Improving Pre-Trained Vision-Language-Action Policies with Model-Based Search

Cyrus Neary[1,2,3], Omar G. Younis[1], Artur Kuramshin[1,2], Özgür Aslan[1,2], Glen Berseth[1,2]
[1]Mila — Quebec AI Institute, Canada, [2]Université de Montréal, Canada
[3]The University of British Columbia, Canada
cyrus.neary@ubc.ca, {omar.younis, artur.kuramshin, ozgur.aslan, glen.berseth}@mila.quebec

***Abstract*— Pre-trained vision-language-action (VLA) models offer a promising foundation for generalist robot policies, but often produce brittle behaviors or unsafe failures when deployed zero-shot in out-of-distribution scenarios. We present *Vision-Language-Action Planning & Search (VLAPS)*— a novel framework and accompanying algorithms that embed model-based search into the inference procedure of pre-trained VLA policies to improve their performance on robotic tasks. Specifically, our method biases a modified Monte Carlo Tree Search (MCTS) algorithm—run using a model of the target environment—using action priors defined by the VLA policy. By using VLA-derived abstractions and priors in model-based search, VLAPS efficiently explores language-conditioned robotics tasks whose search spaces would otherwise be intractably large. Conversely, by integrating model-based search with the VLA policy's inference procedure, VLAPS yields behaviors that are more performant than those obtained by directly following the VLA policy's action predictions. VLAPS offers a principled framework to: i) control test-time compute in VLA models, ii) leverage a priori knowledge of the robotic environment, and iii) integrate established planning and reinforcement-learning techniques into the VLA inference process. Across all experiments, VLAPS significantly outperforms VLA-only baselines on language-specified tasks that would otherwise be intractable for uninformed search algorithms, increasing success rates by as much as 67 percentage points.**

***Index Terms*— Generalist robot policies, vision-language-action models, inference-time compute, model-based search, Monte Carlo tree search, robot learning**

I. Introduction

Large vision-language-action (VLA) models—pre-trained on internet-scale vision and language data, as well as on diverse datasets of robot demonstrations—offer promise as a key building block for generalist robot policies that autonomously solve complex, open-ended tasks. Their natural-language interface enables intuitive task specification, and their multimodal reasoning capabilities support flexible and rapid adaptation to new tasks and environments. However, current VLAs are unable to reason over the potential consequences of their actions, and instead rely purely on imitating observed behaviors from their training data. Myopically following VLA action outputs can thus result in brittle behaviors and unsafe failures, impeding the adoption of such models in many robotics applications.

On the other hand, model-based planning algorithms explicitly reason over future outcomes, but typically rely on carefully handcrafted heuristics to address the combinatorial explosion of the search space. Indeed, many robotics tasks are naturally characterized by sparse rewards in large state-action spaces, making direct search without a well-designed heuristic infeasible. It remains challenging to design heuristics that generalize across diverse robotic tasks—particularly those specified in natural language and deployed in complex, cluttered environments.

Towards developing generalist robot policies that are flexible, robust, and capable of reasoning over long horizons, we introduce *vision-language-action planning and search (VLAPS)*. VLAPS integrates model-based search with a pre-trained VLA policy, to leverage the core strengths of both approaches to problem solving. Figure 1 illustrates the approach: VLAPS uses a pre-trained VLA to bias a Monte Carlo tree search (MCTS)-inspired algorithm [1] through several distinct mechanisms. It begins by using the VLA policy to sample contextually-relevant collections of temporally-abstract *action chunks* at every node of the search tree, automatically refining the large action spaces typical of robotics problems into tractable, task-relevant spaces for search. It then leverages VLA-derived priors to bias node selection during tree traversal, further focusing the search on promising behaviors suggested by the VLA policy. By doing so, VLAPS efficiently explores language-specified robotics tasks whose search spaces would otherwise be intractably large. Conversely, by integrating model-based search with the VLA model's inference procedure, VLAPS yields policies that are more performant than those obtained by directly following the VLA's action predictions.

We demonstrate the benefits of the proposed approach in *LIBERO* [2]—a simulated suite of language-specified robotic manipulation tasks. By integrating a VLA policy with MCTS-style search, VLAPS outperforms the underlying VLA, with task success rates improving by up to 67 percentage points. We also find that VLAPS' performance increases alongside that of the underlying VLA, and that it adaptively allocates more search to difficult instances. Moreover, it can boost the performance of relatively small VLAs to levels that match much larger, sate-of-the-art models.

II. Related Work

A. VLA Models as Generalist Robot Policies

Recent work has demonstrated that large, multimodal transformers can map natural language instructions and visual observations directly to robot actions, enabling the development of generalist robot policies that flexibly adapt

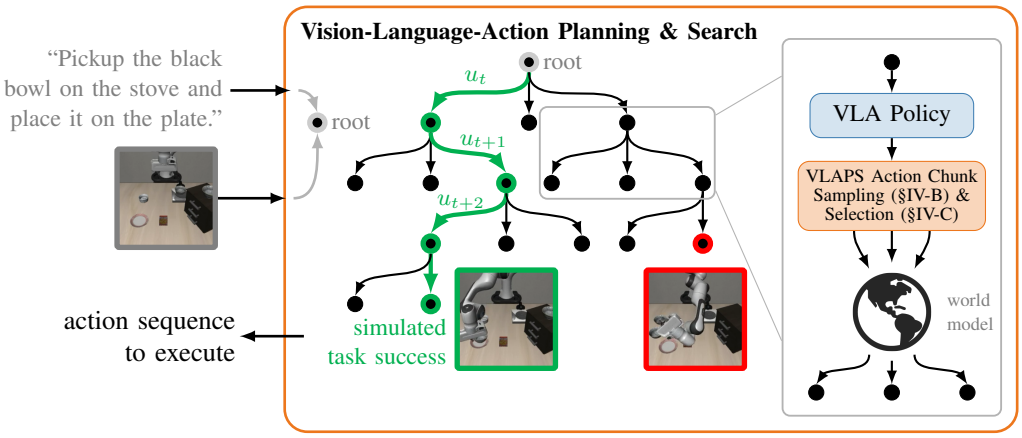

Fig. 1. The proposed vision-language-action planning and search (VLAPS) framework. At each decision-making point in the task, VLAPS uses a world model to conduct a search over temporally-abstract action chunks. The method uses a pre-trained VLA policy to guide the expansion (§IV-B) and selection (§IV-C) procedures of the search. Once the search finds a node that completes the task, or a compute budget is exhausted, VLAPS returns the most promising action sequence to be executed in the environment, before repeating the process with the next observation.

