Value Vision-Language-Action Planning & Search

Ali Salamatian*,†, Ke (Steve) Ren*, Kieran Pattison*, Cyrus Neary
The University of British Columbia
*Equal contribution †Project lead

Abstract

Vision-Language-Action (VLA) models have emerged as powerful generalist policies for robotic manipulation, yet they remain fundamentally limited by their reliance on behavior cloning, leading to brittleness under distribution shift. While augmenting pretrained models with test-time search algorithms like Monte Carlo Tree Search (MCTS) can mitigate these failures, existing formulations rely solely on the VLA prior for guidance, lacking a grounded estimate of expected future return. Consequently, when the prior is inaccurate, the planner can only correct action selection via the exploration term, which requires extensive simulation to become effective. To address this limitation, we introduce Value Vision-Language-Action Planning and Search (V-VLAPS), a framework that augments MCTS with a lightweight, learnable value function. By training a simple multi-layer perceptron (MLP) on the latent representations of a fixed VLA backbone (Octo), we provide the search with an explicit success signal that biases action selection toward high-value regions. We evaluate V-VLAPS on the LIBERO robotic manipulation suite, demonstrating that our value-guided search improves success rates by over 5 percentage points while reducing the average number of MCTS simulations by 5–15% compared to baselines that rely only on the VLA prior.

1 Introduction

Deploying robotic policies in open-world settings requires reliability under distribution shift. Recent advances in robot learning have been driven by large-scale Vision-Language-Action (VLA) models, transformer policies that predict action sequences from multimodal observations and language instructions. While VLA models serve as effective priors for generalist behaviors, they are fundamentally limited by their reliance on behavior cloning. As a result, they often exhibit brittle behavior when facing out-of-distribution (OOD) states.

A promising approach to address this problem is to augment the pretrained model with a planning search algorithm that explores possible outcomes in simulation. One such search algorithm is Monte Carlo Tree Search (MCTS), which constructs a search tree by simulating rollout trajectories, balancing exploration of uncertain actions with exploitation of high-likelihood ones (Świechowski et al., 2022). Following Neary et al. (2025), we can leverage a VLA model as a policy prior to guide the MCTS, using a visit-count heuristic to manage exploration. However, we observe that relying solely on the VLA prior is insufficient for robust long-horizon planning. In the formulation proposed by Neary et al. (2025), the search lacks a value function, meaning it has no grounded estimate of the expected future return. Consequently, if the VLA prior is inaccurate, assigning high probability to suboptimal actions, the planner has no mechanism to correct this bias other than exhaustive count-based exploration. In other words, the search relies on the imitation prior rather than the underlying reward structure.

To address this limitation, we introduce Value Vision-Language-Action Planning and Search (V-VLAPS). As shown in Figure 1, we augment the MCTS action selection with a lightweight and learnable value head that is similar to the search formulation used by Silver et al. (2017b). This value head provides the missing reward signal, biasing action selection toward states with higher

