a plan is found or the max search depth is reached. We choose GPT-o3 for the planning stage for its advanced reasoning capabilities.

V. LLM Guided Sub-goal Learning

We launch Proximal Policy Optimization (PPO) agents to learn the continuous control for each grounded operator in the plan [43]. Our RL pipeline is implemented using the *stable_baselines3* [44] library. We guide the PPO agents toward each sub-goal via reward shaping functions written by GPT-o4-mini with a temperature of 0.3. We choose GPT-o4-mini instead of GPT-o3 for the function writing due to its fast and cost-efficient code generation. We employ a genetic algorithm inspired approach to filter the candidates by eliminating the non-runnable ones and iteratively eliminating the least successful ones.

Figure 4 illustrates the generation and filtering of reward shaping functions. The robot receives numeric observations (e.g., relative distances between the gripper and task-relevant entities) and binary observations (grounded predicate truth values). These, along with the operator definition and maximum end-effector displacement, are used to prompt the

Algorithm 1: ***Plan-Learn-Execute***

Input: $\mathscr{T}'$, *max_iter*
Output: *goal_achieved*
$iteration \leftarrow 0$
$goal\_achieved \leftarrow$ **false**
while $iteration < max\_iter$ **and** $\neg goal\_achieved$ **do**
  ResetEnvironment()
  $\mathscr{P}' \leftarrow$ **Search-And-Prompt**($\mathscr{T}'$)
  while $|\mathscr{P}'| > 0$ **do**
    $o \leftarrow$ **PopFirst**($\mathscr{P}'$)
    $success, executor \leftarrow$ **ExecuteOperator**(o)
    if $\neg executor$ **then**
      $executor \leftarrow$ **LLMGuidedSubgoalLearning**(o)
      $success, executor \leftarrow$ **ExecuteOperator**(o)
    end if
    if $\neg success$ **then**
      break {restart the main loop}
    end if
  end while
  $iteration \leftarrow iteration + 1$
  $goal\_achieved \leftarrow |\mathscr{P}| = 0$ **and** *success*
end while
return *goal_achieved*

Algorithm 2: ***Search-and-Prompt***

Input: $\mathscr{T}'$, *max_depth*
Output: $\mathscr{O}' = \mathscr{O} \cup \mathscr{O}_{missing}, \mathscr{P}'$
$open\_queue \leftarrow [s_0, \mathscr{O})]$
$closed \leftarrow [(s_0, \mathscr{O})]$
$depth \leftarrow 0$
while $|open\_queue| > 0$ **do**
  $s, \mathscr{O} \leftarrow$ **PopFirst**($open\_queue$)
  $o_{missing} \leftarrow$ **Prompt-LLM-For-New-Operator**($s, \mathscr{O}$)
  $\mathscr{O}' \leftarrow \mathscr{O} \cup o_{missing}$
  $\mathscr{P}' \leftarrow$ **Search-Ahead**(s, $\mathscr{O}'$, max_depth) {search symbolically to reduce LLM calls}
  if $\mathscr{P}' \neq$ None **then**
    return $\mathscr{O}', \mathscr{P}'$
  end if
  if $depth \geq max\_depth$ **then**
    break
  end if
  $\mathscr{S}' \leftarrow$ **Applicable**($s, \mathscr{O}'$)
  for $s' \in \mathscr{S}'$ **do**
    if $s' \notin closed$ **then**
      $open\_queue \leftarrow open\_queue \cup [(s', \mathscr{O}')]$
      $closed \leftarrow closed \cup [(s', \mathscr{O}')]$
    end if
  end for
  $depth \leftarrow depth + 1$
end while
return None, None

LLM to generate three reward function candidates. Each candidate outputs a linear, velocity-based reward in the range $[-1, 1]$ based on the change in numeric observations between timesteps. For example, in the *open-drawer* task (Fig. 4), the LLM-generated code calculates and normalizes the distance change between the cabinet and handle, returning a clamped progress reward towards the *open-drawer* sub-goal.

