HIERARCHICAL VISION LANGUAGE ACTION MODEL USING SUCCESS AND FAILURE DEMONSTRATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision–Language–Action (VLA) models are typically post-trained on teleoperated *successful* demonstrations, while the many *failed* attempts that occur naturally during data collection are discarded. Those failures record where and how a policy is fragile, and we show they become a usable learning signal once plan evaluation is posed as a *reach–avoid* problem. We introduce **VINE**, a hierarchical VLA that separates high-level reasoning (System 2) from low-level control (System 1) under a semi-Markov option formulation. Its central component is a subgoal-level value function that we prove equals the first-exit probability of reaching the goal set before the failure set. Because that quantity is defined by *outcomes* rather than by expert actions, it can be regressed directly from mixed-quality offline data, with no reward shaping and no online interaction. System 2 uses this value to run a batched tree search over a text-serialized scene-graph abstraction, proposing subgoal transitions, predicting their successor states, and pruning brittle branches before any motion is executed. System 1 then executes the selected subgoal sequence without altering the agent's motor skills. Across simulated manipulation, the SIMPLER benchmark, and a real 6-DoF arm, **VINE** improves success most sharply under distribution shift ($1.6\times$ over the π_0 baseline on unseen plug insertion and $5.5\times$ on unseen long-horizon real-world packing), suggesting that failure data discarded during collection can improve execution under shift.

1 INTRODUCTION

Vision–Language–Action models (VLAs) learn instruction-conditioned manipulation policies from robot demonstrations (Black et al., 2024; Kim et al., 2024; Bjorck et al., 2025). A language instruction specifies a task objective but generally does not uniquely determine the sequence of subgoals required to achieve it. Multiple sequences can satisfy the same instruction, yet only some of them are feasible under a given scene configuration and the robot's execution capabilities. For example, different object-placement orders may achieve the same packing objective, but an early placement can obstruct a subsequent one. Reliable task execution therefore requires selecting among alternative subgoal sequences by considering both their immediate feasibility and their consequences for subsequent steps.

We hypothesize that **mixed-quality demonstrations** contain the information needed for this selection problem. Successful demonstrations identify feasible task executions, while failed attempts reveal the conditions under which particular subgoal transitions are unsuccessful (Choi et al., 2019; Hoang et al., 2024). These outcomes can inform plan evaluation even when the corresponding actions are unsuitable as imitation targets.

Our key insight is to use mixed-quality demonstration outcomes to supervise feasibility estimation at the level of subgoal transitions. We associate success and failure labels with individual transitions and learn a value function that evaluates progress toward task completion. We formalize this objective as a reach–avoid problem, in which a subgoal sequence is evaluated by its probability of reaching the task goal before encountering failure. This formulation enables successful and failed demonstrations to jointly supervise plan evaluation without requiring imitation of unsuccessful actions. The resulting value function is trained on offline demonstrations and scores alternative subgoal sequences during planning.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We introduce **VINE**, **V**ision–Language–Action model **I**ntegrating **N**egative **E**xperience, a hierarchical VLA framework with a high-level planner (System 2) and a low-level controller (System 1). Both systems share a language backbone and differ only in system-specific adapters and prediction heads. System 2 represents states as text-serialized scene graphs, proposes subgoals, predicts successor states, and estimates their feasibility. These estimates guide a tree search over alternative subgoal sequences, which accounts for their downstream consequences. System 1 generates actions conditioned on the selected subgoal and its predicted target state, and predicts when to advance to the next subgoal. We train System 2 on both successful and failed demonstrations to learn plan generation and evaluation, and System 1 on successful demonstrations to learn subgoal execution and termination. Both systems are trained offline, yielding a shared-backbone architecture that integrates failure-aware planning and action generation.

We evaluate VINE across simulated plug insertion and drawer packing, the SIMPLER benchmark, and real-world manipulation with a 6-DoF arm. The gains are particularly pronounced under changes in object appearance, geometry, and configuration. On the two simulation tasks, replacing language-model-confidence scoring with our failure-trained value increases unseen success from 28.9% to 42.2% and from 52.5% to 67.5%, respectively. On real-world long-horizon packing, VINE achieves 36.7% success on unseen configurations, compared with 6.7% for the π_0 baseline.

Our main contributions are as follows: (1) We formulate subgoal-level plan evaluation as a reach–avoid objective and construct outcome supervision from successful and failed demonstrations. (2) We introduce VINE, which combines a learned feasibility value with tree search over predicted scene-graph transitions and VLA-based execution. (3) We evaluate the approach across simulation, SIMPLER, and real-world manipulation, with controlled comparisons that isolate the contributions of failure supervision and search, and release an environment suite and a mixed-quality dataset for failure-aware planning.

2 Related Work

VLAs and value-guided post-training. VLA models map images and language to actions end-to-end (Brohan et al., 2023; Zitkovich et al., 2023; Open X-Embodiment Collaboration, 2023; Octo Model Team et al., 2024; Kim et al., 2024; Wen et al., 2025), increasingly with flow-matching action experts (Black et al., 2024; Physical Intelligence, 2025; Bjorck et al., 2025) or explicit reasoning traces (Zawalski et al., 2024; Zhao et al., 2025; Huang et al., 2025). Several works post-train them with a learned value instead of pure imitation: advantage conditioning (Physical Intelligence et al., 2025), classifier-free guidance (Frans et al., 2025), action-chunk critics (Yang et al., 2026a; Zhai et al., 2025), test-time re-ranking with an offline Q-function (Nakamoto et al., 2024; Mao et al., 2025), return regression (Zhang et al., 2025), and values on predicted future observations (Wang et al., 2026; Chen et al., 2026). We share the premise that the value, not the expert action, is the appropriate signal for learning from failures, but all of them reweight or re-rank *actions*. VINE scores abstract *state transitions* and prunes a branch before the robot enters it, so failure data can exclude a plan rather than down-weight a motion. We compare against $\pi^*_{0.6}$-RECAP as the representative of this family (Table 1).

Hierarchical VLAs. Language-model planners that emit symbolic plans or code (Liang et al., 2022; Singh et al., 2023; Yao et al., 2023b) reason about task syntax and do not verify feasibility. Recent hierarchical VLAs train both levels, with the high level emitting language (Shi et al., 2025), 2D paths (Li et al., 2025), visual traces (Lee et al., 2025), or slow reasoning (Chen et al., 2025). Hume (Song et al., 2025) scores action chunks with a value for Best-of-N selection, whereas ours operates over state transitions and supports multi-step look-ahead. τ_0-VLA (Cai et al., 2026) scores open-ended language subtasks on images predicted by a world model. VINE instead searches over structured scene-graph transitions and scores them with a value trained on labelled failures.

Tree search with learned values. Tree search is standard in robot planning (Larsson et al., 2020; Labbé et al., 2020; Yershova & LaValle, 2007; Plaku et al., 2005) and in language-model reasoning (Yao et al., 2023a; Zhao et al., 2023; Zhang et al., 2024; Gao et al., 2024; Chi et al., 2025), where learned values guide MCTS (Liu et al., 2024a) or beam search over a visual dynamics model (Yang et al., 2026b). Search quality depends on the leaf evaluator, and success-only data provide no negative examples to calibrate it. Failure trajectories provide this evidence.

3 PROBLEM FORMULATION

In this section, we formalize instruction-conditioned manipulation from mixed-quality demonstrations as a reach–avoid problem over an abstract state graph, so that the *outcome* of a subgoal transition, rather than the expert's action, becomes the learning target and successes and failures are used in a single regression. Derivations are in §B.

3.1 AN SMDP OVER ABSTRACT STATES

Let the robot's low-level interaction be an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \rho)$ under an instruction ℓ, with states s (images and proprioception), actions a, transition kernel $T(\cdot \mid s, a)$, and initial distribution ρ. A *node* is a compact abstraction $n = \phi(s) \in \mathcal{N}$, and an *edge* $e \in \mathcal{E}$ is a subgoal token (e.g. "pick up the spoon") denoting a temporally extended transition executed by a closed-loop controller until termination. At decision epochs, i.e. along the node sequence (n_k), the induced process over $\mathcal{N}$ is a semi-Markov decision process (SMDP). The edge vocabulary includes a terminal token $\langle \texttt{done} \rangle$ that ends a plan. We distinguish the *symbolic* edge from its *option* grounding: each e is associated with an option $o_e = (I_e^N, \pi_e, \beta_e, \Pi_e)$: an initiation set $I_e^N \subseteq \mathcal{N}$ of nodes at which e is admissible, an intra-option policy $\pi_e(a \mid s, \ell)$, a termination rule $\beta_e : \mathcal{S} \to [0, 1]$, and a projection $\Pi_e = \phi$ mapping the terminal low-level state back to a node. System 2 is the meta-controller choosing among options, and System 1 realizes π_e and β_e.

We instantiate each node as a text-serialized 2D scene graph (Ni et al., 2024; Rana et al., 2023; Liu et al., 2025) representing objects and their spatial relations. Each edge represents a subgoal transition between consecutive gripper-state keyframes. System 2 predicts the successor graph n_{k+1} conditioned on the current graph and the selected edge. Training graphs are annotated offline, while System 2 generates them directly at inference (§E).

Data and labels. $\mathcal{D}_{\text{sys2}}$ holds tuples $(n_k, e_k, n_{k+1}, z_{0:k-1}, \ell, s_0, r_k)$ with history context $z_{0:k-1} = [n_0, e_0, \ldots, n_{k-1}]$ and sparse rewards ($r_k{=}0$ for $k{<}K$, $r_K \in \{0, 1\}$), where K is the step at which the episode terminates. $\mathcal{D}_{\text{sys1}}$ holds $(e_k, n_{k+1}, s_t, a_t, \ell)$, conditioning the low-level policy on both the intended transition and the target next state. The label r_K is derived from *step-level* failure labels rather than trajectory-level annotations: a transition fails when the demonstrator cannot reach $\mathcal{G}$ within a prescribed time window.

3.2 PLAN EVALUATION AS A REACH–AVOID PROBLEM

Let $\mathcal{G}$ be a target set and $\mathcal{F}$ an unsafe set, $\mathcal{G} \cap \mathcal{F} = \varnothing$, with hitting times $\tau_{\mathcal{G}} := \inf\{t \geq 0 : s_t \in \mathcal{G}\}$, $\tau_{\mathcal{F}} := \inf\{t \geq 0 : s_t \in \mathcal{F}\}$ and $\tau := \tau_{\mathcal{G}} \wedge \tau_{\mathcal{F}}$, at which the episode terminates. The target set is reached as soon as *some* admissible option enters it, whereas $\mathcal{F}$ is closed under options ($\operatorname{supp} P_o(\cdot \mid s) \subseteq \mathcal{F}$ for every $s \in \mathcal{F}$ and admissible o, where $P_o(\cdot \mid s)$ is the terminal-state distribution of option o started at s, §B), the formal counterpart of an irrecoverable manipulation state. With the sparse entrance reward $r_t := \mathbf{1}\{s_{t+1} \in \mathcal{G}\}$, the undiscounted return $R = \sum_{t=0}^{\tau-1} r_t$ equals 1 if $\tau_{\mathcal{G}} < \tau_{\mathcal{F}}$ and 0 otherwise, giving a discrete, option-level analogue of classical reach–avoid analysis (Margellos & Lygeros, 2011; Fisac et al., 2015) in which the reach and avoid sets are identified from mixed-quality data rather than from a known model.