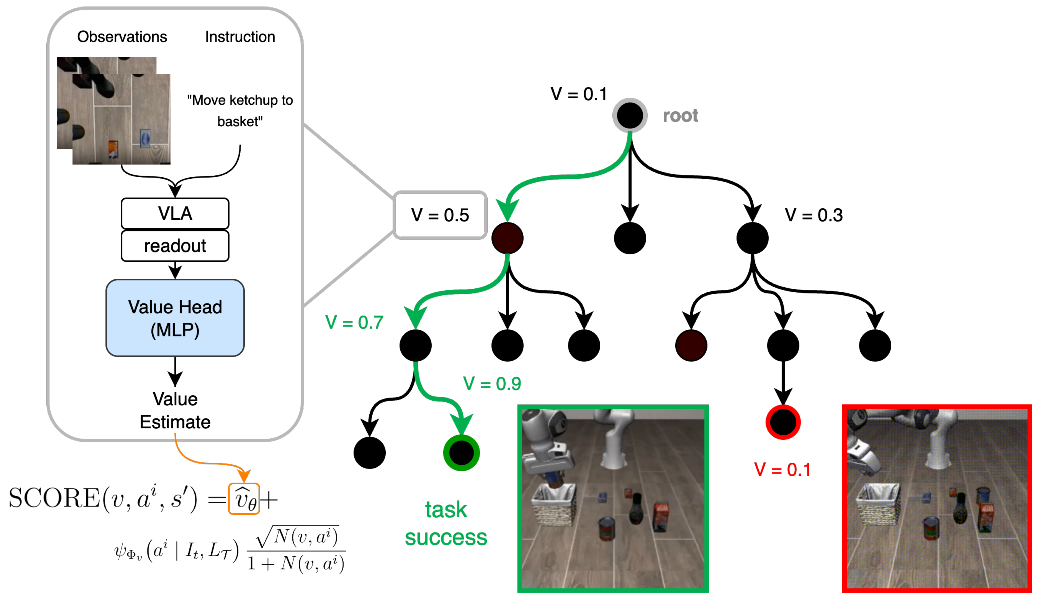

Figure 1: Overview of our value-guided VLAPS extension. At each MCTS node, the current observations and language instruction (e.g., "Move ketchup to basket") are passed through the frozen VLA backbone and our value head (MLP) to produce a scalar value estimate. This value is attached to the corresponding node and used in the VLAPS scoring rule to bias node selection. Nodes with higher predicted value (green) are selected more often and tend to lead to successful task completions, while low-value nodes (red) are down-weighted during search.

expected return and effectively correcting the search when the prior policy is inaccurate. This is a promising approach, as we show in Section 4, even training a very small value head on a small set of data can help with biasing the action selection towards the right region of the action space. We further show the benefits of adding a value head by evaluating V-VLAPS in a simulated suite of robotic manipulation tasks (LIBERO). Our simple change results in over 5 percentage points improvement of the success rate while reducing the average number of MCTS simulations by 5-15% depending on the task suite.

2 Related Work

2.1 Vision–Language–Action Policies

Vision–language–action (VLA) models are designed to map visual observations and natural-language instructions directly to robot actions. Early systems showed that a single transformer policy can be trained on large collections of robot demonstrations and language-labelled tasks, and then be reused across many manipulation skills (Brohan et al., 2023b;a). In our work, we use Octo (Team et al., 2024) as a fixed generalist VLA backbone and build our method on top of its latent representation; the same method can be used with other VLA models.

Although VLA models can perform a wide range of tasks, they are typically used in a purely reactive way: at each step, the model receives the current observation and instruction and outputs the next action (or action chunk), without explicit long-horizon planning. In challenging scenes or out-of-distribution configurations, this can lead to failures that the policy does not recover from (Gu et al., 2025). In the language modeling paradigm, many methods have been introduced to mitigate similar issues by scaling test-time compute, such as chain-of-thought prompting (Wei et al., 2023), self-consistency sampling (Wang et al., 2023), and MCTS during inference (Hao et al., 2023).

Neary et al. (2025) extend these ideas to the robotics domain to address the limitations of reactive execution. They introduced Vision-Language-Action Planning and Search (VLAPS), which embeds a pre-trained VLA into a model-based search procedure and uses the VLA to define action proposals and abstractions for planning. In our work, we build on VLAPS and keep the same fixed VLA backbone, but additionally learn a value function over its latent state and incorporate the resulting value predictions into VLAPS's scoring rule to guide the search.

2.2 Monte Carlo Tree Search

A natural way to compensate for the myopic behaviour of reactive policies is to add a search procedure that can simulate possible future outcomes before committing to an action. Monte Carlo Tree Search (MCTS) is a widely used algorithm for this purpose. It incrementally builds a search tree by repeatedly simulating trajectories from the current state, and at each iteration it selects actions that balance exploring uncertain branches with exploiting branches that have produced good returns in previous simulations (Świechowski et al., 2022). Typically, each iteration proceeds through four phases: selection, expansion, simulation, and backpropagation. The resulting statistics over node visit counts and returns are then used to estimate action values at the root.