to diverse tasks and environments [3]–[7]. To support this paradigm, substantial effort has recently been devoted to collecting large-scale robot datasets [8], [9], evaluating robot policies in the real world [10], and to advancing VLA architectures and training strategies for robot control [11]–[13]. These works focus primarily on developing effective VLA architectures and on leveraging internet-scale data to train generalist robot policies via behavior cloning. By contrast, VLAPS improves the performance of pre-trained VLA policies by integrating them with model-based search.

B. Additional Test-Time Compute Improves the Capabilities of Language Models

Methods to increase test-time compute—such as chain-of-thought prompting [14]–[16], self-consistency sampling [17], and MCTS during inference [18], [19]—have recently demonstrated substantial performance gains in language and reasoning tasks by dynamically allocating additional model evaluations at inference time [20]–[22]. VLAPS brings this paradigm to robotics and embodied reasoning problems by providing a mechanism to explicitly inject prior modeling knowledge into the VLA-driven sequential decision-making loop. This knowledge may be provided by a high-fidelity simulator [2], [23], [24], a learned world model [25]–[27], or a video-prediction model [28]–[32]. In this work, to isolate the core impacts of our contribution, we assume access to an accurate simulator and leave exploration of imperfect or learned models for future work.

C. Model-Based Search as Test-Time Improvement for Learned Policies

The *AlphaZero* and *MuZero* family of algorithms [27], [33]–[35] couple deep reinforcement learning (RL) with online MCTS to achieve superhuman play in Go, Chess, Shogi, and Atari. In [36], Bertsekas draws parallels between *AlphaZero* and model-predictive control (MPC), and shows that even a single step of test-time model-based lookahead optimization can correct for substantial errors in a learned value function. Further exemplifying this point, RL algorithms that employ MPC over learned latent models of dynamics and rewards demonstrate strong performance in a wide variety of complex tasks [25], [26], [37], [38]. VLAPS brings this methodology to generalist robot policies, using model-based search at test time to correct for the errors of pre-trained VLA models.

D. Planning with Language-Conditioned Models for Robot Decision Making

Several recent methods combine language-conditioned models with search or planning to improve robotic decision-making. Many of these works use large language models (LLMs) or vision language models (VLMs) to generate high-level plans that are then executed by motion planners [39], by pre-defined skill libraries [40], through MCTS over discrete object rearrangement primitives [41], or by a VLA [42], [43]. Meanwhile, the authors of [30] couple a VLM with an environment model to predict single-path rollouts and to revise the VLM's text-based action suggestions. While these works either output linear plans over text-based descriptions of actions, or perform search over pre-defined primitives, VLAPS employs a branching search over action abstractions derived from a VLA, directly generating robot-executable action sequences and thus removing the need for additional low-level motion planners or goal-conditioned policies.

III. Background

A. Task Environments

Formally, we represent the robot's environment as a Markov decision process (MDP) $\mathcal{M} = (S, A, T, R)$. Here, S is the set of environment states $s \in S$ (including, for example, object locations, proprioceptive sensor measurements, and images I of the environment), A is the set of raw robot actions $a \in A$ (e.g., end-effector position and gripper commands), and $R : S \to \mathbb{R}$ is a reward function encoding the robot's task. In this work, we model the environment dynamics using a deterministic transition

function $T : S \times A \to S$ which maps a state $s_t \in S$ and action $a_t \in A$ at time t to the corresponding state at the next timestep $s_{t+1} \in S$.

B. Robotic Tasks

We formalize the robot's task as a tuple $\mathcal{T} = (L_\mathcal{T}, S_\mathcal{T})$, where $L_\mathcal{T}$ is a natural-language instruction such as *"Place the orange juice in the basket"* and $S_\mathcal{T} \subseteq S$ is the set of all environment states in which this task has been completed. The sparse reward function $R_\mathcal{T}$ corresponding to a particular task $\mathcal{T}$ is thus defined to return $R_\mathcal{T}(s) = 1$ when $s \in S_\mathcal{T}$, and $R_\mathcal{T}(s) = 0$ otherwise. Our objective in this work is to create a policy that, given information on the current state s and a natural language instructions $L_\mathcal{T}$ for an arbitrary task $\mathcal{T}$, outputs a sequence of actions that efficiently leads to a goal state $s \in S_\mathcal{T}$.

C. Vision-Language-Action Models

A promising approach for designing such policies is to leverage VLA models—large-scale transformer networks trained to directly predict action sequences from multimodal observation histories and natural-language task instructions. Such models are often trained by fine-tuning a pre-trained VLM to either autoregressively predict actions or to guide a diffusion process for sampling them. In order to learn efficiently, VLA models often discretize the continuous action space into a sequence of discrete tokens, a process known as action tokenization. This allows robot actions to be represented in an identical format to language tokens, thus enabling the application of architectures and training techniques originally developed for text modeling.