As effects i.e. sub-goals of an LLM defined operator in Figure 3 are expected to be achieved in order, the sub-goals are trained in phases. As detailed in Algorithm 3, a PPO agent is launched for each reward candidate with sub-goal bonuses increasing ten-fold per stage. Crucially, agents receive no reward for progress beyond the current sub-goal in

Algorithm 3: ***LLM-sub-goal-Reward***

Input: o, prev_num_obs, curr_num_obs, curr_binary_obs
Output: reward
reward $\leftarrow 0$
for $i = 1 \rightarrow$ |o.effects| **do**
  sub-goal_bonus $\leftarrow 10^i$
  max_reward_shaping $\leftarrow 10^{i-1}$
  effect $\leftarrow$ o.effects[i]
  if $\neg$**CurrTrainingPhase**(effect) **then**
    break {avoid giving rewards for locked sub-goals}
  end if
  if CheckEffectSatisfied(effect, curr_binary_obs) **then**
    reward$\leftarrow$ reward + *sub-goal_bonus*
  else
    reward $\leftarrow$ reward + max_reward_shaping $\times$ **LLM-Reward-Shaping**(prev_num_obs, curr_num_obs,effect)
    break {stop as soon as a sub-goal is not satisfied}
  end if
end for
return reward

TABLE I: Hybrid LLM Symbolic Planner vs Operator Discovery (OD)

Domain	Hybrid Successes	OD Successes	Hybrid Time	OD Time
Kitchen	10/10	10/10	75.19 sec	85.51 sec
Nut Assembly	10/10	0/10	120.76 sec	$>$ 7h
Coffee (box)	10/10	0/10	45.26 sec	$>$ 7h
Coffee (drawer)	7/10	0/10	681.83 sec	$>$ 7h

focus. The *CheckEffectSatisfied* function verifies that an effect is satisfied without unintended changes to other entities. Once a phase is complete, rewards for the subsequent sub-goal are unlocked. When an effect is not achieved, the agent receives a progress reward from the LLM-generated function, scaled by a factor *max_reward_shaping* that increases ten-fold per sub-goal. This results in exponentially larger rewards as the agent advances. For instance, in the *grasped-drawer* phase, the agent receives an LLM-shaping reward in $[-1, 1]$ or a 10 point sub-goal bonus. In the subsequent *open-drawer* phase, the shaping reward scales to $[-10, 10]$, with the final sub-goal bonus reaching 100.

Upon completing a training phase, the best-performing candidate and its policy are retained. For the subsequent sub-goal, three new candidates are generated; these reuse the "best" logic from previous phases (e.g., the optimal grasp-drawer-handle snippet) while providing diverse new logic for the current sub-goal (e.g., open-drawer). We currently use a rule-based function with ground-truth state to verify effects to decouple planning performance from perception errors. However, real-robot deployment would interface with a perception module for grounding. Episode horizons start at 100 steps, increasing by 50 per subsequent stage.

VI. Evaluation

We compare our hybrid planner with the Operator Discovery (OD) method. After encountering a planning impasse, many of the existing hybrid planning and RL approaches launch RL agents to discover a solution. As the RL agent interacts with the environment, state changes are tracked. If