Proposition 1 (First-exit value equals reach–avoid success probability) *Assume the node–edge process is Markov given the context z and that $\mathcal{G}, \mathcal{F}$ are absorbing. Then*

$$V(n, z) \;=\; \Pr\big(\tau_{\mathcal{G}} < \tau_{\mathcal{F}} \,\big|\, n, z\big). \tag{1}$$

Proof sketch. Under the first-exit convention the return equals the success indicator pathwise, so $V^{\mu} = \mathbb{E}[R] = \Pr(\tau_{\mathcal{G}} < \tau_{\mathcal{F}} \mid \cdot)$. The SMDP value at decision epochs coincides with the primitive-time sum because option chunks partition the time axis up to τ. The full argument, including the discounted case, is in §B. □

Equation 1 justifies the training procedure of §4: a regression onto observed outcomes (1 for a subgoal that reached $\mathcal{G}$, 0 for one that entered $\mathcal{F}$, bootstrapped otherwise) yields a feasibility score to which failed and successful demonstrations both contribute.

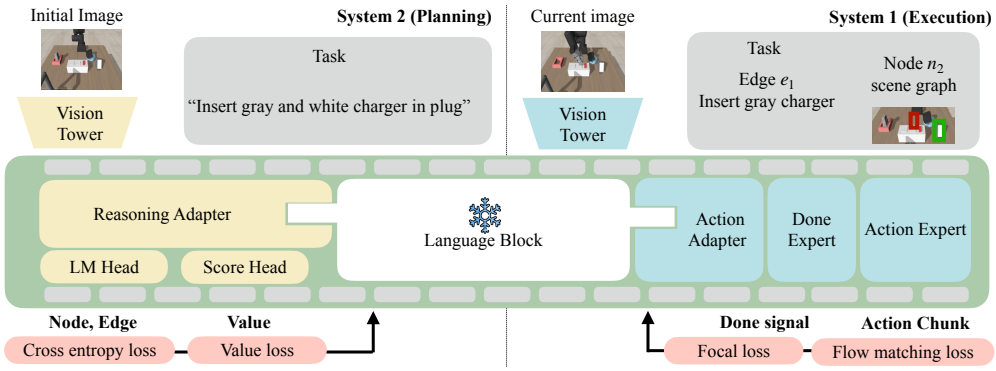


Figure 1: **Model architecture.** Both systems share a vision tower and a frozen language block with system-specific, trainable LoRA adapters. **System 2** (left) takes the initial image and the instruction and emits an edge e_k, a successor node n_{k+1}, and a value v_k through an LM head (cross-entropy) and a score head (expectile value loss). **System 1** (right) receives the current image together with the planned edge and target node, and produces actions via a flow-matching action expert and a termination signal via a focal-loss done expert.

4 VINE

In this section, we introduce VINE, a hierarchical VLA with a shared language backbone for planning and control. System 2 learns scene-graph generation, subgoal proposal, successor-state prediction, and value estimation from successful and failed demonstrations. System 1 learns subgoal execution and termination from successful demonstrations.

4.1 Overall pipeline and architecture

At inference the framework follows a plan–execute procedure: System 2 reads the initial image and instruction, emits the root scene graph, and runs a model-based tree search over candidate edges and their predicted successors, scoring each by its reach–avoid value. System 1 then executes the selected edges in turn, advancing on each termination signal. The tree is built once at the initial state, so latency is a one-time planning cost and System 1 runs at 20 Hz with no further overhead.

Formally, System 2 couples an option generator $\hat{P}_\theta(e \mid n, z, \ell, s_0)$, a learned world model $\hat{P}_\theta(n' \mid e, n, z, \ell, s_0)$, and a failure-aware value $V_\theta(n, z \mid \ell, s_0)$, and System 1 implements $\pi_\theta(a \mid s, \ell, e, n')$ and $\beta_\theta(s, \ell, e, n')$. We build on π_0 (Black et al., 2024) ($\theta_g^{\pi_0}$), which is specialized for action generation rather than text, and recover the text-generation capacity that tree search requires by linearly interpolating its shared layers with a PaliGemma (Beyer et al., 2024) trunk (θ_g^{PG}), $\theta_g^{\mathrm{merge}}(\lambda_m) = \lambda_m \theta_g^{\pi_0} + (1-\lambda_m)\theta_g^{\mathrm{PG}}$, keeping non-overlapping heads separate and training LoRA (Hu et al., 2022) adapters on top (Wortsman et al., 2022; Ilharco et al., 2023) (sensitivity to λ_m in §I). Both systems are trained independently on $\mathcal{D}_{\mathrm{sys1}}$ and $\mathcal{D}_{\mathrm{sys2}}$, sharing perception–language representations while keeping reasoning and control decoupled (Fig. 1).

4.2 System 2: failure-aware tree search

System 2 operates on a look-ahead tree $\mathcal{T} = (\mathcal{N}, \mathcal{E})$ in which each node is a 2D scene graph and each edge a verifiable subgoal inducing $n_k \xrightarrow{e_k} n_{k+1}$. It performs (i) edge proposal, (ii) successor-node prediction, (iii) leaf evaluation with V_θ, and (iv) search, returning a plan $\xi^{\mathrm{sel}} = (n_0, e_0, \ldots, n_K)$.

Edge and node generation. The LM head autoregressively samples an edge string and then the successor node string that the edge leads to, factorizing into token-level likelihoods $\hat{P}_\theta(e_k \mid n_k, z_{0:k-1}, \ell, s_0)$ and $\hat{P}_\theta(n_{k+1} \mid e_k, n_k, z_{0:k-1}, \ell, s_0)$. We use token-level beam search (Freitag & Al-Onaizan, 2017) to obtain diverse edge candidates and filter them with syntactic templates and scene-graph consistency checks (e.g. whether referenced objects exist in the current graph), and successor nodes are decoded greedily. Training uses the standard LM loss over high-level traces

Algorithm 1: VINE: batched tree search with learned V. Frontier leaves are scored by V_θ in place of rollouts. The full training and inference pipeline that calls this routine is Algorithm 2 (§D).

Input : Root n_0, batch B, width w, mix α, iterations M
Output: Plan (edge sequence) from the best frontier node

Initialize tree $\mathcal{T}=\{n_0\}$; for all n: $N(n)=0$, $W(n)=0$, $Q(n)=0$;
for $m=1$ **to** M **do**
  $\mathcal{C} \leftarrow$ frontier leaves; $\mathcal{B} \leftarrow$ Top-B of $\mathcal{C}$ by Q;
  foreach $n_p \in \mathcal{B}$ **do**
    Propose w edges $\{e_i\}$ (beam-search decoding); compute $S(n_p, e_i)=\alpha Q(n_p) + (1-\alpha)\hat{P}(e_i \mid n_p)$; keep the top-scoring pairs;
  For kept (n_p, e): create child n_c (greedy decoding); predict $V(n_c)$;
  For path nodes n_a to n_c: $W(n_a)+=V(n_c)$, $N(n_a)+=1$, $Q(n_a)=W(n_a)/N(n_a)$;
return plan traced from $\arg\max_{n \in \mathcal{C}} Q(n)$

sampled from $\mathcal{D}_{\text{sys2}}$:

$$\mathcal{L}_{\text{LM}} = -\mathbb{E}_{\mathcal{D}_{\text{sys2}}}\big[\log \hat{P}_\theta(e_k \mid n_k, z_{0:k-1}, \ell, s_0) + \log \hat{P}_\theta(n_{k+1} \mid e_k, n_k, z_{0:k-1}, \ell, s_0)\big].$$

Node evaluation from successes and failures. Each edge e_k yields a successor $n_{k+1} = \phi(s_{t_{k+1}})$, labelled success if $s_{t_{k+1}} \in \mathcal{G}$, failure if $s_{t_{k+1}} \in \mathcal{F}$, and continue otherwise. First-exit bootstrapping gives the target

$$y_k = \begin{cases} 1, & \text{terminal in } \mathcal{G} \ (e_k = \langle \texttt{done} \rangle), \\ 0, & \text{terminal in } \mathcal{F} \text{ at step } k{+}1, \\ \gamma\, V_{\theta'}(n_{k+1}, z_{0:k} \mid \ell, s_0), & \text{otherwise,} \end{cases}$$

with discount γ and an EMA target network θ'. By Proposition 1, V_θ then estimates $\Pr(\tau_{\mathcal{G}} < \tau_{\mathcal{F}} \mid n_k, z_{0:k-1}, \ell, s_0)$. Because the data is a fixed offline batch, symmetric TD regression propagates bootstrapped values into regions the logs never cover. We therefore use an asymmetric expectile loss (cf. IQL (Kostrikov et al., 2022)),

$$\mathcal{L}_{\text{val}} = \mathbb{E}_{\mathcal{D}_{\text{sys2}}}\big[L_2^{\tau_e}(y_k - V_\theta(n_k, z_{0:k-1} \mid \ell, s_0))\big], \qquad L_2^{\tau_e}(u) = \big|\tau_e - \mathbb{1}(u<0)\big|\, u^2,$$

with $\tau_e = 0.7$. Performance is stable for $\tau_e \in [0.6, 0.8]$ and degrades once the loss becomes symmetric at $\tau_e = 0.5$ (§I).

Search. With V_θ supplying per-node success-probability estimates we use a batched MCTS-style planner (Algorithm 1). In place of random rollouts, each frontier leaf is scored directly by V_θ. Each iteration selects promising frontier nodes by their running mean Q, proposes and filters w candidate edges, evaluates the resulting children, and backs the values up along the path. After M iterations the plan traced from the highest-valued leaf is returned. Scoring *cumulative* path value rather than the immediate next step distinguishes the search from a greedy planner, and the difference is largest under shift: greedy commits to locally attractive first steps that create downstream conflicts, reducing unseen success by 11 points at similar latency (§I).

4.3 System 1: Option Execution

System 1 grounds each symbolic edge into continuous control. Conditioned on $(s_t, \ell, e_k, n_{k+1})$ it produces the intra-option policy and the termination signal from a shared backbone, with the two experts attending to the backbone but not to each other. It is trained only on successful demonstrations, with the bounding-box coordinates inside n_{k+1} randomly perturbed within a small range so the controller tolerates the localization error present in System 2's generated scene graphs (§G).