D. Monte Carlo Tree Search

MCTS is an algorithm for sequential decision-making that incrementally builds a search tree by simulating possible future trajectories, while automatically balancing exploration of uncertain actions with exploitation of actions whose estimated values are high [1]. MCTS is well-suited for problems with large action spaces and sparse rewards, as it focuses computational effort towards the most promising and underexplored regions of the search space. MCTS proceeds through four main phases at each decision point:

Selection: Starting from the root node v_0, recursively select child nodes according to a selection policy (e.g., PUCT [34]) until reaching a leaf node.

Expansion: If the selected leaf node is not terminal, expand the tree by adding one or more child nodes corresponding to the leaf node's available actions.

Simulation: From each newly added child node, use a *rollout policy* $\pi_{rollout}$ to simulate a trajectory and sample Monte Carlo estimates of the expected outcome.

Backpropagation: Update the value estimates along the path from the expanded node to the root node based on the simulation results, which will in turn be used during the next selection phase.

IV. Vision-Language-Action Planning & Search

We now present VLAPS, which integrates MCTS-inspired search with a pre-trained VLA policy. Given access to a pre-trained VLA policy and a *world model* $\widehat{\mathcal{M}}$ that may be used to simulate the true operating environment $\mathcal{M}$, VLAPS proceeds as follows.

A. An Overview of the VLAPS Search Procedure.

Figure 1 illustrates an overview of the approach, while Figure 2 details the node expansion and action selection procedures. At each decision point t, VLAPS begins by instantiating a search tree with a root node v_0 containing information on the current state $s_t \in S$ from the true environment $\mathcal{M}$. It then searches the tree from this root node using a VLA-guided variant of the PUCT selection criterion (§IV-C), until it reaches a leaf node v corresponding to a potential future state that has not yet been expanded. Once such a leaf node is reached, the method expands the tree by sampling a collection of action sequences from a prior distribution defined by the VLA policy (§IV-B), and simulating them within the world model $\widehat{\mathcal{M}}$ to generate corresponding child nodes v'. From each such expanded child node, VLAPS then uses the model $\widehat{\mathcal{M}}$ to simulate rollouts of the VLA policy until either the task is complete or a maximum horizon length is reached. VLAPS iteratively repeats this process to expand the tree until either a goal state $\widehat{s} \in S_\mathcal{T}$ is reached, or until a maximum number of MCTS steps has been exhausted. In the former case, the algorithm returns the sequence of actions that reach the encountered goal state. Otherwise, it returns the most frequently-selected action sequence from the root node, and repeats the search process from the resulting next state.

We emphasize that our approach leverages the VLA policy to address several challenges faced by search algorithms in our language-conditioned robotics task setting:

Refining the search space (§IV-B). Directly searching over all possible robot actions is infeasible, even for algorithms like MCTS that are specifically designed to explore large search spaces. We accordingly present a method to refine the search space through VLA-directed sampling.

Biasing the MCTS selection phase (§IV-C). Even while operating within this simplified search space, uniform random search may be computationally expensive. We thus introduce

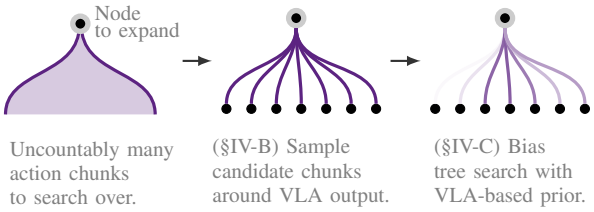

Fig. 2. VLA-informed node expansion and action selection. Whenever a new node is encountered, VLAPS uses the VLA policy to sample a discrete set of candidate action chunks to search over (IV-B). Throughout the remaining search, VLAPS biases action chunk selection towards chunks similar to those output by the VLA (IV-C).

a method to bias the search towards actions favored by the VLA policy.

B. Automatically Defining Task-Oriented, Tractable Search Spaces

The state and action spaces of robotics environments are typically high-dimensional and continuous, rendering the direct application of MCTS—which is designed to sample from a finite set of possible transitions between tree nodes—challenging. Furthermore, the size of the search tree grows exponentially in its depth, which poses a challenge for long-horizon tasks, particularly when considering decision-making timesteps that align with the short timescales of the robot's primitive actions $a \in A$. To address these challenges, existing methods perform search over pre-defined subtasks or over symbolically-defined objectives [41], as opposed to over individual actions. However, such subtasks must typically be hand-designed, limiting their adaptation to new problems.

Towards addressing these challenges, VLAPS performs its search over sets of temporally-abstract *action chunks* [44] $u_t = (a_t, \ldots, a_{t+H}) \in U$ that are flexibly defined at runtime by the outputs of the VLA policy. By leveraging the VLA policy to generate candidate action chunks at each state of the tree search, VLAPS automatically constructs a task-relevant, state-dependent, and tractable search space. This design also enables user control over the chunk horizon H, which governs the frequency at which the VLA must be queried during search.

1) Defining a Finite Library of Candidate Action Chunks: We identify the space of primitive actions A with a subset of $\mathbb{R}^n$, and the space of action chunks U with a subset of $\mathbb{R}^{H \times n}$, where n is the dimension of the action space and H is the length of each chunk. Relevant action chunks may, for example, consist of a short trajectory that executes a coherent robot motion. Rather than conducting search over the entirety of $\mathbb{R}^{H \times n}$, VLAPS restricts planning to a finite set $\Phi \subseteq U$ of candidate action chunks. Although this library is finite, it must be sufficiently large and expressive to support solutions across a diverse range of robotic tasks. In our experiments, we construct Φ by sampling thousands of action chunks from a pre-trained VLA across a suite of tasks. Alternatively, one could define Φ as the collection of all action sequences of length H appearing in the large-scale demonstration datasets used to train the VLA itself. By constructing the candidate library from demonstrations, we restrict the search to behaviorally-relevant regions of the action space, which we hypothesize may improve the feasibility and reliability of the selected actions when deployed on physical robots.