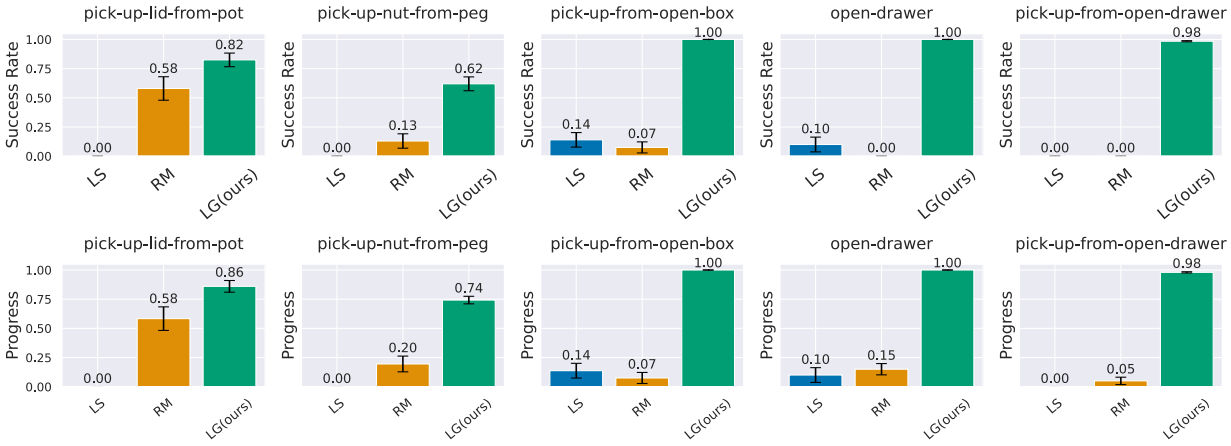

Fig. 5: Comparison of the LLM Guided (LG) Sub-goal Learning with the LEAGUE-Sparse (LS) [18] and Reward Machine (RM) [20] Baselines. We record two metrics: success rate of the operator and progress towards the completion. Progress is the percent of sub-goals achieved. Metrics are averaged across ten seeds with standard error of the mean (SEM).

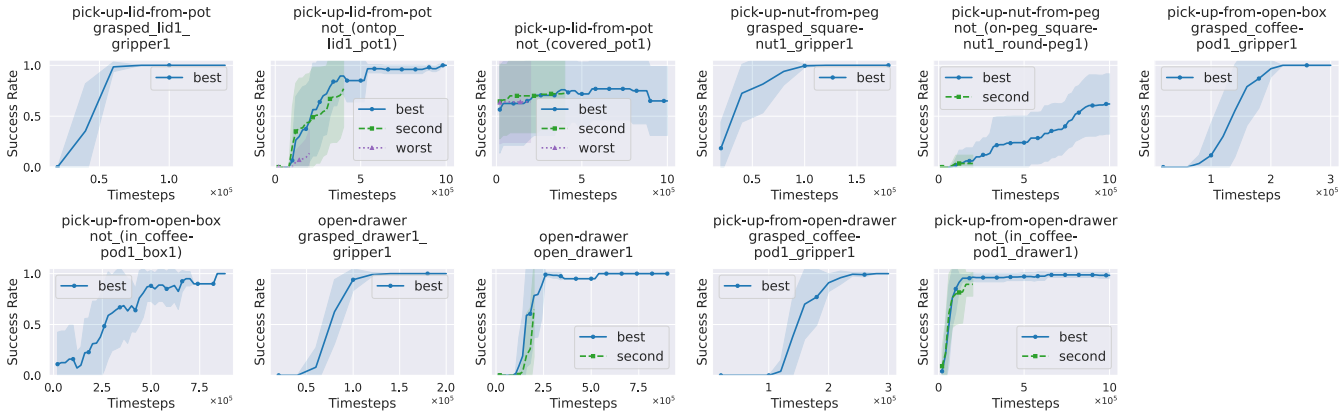

Fig. 6: Reward Function Candidates Evaluation Curves For Each Sub-goal During Training. The worst performing candidate was eliminated at 2×10^5 timestep intervals. Note that some sub-goals did not have three reward function candidates due to them being semantically identical. Refer to Table II for the number of unique reward function candidates per sub-goal.

the RL agent discovers a plannable state, the state change trace can be used to identify new operators. As shown in Figure 2, we evaluate our approach across four domains of increasing difficulty:

- **Kitchen (Easy):** A novel lid blocks the pot. This domain is considered easy because an RL agent can displace the lid through random exploration to discover a plannable state.
- **Nut Assembly (Medium):** A square nut is placed on a novel round peg. Since the robot's existing operators only support tabletop retrieval, it must learn a new pick-up skill for the peg.
- **Coffee-Box (Medium):** The pod spawns inside a novel box. The agent must learn to retrieve the pod from this container rather than the tabletop.
- **Coffee-Drawer (Hard):** The pod is inside a closed drawer. This is the most difficult task, as random exploration is virtually certain to fail at opening the drawer and retrieving the pod.