The action expert is a flow-matching model over action chunks $\mathbf{A}_t = \{a_t, \ldots, a_{t+H}\}$, trained to match the oracle velocity along the interpolation path $\mathbf{A}_t^\eta = \eta \mathbf{A}_t + (1-\eta)\epsilon$. The done expert predicts the end-of-edge probability p_θ^{done} under a focal loss, and the two are combined as $\mathcal{L}_{\text{S1}} = \lambda_{\text{act}}\mathcal{L}_{\text{act}} + \lambda_{\text{done}}\mathcal{L}_{\text{done}}$ (losses written out in §C). At inference we integrate the learned flow from $\mathbf{A}_t^0 \sim \mathcal{N}(0, \sigma_0^2 I)$ to $\eta = 1$ to obtain the chunk $\widehat{\mathbf{A}}_t$, and advance to the next symbolic step when $p_\theta^{\text{done}} \geq \kappa_{\text{done}}$, and the option sequence terminates after the last edge ($k > K$).

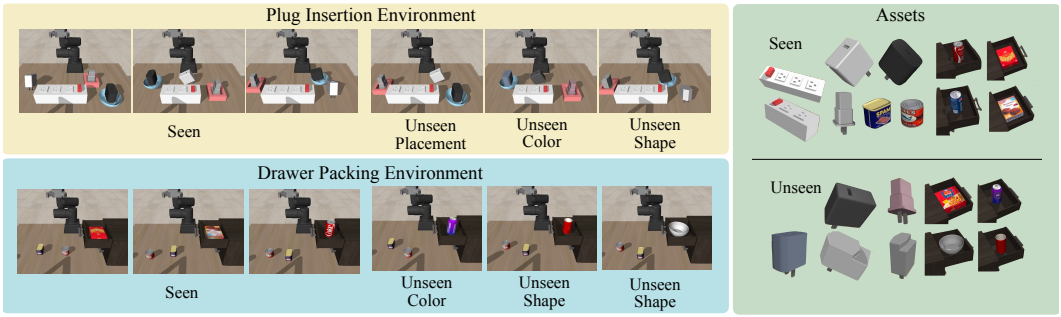


Figure 2: **Plug insertion and drawer packing environments.** We evaluate seen and unseen settings with the corresponding assets and collect mixed-quality demonstrations through a leader–follower teleoperation setup.

5 Experiments

We evaluate VINE across simulation, SIMPLER, and real-world manipulation. In simulation, we compare against flat and hierarchical VLAs and examine the contributions of planner training, tree search, and failure supervision (§5.2). On SIMPLER, we evaluate whether mixed-quality rollouts from post-trained policies can support further improvement (§5.3). Real-world experiments assess performance under appearance and object-configuration shifts across tasks with different planning horizons (§5.4). We report success rates separately for seen and unseen conditions, with task progress as an additional metric for long-horizon packing.

5.1 Simulation setup

Environments and data collection. We construct a MuJoCo (Todorov et al., 2012) suite with seen and unseen configurations to evaluate learning from successful and failed demonstrations (Fig. 2). *Plug insertion* includes 9 seen configurations varying placement, strip orientation, and socket count (2 or 3), with unseen placement, strip color, and charger shape. *Drawer packing* includes 8 seen configurations combining four placements and two distractor types, and 4 unseen configurations. Strategies include pushing the distractor aside, removing it, or leaving it in place. We collect 450 plug-insertion and 240 drawer-packing demonstrations, balanced across subgoal orderings and strategies, using joint-position control at 20 Hz. Success rates depend on the selected strategy and configuration, with per-path statistics in §F.

Baselines. We compare against four groups. **(i) Unified VLAs:** OpenVLA-OFT (Kim et al., 2025), GR00T N1.5 (Bjorck et al., 2025), π_0 (Black et al., 2024), a reward-conditioned π_0 (Yuan et al., 2023), and $\pi^*_{0.6}$-RECAP (Physical Intelligence et al., 2025), which applies advantage-weighted learning to the same backbone and demonstrations (its objective applied to our teleoperated data without policy rollouts). **(ii) Hierarchical VLA:** Hume (Song et al., 2025), which scores action chunks with a Cal-QL value and selects by Best-of-N. Its pretrained weights are unavailable, so it is trained from scratch. **(iii) VLM-as-System 2:** GPT-4o (Hurst et al., 2024) and Gemini-2.5-Flash (Comanici et al., 2025) replace our trained System 2 via few-shot prompting, with and without failure examples, *over the same scene-graph abstraction and with our System 1 fixed*. **(iv) Our variants:** VINE-Chain (no branching, success data only), VINE-Tree (tree search scored by LM confidence), and VINE-Full.

5.2 Simulation results

Comparison with baselines. Table 1 shows lower success on unseen configurations for flat policies: π_0 decreases from 0.689 to 0.267 on plug insertion, and OpenVLA-OFT from 0.244 to 0.044. Incorporating failure data at the action level provides limited gains in these evaluations. Reward-conditioned π_0 achieves 0.111 unseen success on plug insertion, while $\pi^*_{0.6}$-RECAP, which uses the same backbone and demonstrations as our method, achieves 0.244. VINE-Full reaches 0.422 on unseen plug insertion and 0.675 on unseen drawer packing, approximately 1.7× and 1.2× the cor-

Table 1: **Success rate on plug insertion and drawer packing.** "Failure data" marks methods that consume failed demonstrations. Avg. is the success rate over all seen and unseen trials (plug: 45/45, drawer: 80/40). Best in bold, second best underlined.

Model	Failure data	Plug Insertion			Drawer Packing		
		Seen	Unseen	Avg.	Seen	Unseen	Avg.
Unified VLA							
OpenVLA-OFT (Kim et al., 2025)	×	0.244	0.044	0.144	0.638	0.275	0.517
GR00T N1.5 (Bjorck et al., 2025)	×	0.422	0.244	0.333	0.700	<u>0.600</u>	0.667
π_0 (Black et al., 2024)	×	0.689	0.267	0.478	0.725	0.575	0.675
π_0 + Reward Cond. (Yuan et al., 2023)	✓	0.489	0.111	0.300	0.550	0.425	0.508
$\pi^*_{0.6}$-RECAP (Physical Intelligence et al., 2025)	✓	0.711	0.244	0.478	<u>0.750</u>	0.575	<u>0.692</u>
Hierarchical VLA							
Hume (Song et al., 2025)	✓	0.356	0.156	0.256	0.650	0.500	0.600
VLM as System 2 (same abstraction, our System 1)							
GPT-4o (Hurst et al., 2024)	×	0.733	0.311	<u>0.522</u>	0.650	0.475	0.592
GPT-4o (Hurst et al., 2024)	✓	0.711	<u>0.333</u>	<u>0.522</u>	0.738	0.575	0.683
Gemini-2.5-Flash (Comanici et al., 2025)	×	<u>0.756</u>	0.200	0.478	0.638	0.450	0.575
Gemini-2.5-Flash (Comanici et al., 2025)	✓	0.711	0.289	0.500	0.713	0.525	0.650
Ablations of our System 2							
VINE-Chain	×	0.733	0.244	0.489	0.700	0.450	0.617
VINE-Tree	×	0.711	0.289	0.500	0.738	0.525	0.667
VINE-Full (ours)	✓	**0.800**	**0.422**	**0.611**	**0.800**	**0.675**	**0.758**

responding RECAP results. These comparisons support subgoal-level plan evaluation as an effective way to use failure experience in the evaluated tasks.

Planner comparisons and ablations. The VLM-as-System 2 baselines use the same scene-graph abstraction and System 1 controller, allowing comparison of prompted and trained planners with these components held fixed. GPT-4o with failure examples achieves 0.333 unseen success on plug insertion, compared with 0.422 for VINE-Full. This difference supports the benefit of our trained planner, although it does not isolate the contribution of the reach–avoid value from other differences between the planners. In-context failure examples also improve GPT-4o's unseen drawer-packing success from 0.475 to 0.575. Within our variants, replacing chain generation with LM-confidence-guided tree search increases unseen drawer-packing success from 0.450 (VINE-Chain) to 0.525 (VINE-Tree). Using the failure-trained value for search further increases unseen success by 0.133 on plug insertion and 0.150 on drawer packing. These results suggest that exploring alternative plans and evaluating them using failure experience both contribute to performance. Fig. 4(a) illustrates a planning tree, and §H compares execution trajectories with the baselines.

Test-time scaling. Because the planner separates deliberation from control, the reasoning budget can be set at deployment time. With parameters fixed, we vary the per-step expansion width w on unseen plug insertion (Fig. 3): unseen success rises $28.9 \rightarrow 37.8 \rightarrow 42.2 \rightarrow 44.4$ for $w = 1..4$ with near-linear latency growth ($23.9 \rightarrow 30.8$ s, Fig. 18) and drops at w=5 (40.0). Additional search thus improves success up to w=4, and we use w=3 by default as a trade-off with latency. Scene-graph decoding takes 79% of the cost, search only 4.6% (§I).

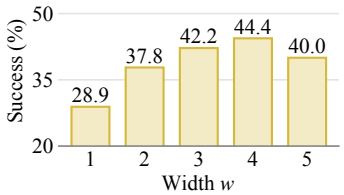


Figure 3: **Test-time scaling.**

Scene-graph fidelity. The scene graphs generated by our trained System 2 recover object relations accurately (relation accuracy 0.85 over $n = 210$ trials), and subgoal-level metrics correlate more strongly with success ($r = 0.29$) than node-level ones ($|r| \leq 0.19$). Predicted successors remain consistent with post-execution states (relation accuracy 0.83). The value ranks plans from identities and relations, which are stable under small coordinate shifts. By contrast, a misselected subgoal leads execution into a branch that downstream accuracy cannot correct. This evidence is observational, not a controlled corruption study. Appendix G gives the full analysis.