MCTS has been particularly successful in domains where a reasonably accurate simulator is available. In board games such as Go, Chess, and Shogi, MCTS combined with learned policies and values has led to systems that reach and surpass human expert performance (Silver et al., 2016; 2017a). In these settings, the search runs entirely in simulation and only the final chosen move is executed in the real environment. Neary et al. (2025) bring this style of planning to vision–language–action models in the VLAPS framework. They use MCTS over discrete action chunks, with a pre-trained VLA policy providing action priors that focus and refine the search of an otherwise completely intractable space. Their formulation combines their VLA-informed prior with visit-count-based exploration heuristics in a PUCT-style scoring rule which allows MCTS to exploit the strengths of the pre-trained policy while still exploring alternative action sequences.

2.3 Learned Value Functions for Tree Search and VLA Planning

Combining learned value functions with tree search has proved highly effective in domains where planning is possible. In AlphaGo and its successors, deep neural networks predict both a policy over moves and a scalar value estimate from a board position; Monte Carlo Tree Search uses the policy to prioritize promising actions and the value to evaluate leaf nodes without long rollouts (Silver et al., 2016; 2017b;a). This combination allows the search procedure to concentrate computation on moves that are both likely under the learned policy and lead to positions that the value function predicts as strong, rather than relying solely on visit counts and simulation outcomes of intractably long rollouts.

VLAPS adopts a similar strategy for vision–language–action policies, but stops short of learning an explicit value function over the VLA latent state. In Neary et al. (2025), the quality of a node is determined by search statistics and VLA-derived action priors, but there is no separate learned estimate of how good it is to be at a particular state. In contrast, we keep the underlying VLA and search procedure fixed, and introduce a small value head which takes the VLA latent state as input. This head is trained to predict Monte Carlo returns from VLA rollouts, and its predictions are incorporated into the VLAPS selection rule. Conceptually, this brings the use of value functions in VLAPS closer to the value-guided tree search used in game-playing systems like AlphaGo, while operating in the setting of vision–language–action control.

3 Methodology

We extended VLAPS by learning a value function over Octo latent states and using this value to guide Monte Carlo Tree Search. We first collected rollouts of the Octo model and computed Monte Carlo value targets for each state (3.1). We then trained a three-layer MLP (the "value head") to predict these targets from Octo's last-layer representation (readout) (3.2). Finally, we integrated the learned value into VLAPS' tree search by modifying the node selection score to bias branches towards high-value nodes (3.3).

3.1 Data Collection

We construct training data for the value head by rolling out a fixed pretrained VLA policy, without any planning, on LIBERO tabletop manipulation tasks. At the beginning of each episode, the environment provides an initial observation o_0 and a language instruction g. At each decision step t, the VLA (Octo) receives (o_t, g), computes a latent readout vector $h_t \in \mathbb{R}^d$ which acts as the summary representation of the past observations and the task. It then uses the readout to output an action chunk c_t, which consists of a sequence of low-level actions. We execute the entire chunk c_t in the environment, applying each low-level action in order, until the chunk finishes or the episode terminates. The environment then returns the next observation o_{t+1}, and we query the VLA again. This produces an episode as a sequence of decision steps $(o_0, c_0, o_1, c_1, ..., o_T)$.

Episodes terminate either when the task is successfully completed or when a timeout horizon is reached. We use a sparse terminal reward: if the task is successfully completed before the timeout, the episode receives a terminal reward of 1; otherwise (failure or timeout), it receives a terminal reward of 0. All intermediate rewards are zero. Thus each episode has a binary outcome indicating success or failure.