We launch 10 seeds of Operator Discovery per domain as PPO RL agents. Each agent is capped at 1 million timesteps. We terminate the episode once the agent finds a plannable state. As a comparison, we run our hybrid planner 10 times per domain with GPT-o3 with a temperature of 0.3 to identify the missing operators. We compare our LLM guided sub-goal learning agent with a Reward Machine (RM) baseline that only receives sub-goal bonuses and a LEAGUE-sparse baseline that only receives a reward once all effects are satisfied. We cap the episode length at 100 for the operator learning agent. All experiments are run on a ubuntu 20.04 machine with 24 12th Gen Intel(R) Core(TM) i9-12900K CPUs and a NVIDIA GeForce RTX 4090 GPU. We use the default hyperparameter values for all of our PPO agents.

VII. Discussion

Table I shows that the hybrid planner is able to identify the missing operator(s) in all domains. The OD agent can only consistently discover a plannable state in the easy kitchen domain. This shows that OD agents' discovery of missing operators in continuous domains is purely accidental.

TABLE II: Diversity in Generated Reward Function Candidates ($Temperature = 1$). Values denote the number of semantically unique candidates generated per sub-goal.

Operator	Sub-goals (Number of Variants)
pick-up-lid-from-pot	grasped_lid1_gripper1 (1) not_(ontop_lid1_pot1) (2) not_(covered_pot1) (2)
pick-up-nut-from-peg	grasped_square-nut1_gripper1 (1) not_(on-peg_square-nut1_round-peg1) (2)
pick-up-from-open-box	grasped_coffee-pod1_gripper1 (1) not_(in_coffee-pod1_box1) (1)
open-drawer	grasped_drawer1_gripper1 (1) open_drawer1 (2)
pick-up-from-open-drawer	grasped_coffee-pod1_gripper1 (1) not_(in_coffee-pod1_drawer1) (2)

TABLE III: Wilcoxon signed tests p-values for Success Rate and Progress. Comparisons are made between our LLM-Guided (LG) Subgoal agent and the baselines-the LEAGUE-sparse agent (LS) and the Reward Machine (RM). Values marked with * indicate statistical significance ($p < 0.05$).

Success Rate p-values		
Operator	LS vs LG (Ours)	RM vs LG (Ours)
pick-up-lid-from-pot	0.0010*	0.1094
pick-up-nut-from-peg	0.0010*	0.0107*
pick-up-from-open-box	0.0010*	0.0010*
open-drawer	0.0020*	0.0010*
pick-up-from-open-drawer	0.0010*	0.0010*
Progress p-values		
Operator	LS vs LG (Ours)	RM vs LG (Ours)
pick-up-lid-from-pot	0.0010*	0.1094
pick-up-nut-from-peg	0.0010*	0.0107*
pick-up-from-open-box	0.0010*	0.0010*
open-drawer	0.0020*	0.0010*
pick-up-from-open-drawer	0.0010*	0.0010*

Therefore OD agents are not a reliable solution to the missing operator identification problem. Figure 5 shows that the LLM guided sub-goal learning agent reaches above 90% average success and progress for most operators. Wilcoxon signed-tests show that the LLM guided agents significantly outperform the baselines for most operators ($p < 0.05$) in Table III. Additionally, having multiple reward function candidates mitigates the risks of code errors or poorly performing reward functions over single candidate approaches such as LEAGUE++, as shown in Figure 6.

The results suggest that the LLM generated reward shaping functions are beneficial in guiding the agents towards achieving the sub-goals. The RM agent outperforms the LS agent in the easy domain, showing that sub-goal bonuses guide the agent towards the overall goal if the sub-goals are easily discoverable. While we demonstrate our approach in simulation, the architecture is applicable to real-world robotics. An implementation would involve object grounding through an open-vocabulary perception module such as OWLv2 [45] and real-to-sim to allow the LLM-guided RL agents to learn policies in simulation. Sim-to-real techniques help deploy the learned policies back onto the physical robot.