5.3 Bootstrapping post-trained policies on SIMPLER

We evaluate whether failures collected from a post-trained policy's own rollouts can support further improvement on SIMPLER (Li et al., 2024). We use π_0 policies post-trained on Bridge v2 for WidowX and Fractal for the Google Robot. For each policy, we collect successful and failed

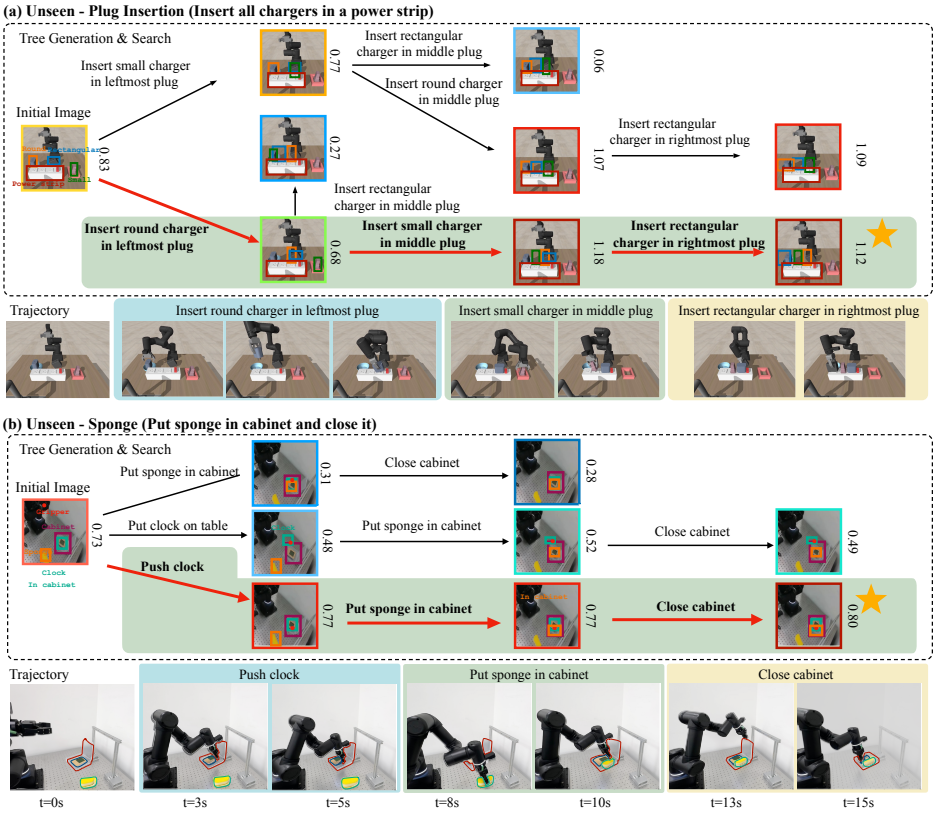


Figure 4: **Qualitative planning and execution in simulation and the real world.** System 2 builds a failure-aware planning tree with reach–avoid values and System 1 executes the selected subgoals. **(a)** Plug insertion, unseen color (simulation). Grasp subgoals are omitted from the tree for clarity (each task involves 6 subgoal steps, 3 grasp + 3 insert). **(b)** Sponge–unseen with a clock distractor (real robot): the planner assigns the clock-grasping branch a low value and routes through *push clock* $\rightarrow$ *put sponge* $\rightarrow$ *close*. Node values are raw value-head outputs (unclipped), so they can exceed 1.

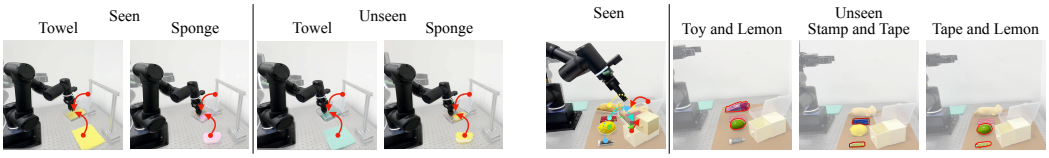
Table 2: **SIMPLER benchmark.** We report the single WidowX (Bridge v2) setting and both Google Robot (Fractal) protocols, VisualMatching (VM) and VariantAggregation (VA). Coke = pick coke can, Near = move near, Drw. = open/close drawer, D+A = open drawer & place apple. Gains concentrate under VA, the distribution-shift protocol.

	WidowX (Bridge v2)					Google Robot (VM)					Google Robot (VA)				
Method	carrot	eggplant	cube	spoon	Avg.	Coke	Near	Drw.	D+A	Avg.	Coke	Near	Drw.	D+A	Avg.
π_0	0.236	**0.791**	0.139	0.388	0.389	0.730	0.525	0.704	0.108	0.517	0.476	0.368	0.402	0.090	0.334
VINE (ours)	**0.375**	0.708	**0.145**	**0.395**	**0.406**	**0.770**	**0.567**	**0.709**	**0.213**	**0.565**	**0.528**	**0.530**	**0.546**	**0.126**	**0.433**

rollout trajectories, fine-tune with our failure-aware objective, and merge the adapted weights. Each edge specifies a subgoal and a 2D end-effector position in the image, allowing search over grasp locations within a fixed subgoal sequence. On WidowX, average success increases from 0.389 to 0.406, although success on *eggplant in basket* decreases from 0.791 to 0.708. On the Google Robot, average success increases from 0.517 to 0.565 under VisualMatching and from 0.334 to 0.433 under VariantAggregation, with relative gains of 44.0% on *move near* and 35.8% on *open/close drawer* under the latter protocol. These results suggest that mixed-quality rollouts from post-trained policies, including failed attempts, can support further policy improvement, with larger gains under visual distribution shift.

5.4 Real-world deployment

We deploy VINE on a 6-DoF arm at 20 Hz using joint-position actions and wrist and third-person cameras (§F). Two task groups with increasing horizons evaluate generalization under distribution



(a) Cabinet packing tasks. (b) Long-horizon packing task.

Figure 5: **Real-world tasks.** **(a)** Cabinet packing with distractors and folding. **(b)** Long-horizon packing, seen (left) and unseen object-pair configurations (right).

Table 3: **Real-world results. Cabinet packing:** $\sim$3 subgoal steps, 100 demos, success rate. **Long-horizon packing:** 4 objects, $\sim$10 subgoal steps, 240 demos. Progress is normalized to $[0, 1]$ and unseen columns average three unseen configurations.

	Cabinet packing ($\sim$3 subgoals)				Long-horizon packing ($\sim$10 subgoals)			
	Seen		Unseen		Progress		Success rate	
Method	sponge	towel	sponge	towel	Seen	Unseen	Seen	Unseen
π_0 (Black et al., 2024)	0.60	0.45	0.55	0.30	0.600	0.422	0.150	0.067
VINE (ours)	**0.75**	**0.55**	**0.65**	**0.55**	**0.683**	**0.689**	**0.300**	**0.367**

shift (Fig. 5). Cabinet packing requires $\sim$3 subgoals and evaluates color generalization. Long-horizon packing requires placing four objects into a cabinet and a drawer over approximately 10 subgoals and evaluates generalization to unseen object pairs. Both use teleoperated demonstrations including failed attempts, with π_0 as the baseline (Table 3).

Cabinet packing ($\sim$3 subgoals, 100 demonstrations). The task requires placing an object into a cabinet and closing its lid. Sponge placement involves a clock distractor that can be pushed aside, removed, or left in place, while towel placement may require folding. We evaluate generalization to unseen colors. VINE outperforms π_0 across all cabinet-packing conditions (Table 3, left), with unseen towel success increasing from 0.30 to 0.55. Fig. 4(b) illustrates failure-aware plan selection. The clock's width is close to the gripper's, making reliable grasping difficult, although this geometric constraint is absent from the text-serialized scene graph. The learned value assigns a low score to the clock-grasping branch, consistent with failures observed during collection, and the planner selects an alternative. This example suggests that failure supervision can inform feasibility estimates despite omitted physical details, although generalization to constraints unobserved during training remains unverified.

Long-horizon packing ($\sim$10 subgoals, 240 demonstrations). The task requires packing four objects (toy, stamp, lemon, tape) into two containers and closing the lids. Demonstrations cover 12 subgoal orderings (20 each), varying the first object, stamp side, and subsequent order. Evaluation uses three unseen object-pair configurations. VINE uses w=2 and a fixed batch B=5, requiring approximately 74 s for planning. Table 3 (right) reports the results. π_0's progress decreases from 0.600 on seen configurations to 0.422 on unseen configurations, while VINE maintains comparable progress (0.683–0.689). Unseen full-task success reaches 36.7%, compared with 6.7% for π_0, a 5.5$\times$ improvement. In evaluated trials, VINE avoids orderings that place the lemon before the tape, which have 0% training success due to physical obstruction. This observation suggests that the value helps avoid infeasible sequences even with $\sim$10 subgoals.

6 Conclusion

We introduced **VINE**, a hierarchical VLA that learns subgoal-level plan evaluation from successful and failed demonstrations. By formulating plan evaluation as a reach–avoid problem, VINE learns a feasibility value that guides tree search over predicted scene-graph transitions. The planner and controller share a language backbone and are trained offline for plan selection and subgoal execution, respectively. Experiments across simulation, SIMPLER, and real-world manipulation show improved task success, with larger gains under distribution shift. These results highlight the value of failed demonstrations as supervision for planning in hierarchical VLAs.

AI USE STATEMENT

In this work, we used generative AI tools to support research execution, to assist in writing the proof of Proposition 1, to draft sections of the paper, and to search for related work. For execution, LLMs helped implement and debug parts of the training, tree-search, and evaluation code and helped format experimental results into tables and figures. All such code was reviewed and tested by the authors, and every reported number comes from runs the authors executed. For the proof, LLMs helped write out and check the derivation in §B, and the authors formulated the claim and verified each step by hand. LLMs were also used to identify and summarize relevant literature, whose citations the authors checked against the original papers, and to draft and polish text for readability. We have reviewed all AI-assisted work and take full responsibility for the final content of this paper, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All architectural details, losses, and search hyperparameters are given in §4 and §M. The datasets, environment configurations, and seen/unseen splits are described in §5.1 and §F, and the proof of Proposition 1 is given in §B. Code, the simulation environment suite, and the mixed-quality demonstration datasets will be released upon publication.

REFERENCES

Christopher Agia, Rohan Sinha, Jingyun Yang, Ziang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *Proc. of the 8th Annual Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=yqLFb0RnDW.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. PaliGemma: A versatile 3B VLM for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. In *Proc. of the Robotics: Science and Systems (RSS)*, 2023.

Xiaowei Cai, Yunuo Cai, Bingao Chen, Jingxiao Chen, Zhi Chen, Siyuan Feng, et al. τ_0-VLA: a hierarchical robot foundation model with world-model-guided test-time computation. *arXiv preprint arXiv:2608.16885*, 2026.

Hanzhi Chen, Anran Zhang, Simon Schaefer, Kejia Chen, Shi Chen, Daniel Cremers, Oier Mees, and Stefan Leutenegger. Robot self-improvement via human-video dynamics models. *arXiv preprint arXiv:2606.21406*, 2026.

Hao Chen, Jiaming Liu, Chenyang Gu, Zhuoyang Liu, Renrui Zhang, Xiaoqi Li, Xiao He, Yandong Guo, Chi-Wing Fu, Shanghang Zhang, and Pheng-Ann Heng. Fast-in-slow: A dual-system VLA model unifying fast manipulation within slow reasoning. In *Proc. of the Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=4asFznbzJq.

Yizhou Chi, Kevin Yang, and Dan Klein. ThoughtSculpt: Reasoning with intermediate revision and search. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7685–7711, Albuquerque, New

Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.428. URL https://aclanthology.org/2025.findings-naacl.428/.

Sungjoon Choi, Kyungjae Lee, and Songhwai Oh. Robust learning from demonstrations with mixed qualities using leveraged Gaussian processes. *IEEE Transactions on Robotics*, 35(3):564–576, 2019. doi: 10.1109/TRO.2019.2891173.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Jaime F Fisac, Mo Chen, Claire J Tomlin, and S Shankar Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proc. of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pp. 11–20, 2015.

Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025.

Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In Thang Luong, Alexandra Birch, Graham Neubig, and Andrew Finch (eds.), *Proc. of the First Workshop on Neural Machine Translation*, pp. 56–60. Association for Computational Linguistics, August 2017.

Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu, Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie Wen. Interpretable contrastive Monte Carlo tree search reasoning. *arXiv preprint arXiv:2410.01707*, 2024.