For each decision step t in an episode, we define a Monte Carlo value target by propagating the terminal reward backward through time. Let $R \in \{0, 1\}$ denote the terminal reward for the episode and let T be the index of the final decision step. We assign each state at decision step t a target:

$$G_t = \begin{cases} \gamma^{T-t}, & \text{if the episode ends in success } (R = 1), \\ 0, & \text{if the episode ends in failure } (R = 0). \end{cases}$$

where $\gamma \in (0, 1]$ is a discount factor (we set it to 0.99). This target can be interpreted as a Monte Carlo estimate of the state value under our sparse success reward when following the VLA policy.

We extract the readout vector and pair it with its value target to obtain training examples (h_t, G_t). Aggregating these pairs across many VLA rollouts yields a dataset of state–value examples that we use to train the value head.

3.2 Value Head Training

Our goal is to learn a value function that maps the VLA's latent representation at each decision step to the scalar value target defined in Section 3.1. Concretely, given the Octo readout vector $h_t \in \mathbb{R}^d$ at decision step t, we learn a function $V_\theta : \mathbb{R}^d \to \mathbb{R}$ that predicts a scalar estimate $\hat{v}_t$ of the state value: $\hat{v}_t = V_\theta(h_t)$.

We parameterize V_θ as a lightweight three-layer multilayer perceptron (MLP). The architecture is intentionally small compared to the underlying Octo model, so that evaluating the value head adds minimal overhead during planning.

We train this value head on the dataset of readout–value pairs $\{(h_t, G_t)\}$. The training objective is to regress onto the Monte Carlo value targets G_t using mean squared error: $\mathcal{L}(\theta) = \mathbb{E}_{(h_t, G_t)}\big[(V_\theta(h_t) - G_t)^2\big]$.

In practice, we approximate this expectation by sampling mini-batches from the dataset and performing stochastic gradient descent with the Adam optimizer. During training, the parameters of the underlying Octo VLA are kept fixed; only the value head parameters θ are updated. After convergence, V_θ provides a compact estimate of the state value at each decision step, which we use to guide the tree search.

3.3 Value Integration

After training our MLP, we defined a PUCT style score for selecting nodes in Monte Carlo Tree Search (Silver et al., 2016). This amounts to using a selection score with the following form, for a given node-action pair (v, a):

$$\text{PUCT}(v, a) = Q(v, a) + U(v, a)$$

Following Neary et al. (2025), we defined the U portion of this score by:

$$U(v, a^i) = \text{VLAPS\_SCORE}(v, a^i) = \psi_{\Phi_v}\left(a^i \mid I_t, L_{\mathcal{T}}\right) \frac{\sqrt{N(v, a^i)}}{1 + N(v, a^i)}$$

where v, a^i are the current node and ith sampled action chunk from the VLA defined distribution β_Φ and $L_{\mathcal{T}}, I_t, N(v, a^i)$ are the task description, observation at time t, and number of times the ith action has been taken from the node v. The distribution β_Φ is a softmax centred around a sampled action chunk from the VLA of action chunks using the flattened euclidean distances from action chunks in a preset, finite library; $\psi_{\Phi_v}\left(a^i \mid I_t, L_{\mathcal{T}}\right)$ is defined similarly but using the candidate actions sampled from the distribution described above instead of the whole library.

The Q portion of our score is then defined using our value estimator:

$$Q(v, a^i) = V_\theta(h)$$

where V_θ is our value estimator parametrized by θ, and h is the VLA's transformer outputs of state information i.e. observations, task description at the state reached by taking a^i. We use a value function approximation as opposed to a state-action value function (V vs. Q) because in this setup, transitions are deterministic. Taken altogether, the selection score is as follows:

$$\text{SCORE}(v, a^i, s') = \widehat{v}_\theta(\text{readout}(s')) + \psi_{\Phi_v}\left(a^i \mid I_t, L_{\mathcal{T}}\right) \frac{\sqrt{N(v, a^i)}}{1 + N(v, a^i)}$$

We then use this score to select nodes to expand, or in the case that they are already expanded, expand their subtrees.