2) Sampling Contextually-Relevant Action Chunk Subsets: Although the library Φ of action chunks is finite, it is still too large to search exhaustively from each tree node during planning. Indeed, many candidate action chunks will likely be irrelevant at any given decision point during a task (e.g., closing the gripper will not be a useful action chunk to consider when the robot's end effector is several feet away from the object it is attempting to grasp). To further refine the search space, we thus use the VLA policy to define a sampling distribution $\beta_\Phi(\cdot|I_t, L_\mathcal{T})$ over action chunks $u \in \Phi$, as in (1) below. Recall that I_t denotes an image observation of the environment and $L_\mathcal{T}$ denotes a natural language task instruction.

To expand a leaf node v during search, VLAPS draws k samples from β_Φ to propose a collection $\Phi_v \triangleq \{u^i | u^i \sim \beta_\Phi(\cdot|I_t, L_\mathcal{T})\}_{i=1}^k$ of candidate action chunks. By reducing the branching factor from $|\Phi|$ to k, these action chunks constrain the local action space at the current node v and serve as the sole candidates considered for tree expansion. Once sampled, they remain fixed for node v for the duration of the planning episode and are reused throughout the search process until a new episode begins and a fresh tree is constructed.

The distribution $\beta_\Phi(\cdot|I_t, L_\mathcal{T})$ uses the VLA model to bias sampling towards contextually-relevant action chunks. As a result, action chunks that are well suited to the current stage of the task are assigned higher probability, enabling a targeted and situation-aware exploration of the action space. More specifically, given an image observation I_t and a language instruction $L_\mathcal{T}$, let $u^{vla}(I_t, L_\mathcal{T}) \sim P_{vla}(I_t, L_\mathcal{T})$ be the action chunk sampled from the VLA model. We define the sampling distribution $\beta_\Phi(\cdot|I_t, L_\mathcal{T})$ for every $u^i \in \Phi$ to be centered around u^{vla} as

$$\beta_\Phi(u^i|I_t, L_\mathcal{T}) = (1-\epsilon)\frac{\exp\big(-\alpha * \rho(u^i, u^{vla})\big)}{\sum_{j=1}^{|\Phi|}\exp\big(-\alpha * \rho(u^j, u^{vla})\big)} + \frac{\epsilon}{|\Phi|}. \tag{1}$$

Here, $\rho(\cdot,\cdot)$ is a distance metric between action chunks; $\alpha \in \mathbb{R}_+$ is an inverse temperature parameter that controls how sharply the distribution favors action chunks similar to the VLA's output; and $\epsilon \in [0, 1]$ governs epsilon-uniform exploration, ensuring that all elements $u^i \in \Phi$ are assigned nonzero probability. While many choices of distance metric ρ are possible, we use the Euclidean distance between flattened action chunks: $\rho(u^1, u^2) \triangleq ||u^1_{flat} - u^2_{flat}||_2$, where each action chunk is reshaped into a vector by flattening along its temporal dimension.

We note that this form of sampling-based action space reduction is closely related to the framework proposed in [45], which formally studies the impact of action subset sampling on policy improvement and evaluation algorithms in large and complex action spaces.

C. Guiding Tree Traversal with a VLA-Based Prior Policy

Having used the VLA policy to define β_Φ, and subsequently to define each node's set Φ_v of action chunks during *expansion*, we now turn to the *selection* phase of the search. In particular, VLAPS further uses the VLA policy to define a prior distribution ψ_{Φ_v} that influences the search's exploration, biasing the tree traversal toward promising subtrees preferred by the VLA policy. Intuitively, while §IV-B uses the VLA policy to constrain each node's action chunk set Φ_v during node expansion, here we use it to bias exploration among those previously-pruned sets of action chunks.

Specifically, we use a policy-guided tree search that omits value estimates, i.e., PUCT [34] with $Q \equiv 0$. At each

selection step of tree traversal, VLAPS chooses the next node v' by maximizing the score (2) over the node-specific candidate action chunks $u^i \in \Phi_v$ sampled from β_Φ.

$$\text{SCORE}(v, u^i) = \psi_{\Phi_v}(u^i | I_t, L_\mathcal{T}) \frac{\sqrt{N(v, u^i)}}{1 + N(v, u^i)} \quad (2)$$

The resulting selection rule allocates visits approximately in proportion to a prior distribution $\psi_{\Phi_v}(u^i | I_t, L_\mathcal{T})$ derived from VLA samples, while ensuring coverage via a node-action visitation count term $N(v, u^i)$. We define $\psi_{\Phi_v}(u^i | I_t, L_\mathcal{T})$ as a softmax-style distribution centered on the output of the VLA model, similar to the construction of β_Φ in (1).

We note that value function estimates—which might be obtained from rollout returns as in classical MCTS [1], observation-conditioned proxies from VLMs [46], or hybrids that combine learned critics and rollouts [33]—could additionally be included in (2) to bias search towards promising branches. However, learning accurate value function estimates for generalist, multi-task robot policies remains an open area of research. In this work we choose to omit such value estimates to isolates the effect of our VLA-based prior on exploration, and to avoid reliance on potentially erroneous critics; we terminate the search upon encountering a simulated success. We find this variant of tree traversal to be well-suited to the studied sparse-reward setting with strong VLA-based priors.

V. Experimental Results

To demonstrate the key benefits of VLAPS, we compare it to pre-trained VLA models on simulated vision-and-language-conditioned robotic manipulation tasks from *Libero* [2]. Project code, datasets, and VLA model checkpoints are publicly available [1].