Huy Hoang, Tien Anh Mai, and Pradeep Varakantham. SPRINQL: Sub-optimal demonstrations driven offline imitation learning. In *Proc. of the Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=uDD44NROOt.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Yu-Chiang Frank Wang, and Fu-En Yang. ThinkAct: Vision-language-action reasoning via reinforced visual latent planning. *arXiv preprint arXiv:2507.16815*, 2025.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2023.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proc. of the 8th Annual Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=ZMnD6QZAE6.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Yann Labbé, Sergey Zagoruyko, Igor Kalevatykh, Ivan Laptev, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Monte-Carlo tree search for efficient visually guided rearrangement planning. *IEEE Robotics and Automation Letters*, 5(2):3715–3722, 2020.

Daniel T. Larsson, Dipankar Maity, and Panagiotis Tsiotras. Q-tree search: An information-theoretic approach toward hierarchical abstractions for agents with computational limitations. *IEEE Transactions on Robotics*, 36(6):1669–1685, 2020. doi: 10.1109/TRO.2020.3003219.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, et al. MolmoAct: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Proc. of the 8th Annual Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=LZh48DTg71.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: Hierarchical action models for open-world robot manipulation. In *Proc. of the Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=h7aQxzKbq6.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

Chuhao Liu, Zhijian Qiao, Jieqi Shi, Ke Wang, Peize Liu, and Shaojie Shen. SG-Reg: Generalizable and efficient scene graph registration. *IEEE Transactions on Robotics*, 41:3870–3889, 2025. doi: 10.1109/TRO.2025.3577020.

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. Don't throw away your value model! Generating more preferable text with value-guided Monte-Carlo tree search decoding. In *Proc. of the First Conference on Language Modeling (CoLM)*, 2024a. URL https://openreview.net/forum?id=kh9Zt2Ldmn.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *Proc. of the European Conference on Computer Vision (ECCV)*, pp. 38–55. Springer, 2024b.

Yanbo Mao, Jianlong Fu, Ruoxuan Zhang, Hongxia Xie, and Meibao Yao. Beyond success: Refining elegant robot manipulation from mixed-quality data via just-in-time intervention. *arXiv preprint arXiv:2511.22555*, 2025.

Kostas Margellos and John Lygeros. Hamilton–Jacobi formulation for reach–avoid differential games. *IEEE Transactions on Automatic Control*, 56(8):1849–1861, 2011.

Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. In *Proc. of the 8th Annual Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=6FGlpzC9Po.

Zhe Ni, Xiaoxin Deng, Cong Tai, Xinyue Zhu, Qinghongbing Xie, Weihang Huang, Xiang Wu, and Long Zeng. GRID: Scene-graph-based instruction-driven robotic task planning. In *Proc. of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13765–13772. IEEE, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, 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. In *Proc. of the Robotics: Science and Systems (RSS)*, Delft, Netherlands, 2024.

Open X-Embodiment Collaboration. Open X-Embodiment: Robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864, 2023.

Physical Intelligence. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: A VLA that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

E. Plaku, K.E. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005. doi: 10.1109/TRO.2005.847599.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding large language models using 3d scene graphs for scalable robot task planning. In *Proc. of The 7th Conference on Robot Learning (CoRL)*, pp. 23–72, 2023.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *Proc. of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317.

Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, Guanghui Ren, Maoqing Yao, Bin Zhao, et al. Hume: Introducing system-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Zhihao Wang, Jianxiong Li, Yu Cui, Yuan Gao, Xianyuan Zhan, Junzhi Yu, and Xiao Ma. World value models for robotic manipulation. *arXiv preprint arXiv:2606.24742*, 2026.

Junjie Wen, Yichen Zhu, Minjie Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. DiffusionVLA: Scaling robot foundation models via unified diffusion and autoregression. In *Proc. of the Forty-second International Conference on Machine Learning (ICML)*, 2025.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proc. of the 39th International Conference on Machine Learning (ICML)*, pp. 23965–23998. PMLR, 2022.

Rushuai Yang, Hecheng Wang, Zhichao Wu, Chiming Liu, et al. ALOE: Action-level off-policy evaluation for vision-language-action model post-training. *arXiv preprint arXiv:2602.12691*, 2026a.

Yanting Yang, Shenyuan Gao, Qingwen Bu, Li Chen, and Dimitris N. Metaxas. Seeing farther and smarter: Value-guided multi-path reflection for VLM policy optimization. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2026b. arXiv:2602.19372.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Proc. of the Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2023b.

Anna Yershova and Steven M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007. doi: 10.1109/TRO.2006.886840.

Hui Yuan, Kaixuan Huang, Chengzhuo Ni, Minshuo Chen, and Mengdi Wang. Reward-directed conditional diffusion: Provable distribution estimation and reward improvement. In *Proc. of the Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/forum?id=58HwnnEdtF.

Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning. In *Proc. of the 8th Annual Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=S70MgnIA0v.

Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Fuxian Huang, Haoran Zhang, Ming Zhou, Shengzhe Zhang, Litao Liu, Sixu Lin, and Jiangmiao Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint arXiv:2509.15937*, 2025.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Proc. of the Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=8rcFOqEud5.

Hongyin Zhang, Zifeng Zhuang, Han Zhao, Pengxiang Ding, Hongchao Lu, and Donglin Wang. ReinboT: Amplifying robot visual-language manipulation with reinforcement learning. In *Proc. of the Forty-second International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=Mzz4BhdIFb.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. CoT-VLA: Visual chain-of-thought reasoning for vision-language-action models. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1702–1713, 2025.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Proc. of the Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/forum?id=Wjp1AYB8lH.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proc. of The 7th Conference on Robot Learning (CoRL)*, pp. 2165–2183, 2023.

Appendix

The appendix is organized as follows.

- §A walks through the concept figure and the inference pipeline.
- §B gives the detailed node–edge formulation and the proof of Proposition 1.
- §C writes out the System 1 training objectives.
- §D lists the full training and inference procedure (Algorithm 2).
- §E describes how training scene graphs and failure labels are annotated offline.
- §F reports dataset statistics for every environment and the real-world setup.
- §G measures the fidelity of the scene graphs generated by the trained System 2.
- §H shows additional qualitative results.

- §I reports ablations on search, latency, pruning, weight merging, and the expectile.
- §J decomposes failure cases by type.
- §K discusses the limitations of VINE.
- §L presents uncertainty-triggered replanning.
- §M lists all hyperparameters.

A Concept and inference pipeline

Why failure data matters. Fig. 6 contrasts VINE with a flat VLA on plug insertion. Teleoperated collection naturally yields both successful and failed trajectories. The failed ones record, for example, insertion orders in which an already-inserted charger blocks the next one. A standard VLA is post-trained only on the successful subset, so it has no signal that separates a plausible insertion order from a feasible one. It can commit to an ordering that collides downstream, as in the failure trajectory in the figure. VINE trains its own System 2 planner on *both* subsets. Successes and failures supervise a reach–avoid value over subgoal transitions (§4), and at test time this value ranks candidate plans before any motion is executed. Failure-prone branches thus receive low values and are pruned, and the robot follows an ordering that leaves room for every charger.

Plan, then execute. Fig. 7 traces one inference episode. System 2 is the fine-tuned high-level head of VINE, trained on $\mathcal{D}_{\text{sys2}}$ with the LM and value losses rather than a prompted external model. From the initial image and instruction it emits the root scene graph. It then expands a look-ahead tree: at each node it proposes candidate subgoals (edges), imagines the successor scene graph each would produce, and scores that successor with its reach–avoid value V_θ. Branches that begin with the rectangular charger receive low values $(0.3,\ 0.2)$ because such orderings failed in the training data. The path that inserts the small charger first reaches $V_\theta = 1.0$ and is selected. The search runs once, before the first action. System 1 then executes the chosen edges in order. Each edge is conditioned on the subgoal text and the predicted next scene graph, and the policy advances when the done expert signals completion. Algorithm 2 (§D) gives the same loop in pseudocode.

B Detailed formulation and proof of Proposition 1

B.1 Node–edge kernel induced by low-level options

We define the high-level node–edge space from the low-level MDP $(\mathcal{S}, \mathcal{A}, T)$ and show that it forms an MDP, then introduce the node-level target/avoid sets corresponding to $\mathcal{G}, \mathcal{F} \subset \mathcal{S}$.

Given the option grounding $o_e = (I_e^N, \pi_e, \beta_e, \Pi_e)$, the closed-loop one-step kernel under o_e is $P_{\pi_e}(s' \mid s) = \sum_{a\in\mathcal{A}} \pi_e(a \mid s, \ell) T(s' \mid s, a)$. With termination-on-arrival, the terminal low-level state distribution is

$$P_{o_e}^S(s^\star \mid s) = \sum_{t\geq 0} \big((1-\beta_e)P_{\pi_e}\big)^t (\beta_e P_{\pi_e})(s^\star \mid s),$$

i.e. the probability that, starting from s and executing o_e, execution eventually terminates in $s^\star$ under β_e. Lift the high-level pair (n, z) to a low-level *arrival* distribution $\nu(\cdot \mid n, z)$, the conditional law of s upon arriving at node n after the option sequence encoded in z (in data, $\nu = \delta_{s_{t_k}}$ with $\phi(s_{t_k}) = n$). Mapping terminal states to nodes via Π_e yields the node-level option kernel with context

$$P_{o_e}^N(n' \mid n, z) = \mathbb{E}_{s\sim\nu(\cdot\mid n,z)}\Big[\sum_{s^\star} P_{o_e}^S(s^\star \mid s)\, \mathbf{1}\{\Pi_e(s^\star) = n'\}\Big].$$

With the deterministic context update $z' = [z, e, n]$, the node–edge kernel is $P((n', z') \mid (n, z), e) = P_{o_e}^N(n' \mid n, z)\mathbf{1}\{z' = [z, e, n]\}$. Defining $\mathcal{G}_N := \phi(\mathcal{G})$ and $\mathcal{F}_N := \phi(\mathcal{F})$ with $\mathcal{G}_N \cap \mathcal{F}_N = \varnothing$, the kernel is absorbing on $\mathcal{G}_N \cup \mathcal{F}_N$. Hence $(\mathcal{X}, \mathcal{U}, P)$ with $\mathcal{X} = \mathcal{N} \times \mathcal{Z}$ and $\mathcal{U} = \mathcal{E}$ is a well-defined MDP on the node–edge space.