4 Experiments and Results

4.1 Dataset and Training Setup

Using the method described in Section 3.1, we collected readout–value pairs across multiple LIBERO tasks. Specifically, we collected training data for tasks 1, 6, and 8 from the spatial suite, and tasks 0, 1, 2, 3, 4, 6, and 8 from the object suite. LIBERO Object tasks primarily test object-centric manipulation: the robot must pick up a specified object and place it into a basket. LIBERO Spatial tasks emphasize spatial reasoning: the robot must pick up a target object from a specified position and place it onto another target region, such as a plate. Interestingly, we observed some of these tasks are much harder for the VLA. For example object task 6 has around 0% success rate, resulting in highly imbalanced data. To prevent the value head from collapsing to always predicting zero, we applied class balancing by randomly downsampling failure states so that the number of successful and failed examples was equal. Table 1 summarizes the final dataset composition per task.

We trained three separate value heads on this dataset: one trained only on spatial tasks, one trained only on object tasks, and one trained jointly on both suites. This allows us to measure the extent to which value learning on one suite benefits the other.

Task	**Train Samples**	**Test Samples**	**Avg Success Episode Length (VLA) in Steps**
Spatial	223931	24881	48.81
Object	300226	33358	29.58
Total	**524157**	**58239**	**37.80**

Table 1: Dataset size and average episode lengths for success trajectories under the VLA policy.

	Spatial Suite Success Rate (%)				**Object Suite Success Rate (%)**			
Task	**VLA**	**VLAPS**	**V-VLAPS (spatial)**	**V-VLAPS (both)**	**VLA**	**VLAPS**	**V-VLAPS (object)**	**V-VLAPS (both)**
0	70	**100**	98	**100**	20	96	**100**	99
1	70	**100**	**100**	91	**100**	80	**100**	90
2	10	83	**93**	90	90	90	**100**	90
3	30	**100**	**100**	**100**	20	89	96	**97**
4	18	73	78	**83**	70	**100**	89	90
5	10	**81**	79	70	60	**100**	**100**	**100**
6	36	78	87	**92**	0	17	**38**	28
7	33	99	91	**100**	0	**75**	61	59
8	20	90	99	**100**	0	**54**	50	39
9	0	16	**47**	32	40	**97**	92	84
Overall	29.7	82.0	**87.2**	85.8	40.0	79.8	**82.6**	77.6

Table 2: Success rate (%) across LIBERO Spatial and Object suites for different planning methods.

4.2 Quantitative Evaluation

To evaluate whether the learned value improves planning quality, we compared the following models on both spatial and object LIBERO suites: the base VLA policy without planning, VLAPS without a learned value, V-VLAPS trained only on one suite, and V-VLAPS trained on both suites. To ensure a robust evaluation, we evaluated each method across 10 rollouts per initial state for all available initializations (0–9). As seen in Table 2, adding the value head increased the success rate by 5.2% for the spatial task suite and 2.8% for the object task suite. Table 3 demonstrates that the average number of MCTS simulations, how many times we started the search again from the root node, is also lower for V-VLAPS compared to VLAPS. Specifically, we observe 5% reduction for spatial suite and 14% for the object suite. Task 9 for the spatial suite is particularly interesting as it clearly highlights the benefit of adding a value head. This is an especially difficult task for Octo as it consistently fails across all different initial states. In this case, V-VLAPS performs 31% better than VLAPS while spending fewer iterations in MCTS. Importantly, our collected dataset did not include any example for task 9, which suggests that the learned value head generalized well across the tasks within the same suite. Interestingly, it seems like training on both tasks suites does not help with the performance. This raises the question of whether we should use one value head that generalizes over different tasks suites or whether we would need one value head per task suite; we further discuss this in Section 6.