A. Experimental Procedures

1) Evaluation Tasks and the Baseline VLA Model: We evaluate our method on tasks drawn from all five of the Libero task suites—*Libero-Spatial*, *Libero-Goal*, *Libero-Object*, *Libero-90*, and *Libero-10*—and we compare VLAPS' performance against a baseline VLA-only policy. We note that this VLA-only baseline reflects the typical deployment of VLA models in the literature.

For both VLAPS and the VLA-only baseline policy, we use *Octo* [47] as the underlying VLA model. More specifically, we finetune *Octo-base-1.5* on the Libero dataset [2] to predict end-effector pose and gripper commands from language instructions, a 256x256 fixed camera image, and a 128x128 wrist camera image. The finetuning dataset comprises demonstrations from each of the Libero task suites, and we process the demonstrations to filter out all "no-op" actions, similarly to as in [11].

[1] Code: github.com/cyrusneary/vlaps.
Checkpoints: huggingface.co/cyrusneary/vlaps_v1_octo_libero_finetuned.
Datasets: huggingface.co/datasets/real-lab/libero_filtered_noops_rlds_dataset.

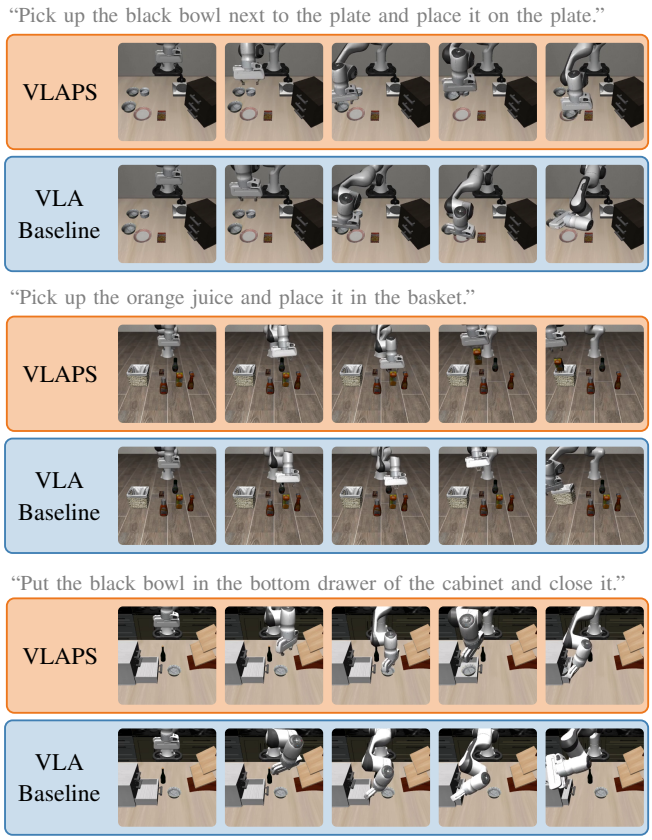

Fig. 3. Illustrative examples where VLAPS succeeds while the underlying VLA policy fails. The VLA policy is *Octo* finetuned for 50k steps on Libero demonstration data. In the top two tasks, the VLA policy fails to grasp the target object, which prevents successful task completion, whereas VLAPS successfully completes the pick-and-place. In the bottom task, the VLA policy prematurely closes the drawer before retrieving the bowl, while VLAPS correctly places the bowl inside before closing it.

2) VLAPS Implementation: All experiments were conducted on an Nvidia A100 GPU. We used 300 Monte Carlo samples per search iteration, expanded $k = 10$ children per node, and allowed simulated rollouts of up to 300 steps. Action chunks had a horizon of $H = 4$, consistent with the Octo training setup, and the maximum tree search depth was set to 100. The candidate library of action chunks Φ was constructed by aggregating action sequences from successful runs under the VLA-only policy, normalizing these collections of action chunks per dimension, and applying K-Medoids clustering to extract a set of 2000 representative action chunk prototypes. To define the action chunk distributions $\beta_\Phi(\cdot)$ and $\psi_{\Phi_v}(\cdot)$, we used a softmax distribution with inverse temperature parameters $\alpha_{\beta_\Phi} = 10.0$ for action-chunk sampling and $\alpha_{\psi_{\Phi_v}} = 5.0$ for the VLA-defined prior policy, with uniform exploration probabilities $\epsilon_{\Phi_\beta} = \epsilon_{\psi_{\Phi_v}} = 0.1$. The above hyperparameter values were chosen to balance exploration with runtime and were held constant across all experiments.

While VLAPS's search procedure could, in principle, be allowed to continue indefinitely, we limit the algorithm's maximum wall-clock execution time to 600 seconds per task to ensure that all experiments can be run in practice. That

is, if VLAPS has not successfully completed a task within 600 seconds, the attempt is considered a failure. This timeout is not intended to reflect a practical deployment constraint, but rather to make large-scale evaluation tractable and to demonstrate the performance improvements that VLAPS can achieve even under finite compute budgets.

3) Effect of Base VLA Quality on VLAPS Performance and Search Efficiency: To examine the relationship between VLAPS' performance and that of the underlying VLA policy, we use the finetuning procedure as a controllable proxy for model quality. That is, we save model checkpoints at $10\text{k}, 50\text{k}, 100\text{k}, 150\text{k}$, and 200k gradient steps during finetuning, and evaluate VLAPS using each checkpoint. Figure 4 illustrates VLAPS' resulting performance values, alongside the performance of the same models when deployed as standalone VLA-only policies. More specifically, for each model checkpoint the figure shows the method's success rate over a total of 1000 tasks drawn from all five Libero task suites, as well as the average algorithm runtime for each successful task evaluation. Table I further elaborates on these results, detailing performance metrics of the evaluated checkpoints separately on each of the various Libero task suites. From these results, we observe that VLAPS consistently outperforms the VLA-only baseline, that these gains remain substantial even when the base VLA has a very low success rate, and that as the base model's quality improves, VLAPS's search procedure becomes markedly faster. We also observe that VLAPS is capable of boosting the performance of small VLA policies to match that of much larger, state-of-the-art models. We discuss these points in more detail in §V-B.