B.2 Value equals reachability probability (undiscounted first exit)

Define the decision-epoch hitting times $\tau_{\mathcal{G}} := \inf\{t \geq 0 : n_t \in \mathcal{G}_N\}$, $\tau_{\mathcal{F}} := \inf\{t \geq 0 : n_t \in \mathcal{F}_N\}$, and $\tau := \tau_{\mathcal{G}} \wedge \tau_{\mathcal{F}}$. With the entrance reward $r(n, z, e, n') = \mathbf{1}\{n' \in \mathcal{G}_N\}$ and the undiscounted

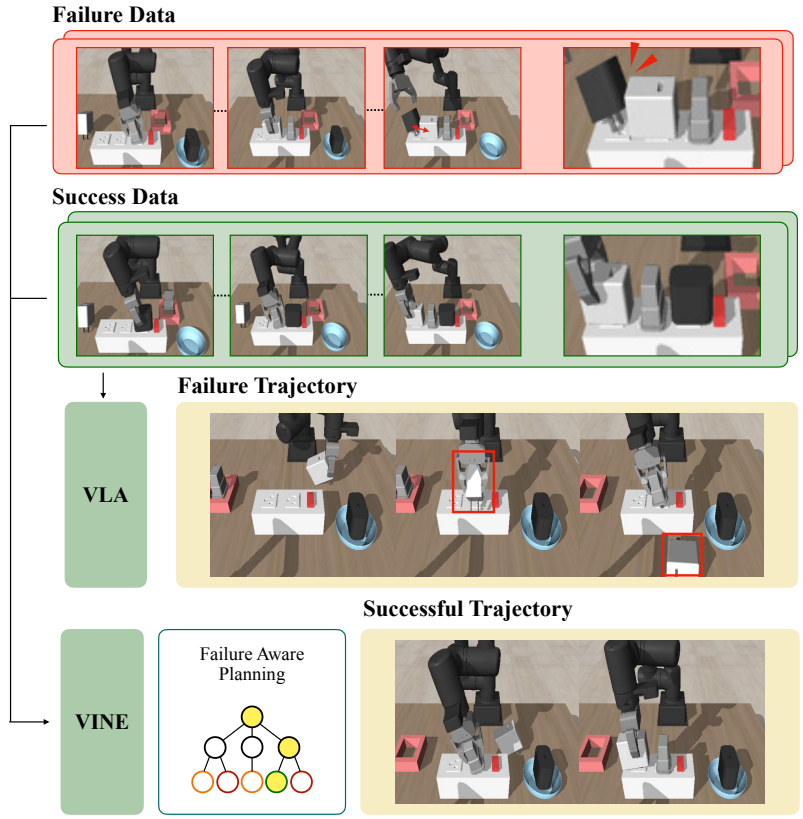


Figure 6: **Concept.** Standard VLAs trained only on success data may produce infeasible trajectories. VINE trains its System 2 planner on both success and failure trajectories, and plans with the learned failure-aware value, yielding more robust executions.

first-exit return $R := \sum_{t=0}^{\tau-1} r(n_t, z_t, e_t, n_{t+1})$, we have the pathwise identity $R = \mathbf{1}\{\tau_{\mathcal{G}} < \tau_{\mathcal{F}}\}$: exactly one nonzero reward can occur, on the unique transition that first enters $\mathcal{G}_N$. If $\mathcal{F}_N$ is hit first, all rewards are zero and the episode terminates. Therefore, for any policy μ on the node–edge MDP,

$$V^{\mu}(n,z) := \mathbb{E}_{\mu}\left[\sum_{t=0}^{\tau-1} r(n_t, z_t, e_t, n_{t+1}) \;\middle|\; (n_0, z_0) = (n, z)\right] = \Pr_{\mu}\big(\tau_{\mathcal{G}} < \tau_{\mathcal{F}} \mid n, z\big),$$

which is Proposition 1. The corresponding Bellman relations are the standard reach–avoid equations,

$$V^{\mu}(n,z) = \sum_{e} \mu(e \mid n, z) \sum_{n'} P^{N}_{o_e}(n' \mid n, z)\Big[\mathbf{1}\{n' \in \mathcal{G}_N\} + \mathbf{1}\{n' \notin \mathcal{G}_N \cup \mathcal{F}_N\} V^{\mu}(n', [z, e, n])\Big],$$

with boundary conditions $V^{\mu}(n,z) = 1$ for $n \in \mathcal{G}_N$ and $V^{\mu}(n,z) = 0$ for $n \in \mathcal{F}_N$. The optimality version replaces the inner expectation by a maximization over $e \in \mathcal{O}(n,z)$. With a discount $\gamma \in (0,1)$ the pathwise return satisfies $R_{\gamma} = \gamma^{\tau_{\mathcal{G}}-1}\mathbf{1}\{\tau_{\mathcal{G}} < \tau_{\mathcal{F}}\}$, so the discounted value equals the success probability up to a depth-dependent factor. At approximately fixed search depth this factor is near-constant and the ranking by reachability is preserved.

B.3 Expectile losses on the node–edge MDP

Let $\mathcal{D} = \{(n_k, z_{0:k-1}, e_k, n_{k+1})\}$ be logged transitions from a behavior policy $\mu_b(e \mid n, z)$, with entrance reward $r_k = \mathbf{1}\{n_{k+1} \in \mathcal{G}_N\}$ and terminal flag $\mathrm{term}_k = \mathbf{1}\{n_{k+1} \in \mathcal{G}_N \cup \mathcal{F}_N\}$. From the Bellman equation under μ_b we form the one-step target $y_k = r_k + \gamma(1 - \mathrm{term}_k)V_{\theta'}(n_{k+1}, z_{0:k})$ and minimize $\mathcal{L}_{\mathrm{val}}(\theta) = \mathbb{E}_{\mathcal{D}}[L_2^{\tau_e}(y_k - V_{\theta}(n_k, z_{0:k-1}))]$ with $L_2^{\tau_e}(u) = |\tau_e - \mathbf{1}\{u < 0\}|u^2$. Symmetric-MSE TD is well suited to online RL, where fresh samples correct bootstrap error. Our setting is offline, so training on a fixed batch reduces the TD error only on the logged support and bootstrapped

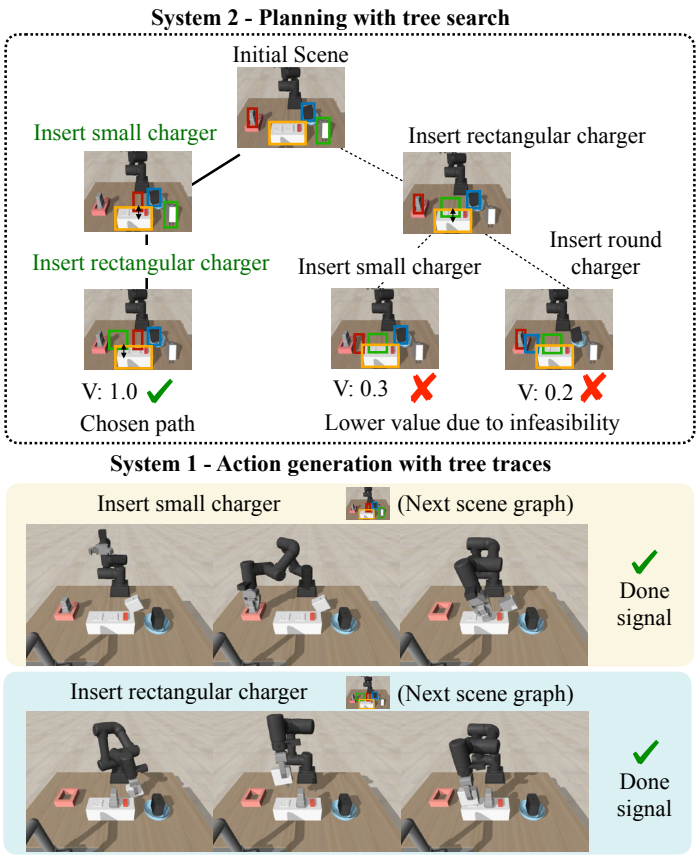


Figure 7: **Inference-time pipeline.** The trained System 2 selects the highest-value subgoal path via tree search over generated scene graphs, and System 1 sequentially executes each edge, advancing on a done signal.

propagation beyond that support is unreliable under a symmetric loss. Under our residual $u = y_k - V_\theta$, $\tau_e > 0.5$ weights positive residuals more heavily, and $\tau_e = 0.7$ performs best in our experiments (§I).

C System 1 Training Objectives

The action expert is a flow-matching model over action chunks $\mathbf{A}_t = \{a_t, \ldots, a_{t+H}\} \in \mathbb{R}^{H \times d_a}$. With interpolation time $\eta \in [0, 1]$, Gaussian noise ϵ, and $\mathbf{A}_t^\eta = \eta \mathbf{A}_t + (1 - \eta)\epsilon$, it predicts the flow $\dot{\mathbf{A}}_\theta$ and matches the oracle velocity $\mathbf{u}(\mathbf{A}_t^\eta \mid \mathbf{A}_t)$:

$$\mathcal{L}_{\text{act}} = \mathbb{E}_{\mathcal{D}_{\text{sys1}}} \left\| \dot{\mathbf{A}}_\theta(\mathbf{A}_t^\eta, s_t, \ell, e_k, n_{k+1}) - \mathbf{u}(\mathbf{A}_t^\eta \mid \mathbf{A}_t) \right\|_2^2.$$

The done expert predicts the end-of-edge probability $p_\theta^{\text{done}} = \beta_\theta(s_t, \ell, e_k, n_{k+1})$ under a focal loss

$$\mathcal{L}_{\text{done}} = \mathbb{E}_{\mathcal{D}_{\text{sys1}}} \left[-\alpha_d (1 - p_\theta^{\text{done}})^{\gamma_d} \log p_\theta^{\text{done}} \right],$$

and the joint objective is $\mathcal{L}_{\text{S1}} = \lambda_{\text{act}} \mathcal{L}_{\text{act}} + \lambda_{\text{done}} \mathcal{L}_{\text{done}}$.

D Training and Inference Pipeline

Algorithm 2 lists the complete procedure: the two offline training stages of §4 and the plan–execute loop at inference. The tree search it calls in line 8 is Algorithm 1 in §4, and §A illustrates the same inference loop.

Algorithm 2: VINE: training and inference. Perception, planning, and control all run inside one model at test time, and TREESEARCH is Algorithm 1.

Input : Scene s_0, instruction ℓ, datasets $\mathcal{D}_{\text{sys1}}$, $\mathcal{D}_{\text{sys2}}$
Output: Executed trajectory

`/* Training (offline) */`
Train System 2 on $\mathcal{D}_{\text{sys2}}$ (success + failure);
  LM head: edge/node generation ($\mathcal{L}_{\text{LM}}$);
  Value head: reach–avoid value ($\mathcal{L}_{\text{val}}$, expectile τ_e);
Train System 1 on $\mathcal{D}_{\text{sys1}}$ (success only);
  Action expert: flow matching ($\mathcal{L}_{\text{act}}$);
  Done expert: subgoal completion ($\mathcal{L}_{\text{done}}$, focal loss);

`/* Inference */`
Construct scene graph $n_0 \leftarrow \phi(s_0)$;
$\xi^{\text{sel}} \leftarrow$ TREESEARCH(n_0, ℓ, s_0) ; `// Alg. 1`
foreach (e_k, n_{k+1}) *in* ξ^{sel} **do**
  repeat
    $\widehat{\mathbf{A}}_t \leftarrow \pi_\theta(\cdot \mid s_t, \ell, e_k, n_{k+1})$; `// action chunk`
    Execute $\widehat{\mathbf{A}}_t$ at 20 Hz; observe s_{t+H};
    $p_\theta^{\text{done}} \leftarrow \beta_\theta(s_{t+H}, \ell, e_k, n_{k+1})$;
  until $p_\theta^{done} \geq \kappa_{done}$;

E ANNOTATING THE ABSTRACTION AND THE FAILURE LABELS

All annotation steps in this section are performed *once, offline* to build $\mathcal{D}_{\text{sys2}}$ and $\mathcal{D}_{\text{sys1}}$. The annotation tools below (a detector and a VLM) only produce training targets and are never used as the planner. System 2 is a model we train on these targets with the LM and value losses of §4, and none of the annotation pipeline runs at test time. The deployed system is a single VINE model that emits the root scene graph, every imagined successor, and their values from raw images and the instruction, with no detector and no external service in the loop. Appendix G therefore measures the model's own abstraction rather than a perception front-end.