4.3 Qualitative Analysis

Motivated by the performance gains we observed and by prior work suggesting a separation between success and failure latent representations in other VLA models (Gu et al., 2025), we analyzed the

Task	Spatial Suite Avg MCTS Simulation			Object Suite Avg MCTS Simulation		
	VLAPS	**V-VLAPS (spatial)**	**V-VLAPS (both)**	**VLAPS**	**V-VLAPS (object)**	**V-VLAPS (both)**
0	3.19	6.44	3.84	6.62	5.33	6.03
1	3.30	3.53	6.28	7.35	2.36	6.66
2	10.04	10.77	10.27	5.58	2.54	7.33
3	4.24	4.13	3.69	9.07	7.68	6.15
4	19.00	11.61	10.72	3.47	7.59	6.37
5	19.25	13.75	15.67	2.65	2.30	3.83
6	11.75	9.96	5.02	28.99	27.69	31.79
7	8.00	5.86	3.50	18.60	16.25	17.49
8	6.05	6.92	6.32	25.57	22.87	25.51
9	24.81	20.65	28.80	7.58	7.77	10.96
Overall	10.96	10.46	9.71	11.95	10.24	12.61

Table 3: Comparison of planning efficiency metrics across Spatial and Object suites.

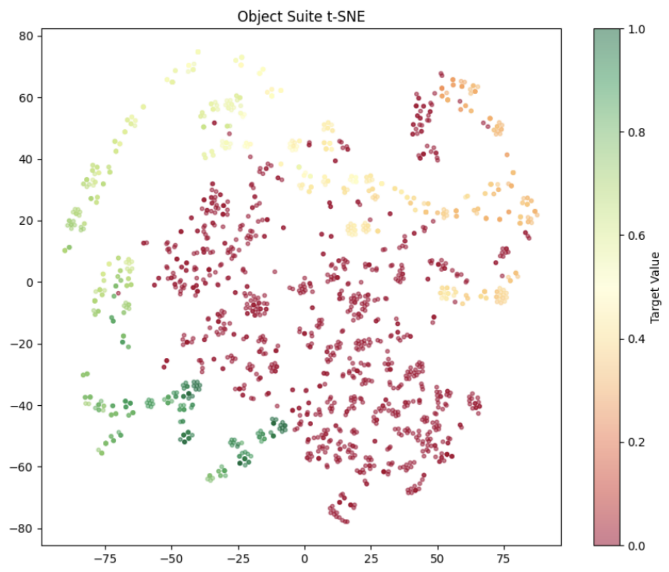

Figure 2: t-SNE projection of transformer readouts from LIBERO tasks, colored by value target.

structure of Octo latent readouts. We projected readouts into two dimensions using t-SNE and inspected their distribution. Figure 2 visualizes this embedding and highlights differences between successful and unsuccessful trajectories. As seen, a very nice pattern emerges, where we can see the path that should be taken in this 2D space to go from the initial state to the target, suggesting the readouts may encode enough information to get an estimation for the expected reward.

We also performed some qualitative analysis by visualizing the value estimations alongside corresponding rollouts. Figure 3 shows an example where the predicted value increases steadily as the robot approaches a successful completion state. Most of the learned values behaved meaningfully,

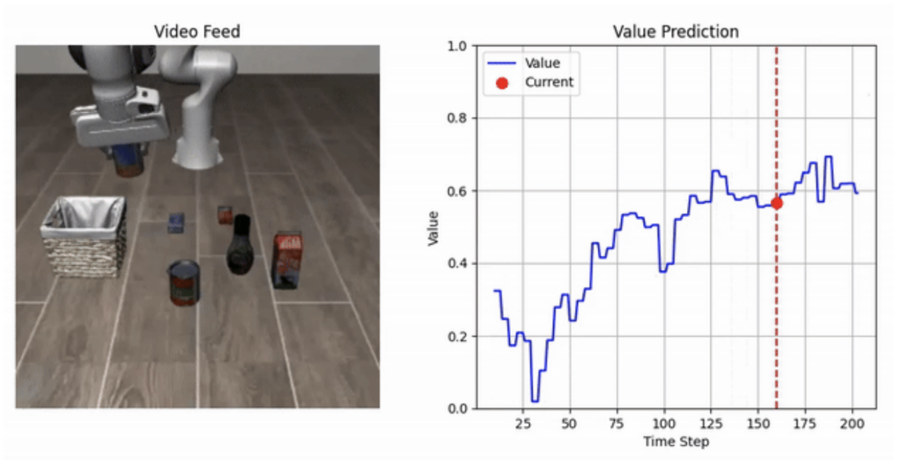

Figure 3: Example qualitative evaluation showing the predicted value throughout a successful trajectory.