B. Experimental Results and Discussion

1) VLAPS Provides Strict Improvements Over VLA-Only Baseline Policies in All Experiments: The bar chart on the top of Figure 4 illustrates the consistent improvements to task success rate that VLAPS enjoys over the VLA-only baseline method. These improvements are not only consistent—occurring for every tested checkpoint of the base model—but also substantial: VLAPS boosts overall task success rate by 42% for the VLA finetuned for $50k$ steps, and by at least 20% for each of the $100k$-, $150k$-, and $200k$-step checkpoints.

Figure 3 illustrates several example tasks where VLAPS succeeds while the underlying VLA policy fails. When examining such individual tasks, we qualitatively observe that VLAPS more frequently avoids small errors—such as dropping an object or moving the robot into uncommon states not well represented in the demonstration data—that can derail an entire trial.

2) VLAPS Yields the Largest Relative Gains when Augmenting VLA-Only Policies with Low Baseline Success Rates: When examining the results of the $50k$-step model checkpoint in Table I, we observe that VLAPS boosts the VLA's success rate from 6% to 73% on the *Libero-Object* task suite. For this same model checkpoint, VLAPS also increases performance from 34% to 97% and from 50% to 86% on the Libero-Spatial and Libero-Goal task suites,

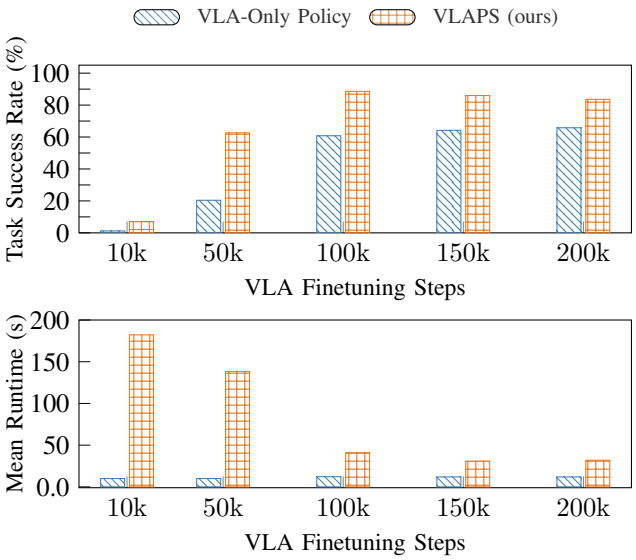

Fig. 4. Performance of VLAPS and the VLA-only policy in Libero, as a function of the number of finetuning steps of the underlying VLA model. Top: Task success rate. Bottom: Mean algorithm runtime to complete each task. Runtimes are reported only for successful task evaluations. Failed VLAPS evaluations consistently reach the 600s search timeout. For each VLA checkpoint, both VLAPS and the VLA-only policy are evaluated on 1000 total tasks, drawn from five Libero task suites: Libero-Spatial, Libero-Goal, Libero-Object, Libero-10, and Libero-90. Each suite contributes 10 distinct tasks, each of which we test from ten different initial conditions.

respectively. These results indicate that VLAPS' planning procedure can substantially increase the performance of less specialized models without additional finetuning, even when the quality of the initial policy is poor. However, the $10k$-step checkpoint results highlight the limits of this effectiveness: when the base VLA policy fails entirely, VLAPS does not succeed within the allocated search time.

3) VLAPS Automatically Allocates More Time to Search When the Base Policy Fails: We observe from the bar chart on the right of Figure 4 that as the performance of the underlying VLA policy improves, the average time VLAPS spends on search decreases sharply. Intuitively, this occurs because when the VLA policy alone can solve a task, VLAPS will return the VLA-suggested solution very quickly. It is only when the base policy fails that VLAPS expands its search tree, effectively allocating additional compute to find an alternate, successful trajectory. We remark that the model checkpoints for which VLAPS spends the most time searching (on average) are exactly those for which base policy performs poorly—and where search yields the largest relative performance gains.

We note that while in this work we focus on finetuning as a controllable proxy for policy quality, we expect that the observed trends will generalize beyond this specific setting. For example, VLAPS is likely to benefit from improved task success rates and reduced search times as more capable VLA policies are developed—whether through alternative architectures, training datasets, or observation spaces—not only through finetuning on task-specific data.

TABLE I

TASK SUCCESS RATE AND MEAN ALGORITHM RUNTIME ACROSS LIBERO TASK SUITES AND VLA FINETUNING CHECKPOINTS. EACH REPORTED SUCCESS RATE IS COMPUTED OVER 100 SEPARATE TRIALS CONSISTING OF 10 DISTINCT TASKS AND 10 DIFFERENT INITIAL CONDITIONS PER TASK. RUNTIMES ARE REPORTED ONLY FOR SUCCESSFUL TASK EVALUATIONS. FAILED VLAPS EVALUATIONS CONSISTENTLY REACH THE 600S SEARCH TIMEOUT.