Keyframes and edges. A trajectory is segmented at consistent gripper-state events: `open`→`close` opens a `grasp`, `close`→`open` ends a `grasp`/`insert`, and `open`→`open` follows a `place`/`push`. Each interval $[t_k, t_{k+1})$ becomes one edge, and the trajectory becomes a subgoal chain aligned with the instructed order (e.g. `grasp-small charger` → `insert-small charger in rightmost plug` → $\cdots$).

Node serialization. Around each transition we take a short image window and use its central frame as the scene, then build a 2D scene graph with three ingredients. The **gripper node** is the measured 3D pose projected into the camera frame using known intrinsics and extrinsics. **Object nodes** come from Grounding DINO (Liu et al., 2024b) box proposals, with a VLM (Gemini-2.5-Flash (Comanici et al., 2025)) assigning semantic names to the crops under a task-vocabulary prompt. **Relations** are coarse spatial predicates inferred from box geometry and, when ambiguous, from the same VLM. Each subgoal thus yields an {image, scene graph, label} triple. These triples are the supervision targets from which System 2 learns to generate scene graphs itself.

Failure labelling protocol. Because a language instruction is abstract, several execution paths satisfy it, and each decomposes into a subgoal sequence. During collection the teleoperator follows a prescribed sequence (e.g. `grasp-spam` → `put-spam-in-drawer` → `close-drawer`) and can be viewed as a policy conditioned on it. A transition is labelled *failure* by a reachability criterion: it fails when the operator cannot reach the designated success set $\mathcal{G}$ within a prescribed time window. Two properties follow. The label is *localized* to the subgoal at which feasibility broke down rather than spread over the whole trajectory, which provides denser planner supervision than a single trajectory-level label. And it depends only on *whether* the physical outcome was achieved, never on *why*, which reduces the need for annotator judgement but does not capture action-level causes such as grasp slippage (§K). This corresponds directly to the first-exit objective: $V_\theta(n, z) \approx \Pr(\tau_{\mathcal{G}} < \tau_{\mathcal{F}} \mid n, z)$.

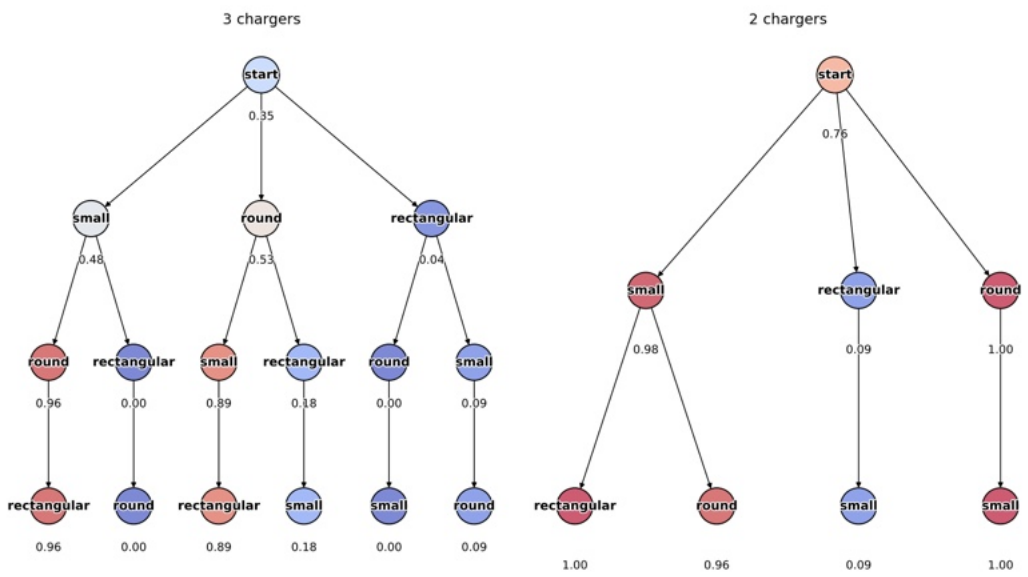


Figure 8: **Per-path success rates from human demonstrations in the plug-insertion environment.**

F Dataset details

All teleoperated environments use a leader–follower setup with four operators. For every run we log time-stamped robot state (arm pose, gripper command), RGB images, and camera calibration. Subgoal labels are derived from gripper-state transitions (e.g. `open`→`close` for grasps). In the figures below, each root-to-leaf path is a complete subgoal sequence and node values report $\widehat{p}_{\text{succ}} = \sharp\text{successes}/\sharp\text{trials}$.

Plug insertion. We collect 450 demonstrations covering all admissible insertion orders (six paths for 3-socket, four for 2-socket). Subgoals are **Grasp-x** and **Insert-x in y**. Fig. 8 shows per-path success rates. The distribution is skewed: for 3-charger, `small`→`round`→`rectangular` is highly reliable (0.96) while paths placing `rectangular` early tend to fail (0.00–0.18). For 2-charger several paths are near-perfect (1.00) whereas `rectangular`→`small` is weak (0.09).

Drawer packing. We collect 240 demonstrations (120 per distractor type: `boxed food` / `cans`) covering the PICK, PUSH, and LEAVE strategies. Subgoals include **Pick-x**, **Push-x**, **Grasp-y**, **Put-y on drawer**, and **Close drawer**. In the `boxed food` setting PUSH is consistently reliable (1.00) while LEAVE is weak (0.30–0.35). In the `can` setting PICK dominates (0.91–1.00) while LEAVE almost always fails (0.00–0.05).

SIMPLER (WidowX / Bridge v2). Data is collected through autonomous rollouts from a Bridge v2 post-trained π_0 policy, with no teleoperation. Subgoals are **grasp-x** and **put x on top of y**, annotated with the environment flag `info["is_src_obj_grasped"]`. Table 4 summarizes 600 trajectories.

SIMPLER (Google Robot / Fractal). We extend the evaluation using a Fractal post-trained π_0 policy. For pick and move tasks we use the same grasp edge type, and for drawer tasks we define a drawer-state edge from the ground-truth joint position (`qpos`). Table 5 summarizes 2,636 trajectories.

Real-world cabinet packing. For `sponge` trials a `clock` distractor is present and operators use *push*, *pick*, or *leave*. For `towel` trials operators either *fold* before insertion or insert directly. **Push-clock** dominates (0.95), **Pick-clock** is moderate (0.47), and **Leave** almost always fails (0.05). For towels **Fold** is highly reliable (0.90) versus **Leave** (0.15).

Real-world long-horizon packing. Packing four objects (toy, stamp, lemon, tape) into two containers and closing both lids ($\sim$10 subgoal steps), with 240 demonstrations across 12 orderings (20

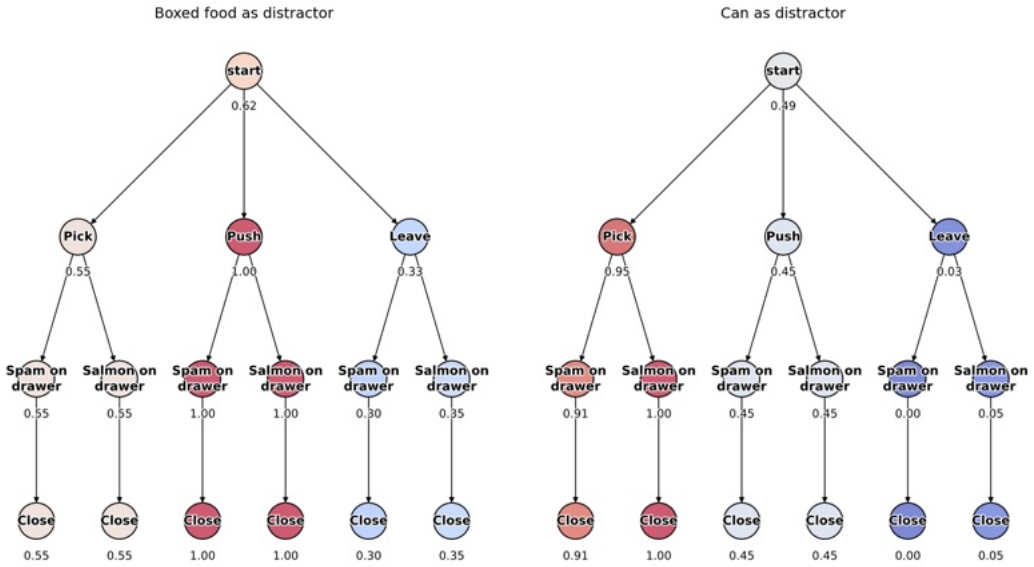


Figure 9: **Per-path success rates from human demonstrations in the drawer-packing environment.**

Table 4: **WidowX outcomes per object.** Rates are ♯/total (%).

Object	Total	Grasped	Success
eggplant	72	66 (91.7%)	60 (83.3%)
carrot	192	112 (58.3%)	48 (25.0%)
spoon	144	77 (53.5%)	56 (38.9%)
cube	192	128 (66.7%)	32 (16.7%)
Overall	**600**	**383 (63.8%)**	**196 (32.7%)**

Table 5: **Google Robot (Fractal) outcomes per task group.**

Task group	Total	Success	Rate
pick coke can	850	478	56%
open/close drawer	546	292	53%
move near	351	162	46%
place apple in drawer	889	293	33%
Overall	**2,636**	**1,225**	**46.5%**

each). Table 6 reports per-ordering success. Every ordering placing the lemon before the tape yields 0% success, reflecting a physical obstruction.

Table 6: **Per-ordering success rates in the long-horizon packing task (20 demos each, 240 total).** L→T / T→L / L+T denote lemon-first, tape-first, and simultaneous.

Ordering	L→T	T→L	L+T
Toy→Stamp L	0%	55%	45%
Toy→Stamp R	0%	40%	20%
Stamp L→Toy	0%	65%	65%
Stamp R→Toy	0%	50%	45%
Overall	0%	53%	44%

Real-world setup. Fig. 11 shows the real-world workspace. Demonstrations are teleoperated with a leader arm driving the follower arm, and the policy observes a wrist-mounted camera and a fixed third-person camera.