although we observed occasional failures due to rare behaviors such as object drops. We expect this issue to diminish as we train on a larger and more diverse dataset that includes more tasks.

5 Discussion and Limitations

Due to time and resource constraints, we could only test on tasks from two of the LIBERO task suites: Spatial and Object, and train on a subset of those. This setup naturally limits the ability of our value estimator to generalize effectively and limits the scope of our results to a more narrow class of tasks. Taking this into account, we were still able to achieve some improvement, and we hope to remedy these constraints in the future to iterate on these results.

The Octo VLA model comes with its own limitations. The data collected from episodes performed under the Octo VLA policy can be of poor quality, namely, on many tasks Octo either fails on all rollouts or succeeds on all rollouts. This uniformity makes it challenging to learn a general and interpretable value estimator with no reward signal. Once again, we still saw success and anticipate that if we collect data using a more sophisticated, guided approach, that we can see even better results. We see some structure in the readouts produced by the transformer on the Object suite, but less so in Spatial. Due to this, we also list the transformer readouts as a potential limitation as latent inputs to our value estimator. We might improve this by using a different VLA or passing a bigger observation context into the transformer. Finally, we will further investigate the generalization of our value head across task suites, training on single suites and multiple suites to assess the feasibility of training a generalist value estimator.

Lastly, like in Neary et al. (2025), we must assume access to a realistic simulator/world model to carry out Monte Carlo tree search. Handling the inaccuracies that a simulator exhibits is an active area of research, and in the scope of this project, we do not address it; given the advancements in simulation accuracy and efficacy, we believe that this work can be extended to physical robotic tasks. Keeping with the theme of realism, the additional VLA calls that tree search requires are computationally expensive and time-consuming. We hope that improvements continue to be made in inference time and compute for VLA models and look forward to optimizing our current implementation for efficiency.

6 Conclusions and Future Work

We have presented V-VLAPS, a method that augments VLA-guided MCTS with a lightweight, learnable value estimator. Our results demonstrate that this simple, low-cost addition significantly improves planning efficiency and success rates by biasing the search toward high-return states.

Moving forward, we identify three key directions to enhance this framework. First, we aim to improve generalization by training on more diverse task suites. Specifically, we intend to analyze whether a single value head can effectively generalize across different domains or if task-specific estimators are required. Second, we plan to guide the data collection process using VLAPS. We hypothesize that training on search-generated data, rather than VLA rollouts, will yield more accurate value estimates and better test-time guidance. Finally, we intend to explore adaptive compute, using the learned value estimator to dynamically prune unpromising branches during rollouts, thereby allocating the search budget more efficiently.

References

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023a. URL https://arxiv.org/abs/2307.15818.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023b. URL https://arxiv.org/abs/2212.06817.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models, 2025. URL https://arxiv.org/abs/2506.09937.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

Cyrus Neary, Omar G. Younis, Artur Kuramshin, Ozgur Aslan, and Glen Berseth. Improving pretrained vision-language-action policies with model-based search, 2025. URL https://arxiv.org/abs/2508.12211.

David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017a. URL https://arxiv.org/abs/1712.01815.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, October 2017b. doi: 10.1038/nature24270.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy, 2024. URL https://arxiv.org/abs/2405.12213.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3): 2497–2562, July 2022. ISSN 1573-7462. doi: 10.1007/s10462-022-10228-y. URL http://dx.doi.org/10.1007/s10462-022-10228-y.