VIII. Conclusion and Future Work

In conclusion, we show that a hybrid LLM-symbolic planning framework complemented by LLM-guided RL provides a path for novelty adaptation in robotics. We outsource the missing operator identification problem and dense reward shaping to LLMs, leveraging their common sense reasoning and code generation capabilities to circumvent the brittleness of hand-coded planners and the inefficiency of unguided RL. Our evaluation across increasingly difficult domains indicate that LLM-informed symbolic planning and learning can generalize to varied novel objects, and that the generated reward functions accelerate learning and improve final performance. Future work will explore scaling this method to environments with multiple concurrent novelties and missing operators. Additionally, we plan to validate the framework on physical robotic platforms using open-vocabulary perception to evaluate real-world grounding and manipulation. Finally, leveraging vision-language models for autonomous predicate invention remains a promising direction to relax the assumption of predicate completeness and enable the dynamic expansion of both predicates and operators.

References

[1] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson *et al.*, "Pddl—the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[2] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*, 1st ed. Cambridge University Press, 2016. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781139583923/type/book

[3] P. Chatterjee, A. Chapagain, W. Chen, and R. Khardon, "Disprod: differentiable symbolic propagation of distributions for planning," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, ser. IJCAI '23, 2023. [Online]. Available: https://doi.org/10.24963/ijcai.2023/591

[4] S. Goel, Y. Shukla, V. Sarathy, M. Scheutz, and J. Sinapov, "RAPid-learn: A framework for learning to recover for handling novelties in open-world environments," in *2022 IEEE International Conference on Development and Learning (ICDL)*, 2022, pp. 15–22. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9962230

[5] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, and M. Scheutz, "SPOTTER: Extending symbolic planning operators through targeted reinforcement learning," 202. [Online]. Available: http://arxiv.org/abs/2012.13037

[6] E. Gizzi, L. Nair, S. Chernova, and J. Sinapov, "Creative problem solving in artificially intelligent agents: A survey and framework," *Journal of Artificial Intelligence Research*, vol. 75, Nov. 2022. [Online]. Available: http://dx.doi.org/10.1613/jair.1.13864

[7] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," 2023. [Online]. Available: https://arxiv.org/abs/2305.14078

[8] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox, "MimicGen: A data generation system for scalable robot learning using human demonstrations," 2023. [Online]. Available: http://arxiv.org/abs/2310.17596

[9] S. Goel, P. Lymperopoulos, R. Thielstrom, E. Krause, P. Feeney, P. Lorang, S. Schneider, Y. Wei, E. Kildebeck, S. Goss, M. C. Hughes, L. Liu, J. Sinapov, and M. Scheutz, "A neurosymbolic cognitive architecture framework for handling novelties in open worlds," vol. 331, p. 104111, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437022400047X

[10] F. Yang, D. Lyu, B. Liu, and S. Gustafson, "PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making," 2018. [Online]. Available: http://arxiv.org/abs/1804.07779

[11] H. Kokel, A. Manoharan, S. Natarajan, B. Ravindran, and P. Tadepalli, "RePReL: Integrating relational planning and reinforcement learning for effective abstraction," vol. 31, pp. 533–541, 2021. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/16001

[12] K. Acharya, W. Raza, C. Dourado, A. Velasquez, and H. H. Song, "Neurosymbolic reinforcement learning and planning: A survey," vol. 5, no. 5, pp. 1939–1953, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10238788

[13] J. Balloch, Z. Lin, R. Wright, X. Peng, M. Hussain, A. Srinivas, J. Kim, and M. O. Riedl, "Neuro-symbolic world models for adapting to open world novelty," 2023. [Online]. Available: http://arxiv.org/abs/2301.06294

[14] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," vol. 30, pp. 540–550, 2020. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/6750

[15] L. Guan, S. Sreedharan, and S. Kambhampati, "Leveraging approximate symbolic models for reinforcement learning via skill diversity," in *Proceedings of the 39th International Conference on Machine Learning*. PMLR, 2022, pp. 7949–7967, ISSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v162/guan22c.html

[16] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," 2022. [Online]. Available: http://arxiv.org/abs/2206.10680

[17] P. Lorang, S. Goel, Y. Shukla, P. Zips, and M. Scheutz, "A framework for neurosymbolic goal-conditioned continual learning in open world environments," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 12 070–12 077, ISSN: 2153-0866. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10801627

[18] S. Cheng and D. Xu, "League: Guided skill learning and abstraction for long-horizon manipulation," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, p. 6451–6458, Oct. 2023.