Task Suite	ckpt.	Success Rate (%)		Mean Runtime (s)	
		VLA	VLAPS	VLA	VLAPS
Spatial	10k	0	0	N/A	N/A
	50k	34	97	8.9	71.9
	100k	83	99	8.5	17.9
	150k	87	97	8.0	12.4
	200k	86	98	7.9	13.5
Goal	10k	3	24	6.8	136.3
	50k	50	86	6.7	54.3
	100k	87	94	6.5	19.8
	150k	87	95	6.6	18.9
	200k	91	91	6.6	15.3
Object	10k	0	2	N/A	289.0
	50k	6	73	9.0	144.8
	100k	32	82	7.9	41.8
	150k	22	63	8.4	38.8
	200k	19	54	9.1	39.8
90	10k	3	9	13.2	121.2
	50k	12	51	15.3	147.3
	100k	65	94	14.7	37.8
	150k	69	94	14.5	36.5
	200k	70	91	15.1	32.6
10	10k	0	0	N/A	N/A
	50k	0	6	N/A	273.8
	100k	37	74	23.3	87.5
	150k	56	81	22.1	47.8
	200k	63	84	20.6	58.3

4) VLAPS Boosts Small Model Performance to that of Much Larger, State-of-the-Art Models: Table II compares the task success rates of Octo, VLAPS (using Octo as the base VLA policy), and π_0-FAST [13]. Both Octo and π_0-FAST were finetuned on demonstrations from the four Libero task suites that are included in the table.

In comparison with π_0-FAST—a state-of-the-art, 3.3 billion parameter VLA model finetuned from a PaliGemma VLM [48] backbone—Octo is relatively small, containing only 93 million parameters. While Octo alone achieves relatively low performance, VLAPS boosts this performance to a level that is comparable with the much larger π_0-FAST. This highlights VLAPS as an effective method to achieve large-model performance from smaller, more compact VLAs.

5) The Proposed VLA-Driven Action Sampling Enables Efficient Search That Would Otherwise Be Intractable: Finally, we note that without the context-aware refinement from §IV-B, the branching factor of search is $b = |\Phi| \approx 2000$. Across our experiments, we observed typical search depths on the order of $d = 100$ before receiving a sparse reward for task completion. A full m-ary tree of height d has $\frac{b^{d+1}-1}{b-1} \approx b^d$ nodes; with $b = 2000$ and $d = 100$ this would result in 2000^{100} total nodes, which clearly renders direct search intractable. Meanwhile, uniform sampling would require on the order of b^d trials in expectation to sample a successful trajectory, which is far beyond any reasonable compute budget. By contrast, VLAPS focuses search on trajectories that are similar to those initially suggested by the VLA policy, and consistently finishes search within the finite compute budgets described in §V-A.

TABLE II

TASK SUCCESS RATES (%) ON LIBERO TASK SUITES. WITH VLAPS, OCTO (97M) ACHIEVES COMPARABLE PERFORMANCE TO π_0-FAST (3.3B), A MUCH LARGER STATE-OF-THE-ART MODEL. WITHOUT VLAPS, OCTO PERFORMS RELATIVELY POORLY BY COMPARISON.

Task Suite	Octo (93M Params)	Octo + VLAPS (Ours)	π_0-FAST (3.3B Params)
Spatial	83	99	96
Goal	87	94	96
Object	32	82	99
Libero-10	37	74	71

VI. LIMITATIONS AND FUTURE WORK

The proposed approach requires access to a world model to simulate sequences of future outcomes. This model may be a high-fidelity robot simulator, a learned dynamics model coupled with a visual rendering engine, or a video prediction model. We expect that frequent feedback from observations could help mitigate any sim-to-real errors when VLAPS is used in a receding-horizon fashion. In this work, however, we focus on the core challenge of integrating model-based search with pre-trained VLA policies, assuming an accurate model is available. Developing methods to jointly learn a world model and use it for planning—similar to *MuZero* [27]—as well as studying the impact of model-environment mismatch, are important directions for future work.

VLAPS improves the performance of pre-trained VLA policies without requiring additional training. However, it incurs additional calls to the VLA model at test time during tree search, and the long inference times of current VLA models can thus lead to increased planning latency. However, optimizations to the algorithm's implementation can help to mitigate these challenges in practice. For instance, VLA queries can be queued and processed in batches, while the expansion and rollout phases of the search algorithm can also be executed in parallel. Additionally, the branching factor and maximum tree depth are configurable parameters that provide direct control over test-time computational cost. Looking forward, VLA models themselves are likely to benefit from significant future improvements to efficiency. Techniques such as model quantization and distillation, for example, may substantially reduce inference times, making VLAPS increasingly practical for operational deployment.

VII. Conclusions

We present vision-language-action planning and search (VLAPS)—a framework that integrates model-based search with pre-trained vision-language-action (VLA) models to enable generalist robot policies that are robust, and capable of reasoning over future action outcomes. By using a VLA model to define context-aware priors over the search space, VLAPS enables efficient search over robot actions, even in environments with large action spaces and sparse rewards. Our experiments in language-specified robot reasoning and manipulation tasks demonstrate that VLAPS consistently and significantly outperforms VLA-only baseline policies. Importantly, VLAPS requires no additional training and is agnostic to the specific VLA model used—our results indicate that its performance will improve alongside future advances in pre-trained VLA models. These results highlight VLAPS as a promising direction for increasing test-time compute and model-aware planning capabilities in VLA policies.

Acknowledgment

We acknowledge funding support from the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Canadian Institute for Advanced Research (CIFAR), as well as compute support from the Digital Research Alliance of Canada, Mila IDT, and NVIDIA.

References

[1] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023.

[2] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, *et al.*, "Libero: Benchmarking knowledge transfer for lifelong robot learning," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 44 776–44 791.

[3] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, *et al.*, "RT-1: Robotics Transformer for Real-World Control at Scale," in *Proceedings of Robotics: Science and Systems*, 2023.

[4] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*, vol. 229, PMLR, 2023.

[5] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*, PMLR, 2023, pp. 785–799.

[6] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, *et al.*, "PaLM-e: An embodied multimodal language model," in *International Conference on Machine Learning*, vol. 202, PMLR, 2023.

[7] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, *et al.*, "Octo: An open-source generalist robot policy," in *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

[8] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, *et al.*, "Droid: A large-scale in-the-wild robot manipulation dataset," in *Robotics: Science and Systems*, 2024.