[19] P. Lorang, H. Horvath, T. Kietreiber, P. Zips, C. Heitzinger, and M. Scheutz, "Adapting to the "open world": The utility of hybrid hierarchical reinforcement learning and symbolic planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 508–514. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10611594

[20] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 173–208, 2022.

[21] P. Lorang, H. Lu, and M. Scheutz, "Curiosity-driven imagination: Discovering plan operators and learning associated policies for open-world adaptation," 2025. [Online]. Available: http://arxiv.org/abs/2503.04931

[22] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz, "Generalized planning in PDDL domains with pretrained large language models," vol. 38, no. 18, pp. 20 256–20 264, 2024, number: 18. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/30006

[23] 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," 2022. [Online]. Available: http://arxiv.org/abs/2209.11302

[24] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," 2022. [Online]. Available: http://arxiv.org/abs/2201.07207

[25] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can, not as i say: Grounding language in robotic affordances," 2022. [Online]. Available: http://arxiv.org/abs/2204.01691

[26] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman, and B. Ichter, "Grounded decoding: Guiding text generation with grounded models for embodied agents," 2023. [Online]. Available: http://arxiv.org/abs/2303.00855

[27] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," 2023. [Online]. Available: http://arxiv.org/abs/2210.03629

[28] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," 2022. [Online]. Available: http://arxiv.org/abs/2207.05608

[29] G. Dagan, F. Keller, and A. Lascarides, "Dynamic planning with a LLM," 2023. [Online]. Available: http://arxiv.org/abs/2308.06391

[30] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents," 2024. [Online]. Available: http://arxiv.org/abs/2302.01560

[31] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "LLM+p: Empowering large language models with optimal planning proficiency," 2023. [Online]. Available: http://arxiv.org/abs/2304.11477

[32] 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 *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 2023, pp. 8657–8677, ISSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v202/du23f.html

[33] Y. Shukla, W. Gao, V. Sarathy, A. Velasquez, R. Wright, and J. Sinapov, "LgTS: Dynamic task sampling using LLM-generated sub-goals for reinforcement learning agents," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '24. International Foundation for Autonomous Agents and Multiagent Systems, 2024, pp. 1736–1744.

[34] B. Quartey, A. Shah, and G. Konidaris, "Exploiting contextual structure to generate useful auxiliary tasks," 2024. [Online]. Available: http://arxiv.org/abs/2303.05038

[35] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," 2023. [Online]. Available: http://arxiv.org/abs/2303.00001

[36] 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," 2023. [Online]. Available: http://arxiv.org/abs/2309.06687

[37] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2reward: Reward shaping with language models for reinforcement learning," 2024. [Online]. Available: http://arxiv.org/abs/2309.11489

[38] 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," 2024. [Online]. Available: http://arxiv.org/abs/2310.12931

[39] Z. Li, K. Yu, S. Cheng, and D. Xu, "League++: Empowering continual robot learning through guided skill acquisition with large language models," in *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

[40] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: http://arxiv.org/abs/2201.11903

[41] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," 2023. [Online]. Available: http://arxiv.org/abs/2205.11916

[42] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "SELF-CONSISTENCY IMPROVES CHAIN OF THOUGHT REASONING IN LANGUAGE MODELS," 2023.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[44] 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. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[45] M. Minderer, A. Gritsenko, and N. Houlsby, "Scaling open-vocabulary object detection," *Advances in Neural Information Processing Systems*, vol. 36, pp. 72 983–73 007, 2023.