[9] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, *et al.*, "Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0," in *IEEE International Conference on Robotics and Automation*, IEEE, 2024.

[10] P. Atreya, K. Pertsch, T. Lee, M. J. Kim, A. Jain, A. Kuramshin, *et al.*, "Roboarena: Distributed real-world evaluation of generalist robot policies," in *Conference on Robot Learning*, 2025.

[11] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, *et al.*, "OpenVLA: An open-source vision-language-action model," in *Conference on Robot Learning*, 2024.

[12] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, *et al.*, "Pi0: A vision-language-action flow model for general robot control," *arXiv preprint arXiv:2410.24164*, 2024.

[13] K. Pertsch, K. Stachowicz, B. Ichter, D. Driess, S. Nair, Q. Vuong, *et al.*, "Fast: Efficient action tokenization for vision-language-action models," *arXiv preprint arXiv:2501.09747*, 2025.

[14] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter brian, F. Xia, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, Curran Associates, Inc., 2022.

[15] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, *et al.*, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in neural information processing systems*, vol. 36, 2023.

[16] V. Xiang, C. Snell, K. Gandhi, A. Albalak, A. Singh, C. Blagden, *et al.*, "Towards system 2 reasoning in llms: Learning how to think with meta chain-of-though," *arXiv preprint arXiv:2501.04682*, 2025.

[17] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, *et al.*, "Self-consistency improves chain of thought reasoning in language models," in *International Conference on Learning Representations*, 2023.

[18] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, *et al.*, "Reasoning with language model is planning with world model," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[19] X. Feng, Z. Wan, M. Wen, Y. Wen, W. Zhang, and J. Wang, "Alphazero-like tree-search can guide large language model decoding and training," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[20] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[21] Y. Ji, J. Li, H. Ye, K. Wu, J. Xu, L. Mo, *et al.*, "Test-time computing: From system-1 thinking to system-2 thinking," *arXiv preprint arXiv:2501.02497*, 2025.

[22] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, *et al.*, "From system 1 to system 2: A survey of reasoning large language models," *arXiv preprint arXiv:2502.17419*, 2025.

[23] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, *et al.*, "Isaac gym: High performance GPU based physics simulation for robot learning," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[24] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, *et al.*, "Robosuite: A modular simulation framework and benchmark for robot learning," *arXiv preprint arXiv:2009.12293*, 2020.

[25] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations*, 2020.

[26] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," in *International Conference on Learning Representations*, 2021.

[27] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[28] S. Yang, Y. Du, S. K. S. Ghasemipour, J. Tompson, L. P. Kaelbling, D. Schuurmans, *et al.*, "Learning interactive real-world simulators," in *International Conference on Learning Representations*, 2024.

[29] J. Bruce, M. D. Dennis, A. Edwards, J. Parker-Holder, Y. Shi, E. Hughes, *et al.*, "Genie: Generative interactive environments," in *International Conference on Machine Learning*, 2024.

[30] Y. Feng, J. Han, Z. Yang, X. Yue, S. Levine, and J. Luo, "Reflective planning: Vision-language models for multi-stage long-horizon robotic manipulation," *arXiv preprint arXiv:2502.16707*, 2025.

[31] J. Parker-Holder, P. Ball, J. Bruce, V. Dasagi, K. Holsheimer, C. Kaplanis, *et al.*, "Genie 2: A large-scale foundation world model," 2024.

[32] Y. Du, S. Yang, P. Florence, F. Xia, A. Wahid, brian ichter, *et al.*, "Video language planning," in *The Twelfth International Conference on Learning Representations*, 2024.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[35] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[36] D. P. Bertsekas, "Model predictive control and reinforcement learning: A unified framework based on dynamic programming," *IFAC-PapersOnLine*, vol. 58, no. 18, 2024, 8th IFAC Conference on Nonlinear Model Predictive Control NMPC 2024. DOI: https://doi.org/10.1016/j.ifacol.2024.09.056.

[37] N. A. Hansen, H. Su, and X. Wang, "Temporal difference learning for model predictive control," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 17–23 Jul 2022, pp. 8387–8406.

[38] N. Hansen, H. Su, and X. Wang, "TD-MPC2: Scalable, robust world models for continuous control," in *International Conference on Learning Representations*, 2024.

[39] M. Dalal, T. Chiruvolu, D. S. Chaplot, and R. Salakhutdinov, "Plan-seq-learn: Language model guided RL for solving long horizon robotics tasks," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[40] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*, 2022.

[41] H. Chang, K. Gao, K. Boyalakuntla, A. Lee, B. Huang, J. Yu, *et al.*, "Lgmcts: Language-guided monte-carlo tree search for executable semantic object rearrangement," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024. DOI: 10.1109/IROS58592.2024.10802562.

[42] L. X. Shi, brian ichter, M. R. Equi, L. Ke, K. Pertsch, Q. Vuong, *et al.*, "Hi robot: Open-ended instruction following with hierarchical vision-language-action models," in *International Conference on Machine Learning*, 2025.

[43] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine, "Robotic control via embodied chain-of-thought reasoning," in *Conference on Robot Learning*, 2024.

[44] L. fine-grained bimanual manipulation with low-cost hardware, "Zhao, tony z and kumar, vikash and levine, sergey and finn, chelsea," in *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[45] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver, "Learning and planning in complex action spaces," in *International Conference on Machine Learning*, PMLR, 2021.

[46] Y. J. Ma, J. Hejna, C. Fu, D. Shah, J. Liang, Z. Xu, *et al.*, "Vision language models are in-context value learners," in *International Conference on Learning Representations*, 2024.

[47] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, *et al.*, "Octo: An open-source generalist robot policy," in *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

[48] L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, *et al.*, "Paligemma: A versatile 3b vlm for transfer," *arXiv preprint arXiv:2407.07726*, 2024.