G Fidelity of System 2's generated scene graphs

At test time the scene graphs are generated by VINE's own trained System 2 (§3). The detector and VLM of §E are used only to annotate the training set, and System 2 learns to produce these graphs from $\mathcal{D}_{\text{sys2}}$. What follows therefore measures the *model's* own abstraction and world model.

Protocol. For each trial we extract the executed path by following the highest-value child from the root, align it step-by-step with the ground-truth subgoal annotations, and compute five met-

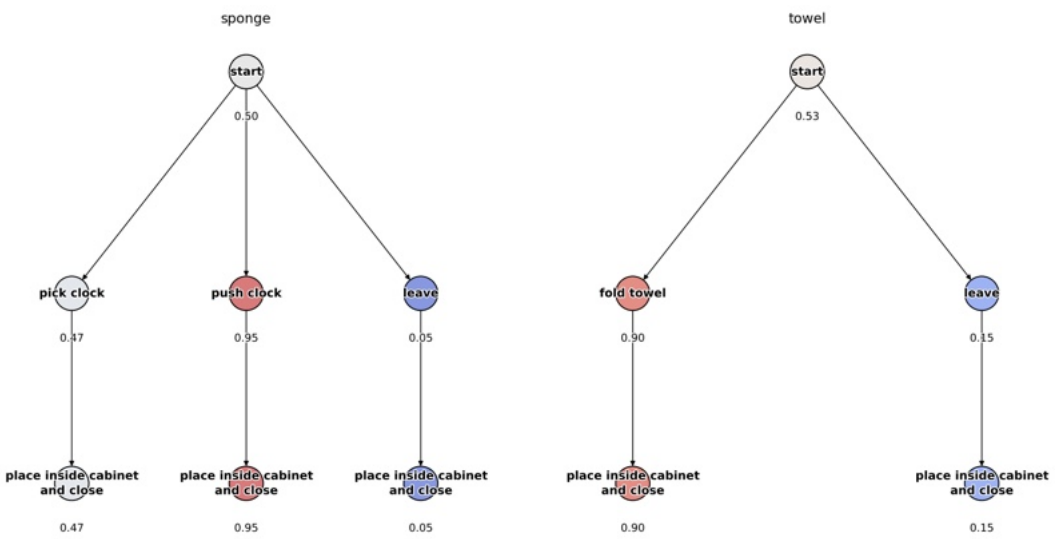


Figure 10: **Per-path success rates from human demonstrations in the real-world environment.**

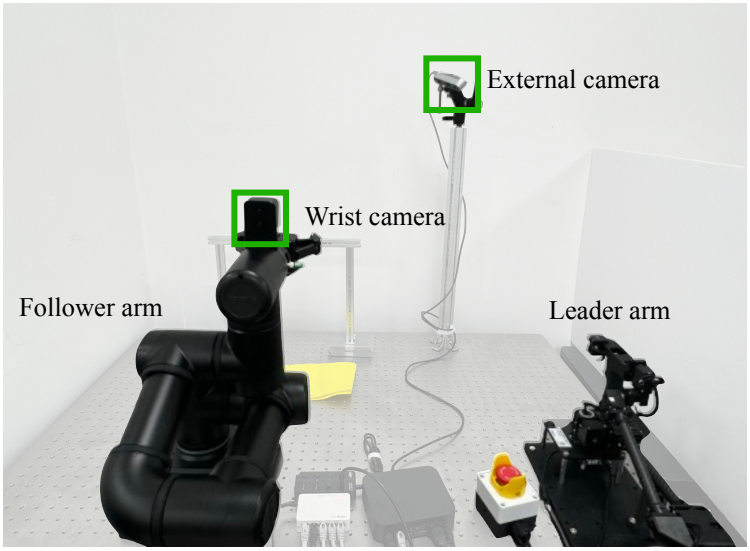


Figure 11: **Real-world workspace.** A 6-DoF arm under joint-position control at 20 Hz with a wrist-mounted and a fixed third-person camera.

rics, which we then correlate with task success across all $n = 210$ trials (Table 7). Here `action_set_iou` is the Jaccard overlap between predicted and ground-truth subgoal sets (order-agnostic coverage), and proximity metrics are $1 -$ dist between 2D positions in the graph. `relation_acc` is averaged only over objects common to both graphs, and all coordinates are normalized to $[0, 1]^2$.

Interpretation. The generated graphs are accurate about *relations* (0.85) and localize objects reasonably (bbox center proximity 0.79). Trials with fewer shared objects can inflate per-object relation accuracy, so the former should be read as an upper estimate. The planner is designed to use identities and relations, which are stable under small coordinate shifts, and System 1 is trained with bounding boxes perturbed within a small range so that residual localization error does not propagate into control. This is consistent with the correlation structure: node-level metrics show weak associations with success ($|r| \leq 0.19$), whereas subgoal-level metrics are stronger ($r = 0.29$), since a misselected subgoal redirects execution down a branch that downstream accuracy cannot correct.

Table 7: **Point-biserial correlation of scene-graph and execution metrics with task success** ($n = 210$; $^*p < 0.05$). All quantities are computed on scene graphs generated by System 2. Proximity metrics are $1 - \text{dist}$.

	Metric	r	p	Fail	Succ	Total
Edge	action_set_iou	$+0.29^*$	0.001	0.45	0.70	0.62
	subgoal_acc	$+0.22^*$	0.002	0.40	0.62	0.55
Node	relation_acc	-0.17^*	0.014	0.93	0.81	0.85
	bbox center prox.	$+0.075$	0.28	0.78	0.79	0.79
	gripper→obj prox.	-0.19^*	0.005	0.74	0.70	0.72

What this evidence does not establish. Table 7 is observational. It correlates naturally occurring scene-graph quality with success rather than degrading the graph and measuring the induced drop, and under limited variance a weak correlation can arise from restricted range rather than genuine insensitivity. A controlled study (injecting bounded noise or drop-outs into the generated graph at inference and plotting success against corruption level) would provide a more direct test, which we leave to future work. We therefore present these numbers as a diagnostic, not as a robustness guarantee.

World-model diagnostic. Restricting attention to the *imagined* successors, System 2's predicted relations match observed post-execution states with accuracy 0.83, spatial proximity stays high (0.79 bbox center, 0.71 gripper→object), and action-set coverage is 0.65. Predicted subgoal-sequence accuracy is more moderate (0.58), and we do not have an oracle-successor ablation isolating how successor-prediction error changes branch selection. The diagnostic is therefore evidence that the abstract world model is grounded enough for plan ranking, not that it is reliably accurate.

H Additional qualitative results

Fig. 12 shows real-world planning trees and executions on the cabinet-packing tasks. Fig. 13 compares full trajectories on an unseen drawer-packing configuration against π_0, prompted Gemini-2.5-Flash in place of our trained System 2, and VINE-Chain. In this example, VINE-Full relocates the obstructing bowl to the table before placing the target object, whereas the other methods leave it in place or push it aside. Fig. 14 shows a generated tree and its execution on SIMPLER, where each edge carries a 2D end-effector target. Fig. 15 shows VINE-Prune completing the $\sim$10-step long-horizon packing task.

I Ablations

Search strategy: MCTS vs. DFS vs. greedy. Fig. 16 compares MCTS with a fixed-order DFS on plug insertion. MCTS is better on both seen (80.0% vs. 68.9%) and unseen (42.2% vs. 33.3%) configurations and nearly 10 s faster (28.37 s vs. 38.09 s), because DFS must fully expand unpromising branches before backtracking whereas MCTS uses the value to prioritize. Table 8 and Fig. 17 compare against a greedy baseline that uses the same trained System 2 but executes its top-ranked first step without search. The two tie on seen configurations (0.800) but diverge under shift (0.422 vs. 0.311, +35.7% relative) at negligible latency cost, since both methods require similar numbers of scene-graph expansions. Fig. 17 shows an example. Greedy takes the highest immediate value (0.758, small charger first) and continues through 1.052 and 1.102, whereas MCTS accepts a lower first step (0.619, round charger first) on a path with higher subsequent values (1.185 and 1.140). In execution, the greedy ordering creates spatial conflicts, while the MCTS ordering maintains clearance.

Table 8: **MCTS vs. greedy plan execution (plug insertion).**

Method	Seen	Unseen	Time (s)
Greedy (System 2, no search)	**0.800**	0.311	27.62
MCTS (System 2, full search)	**0.800**	**0.422**	28.37

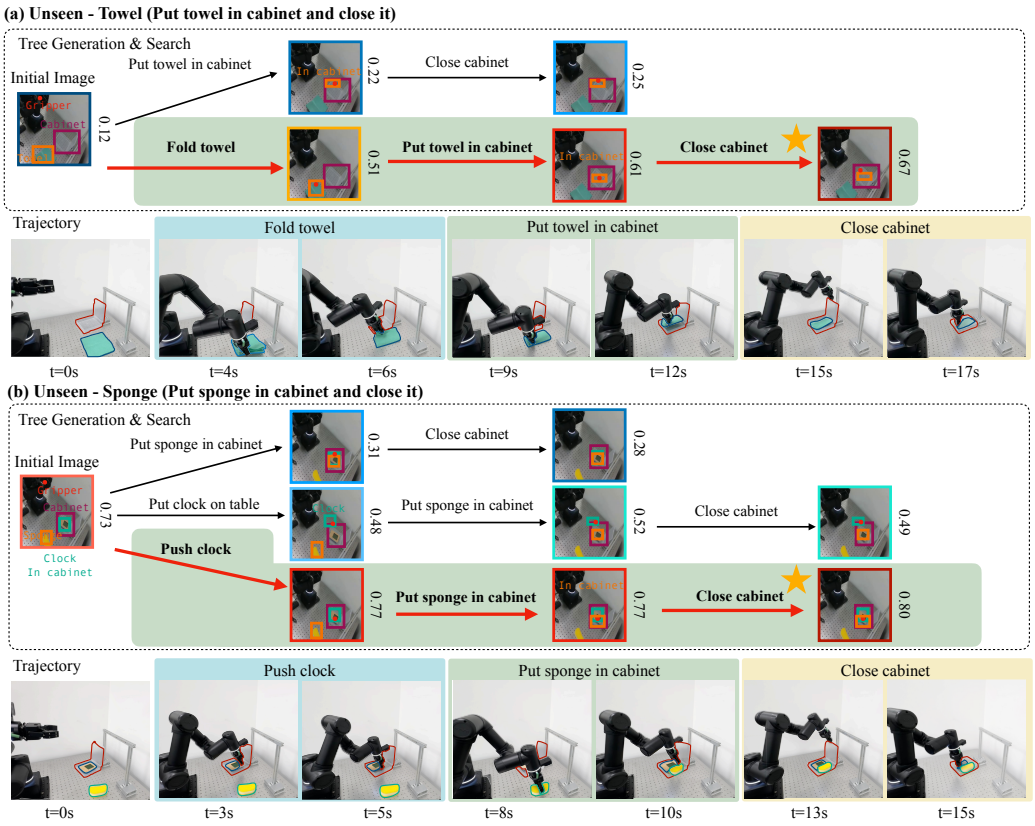


Figure 12: **Real-world planning and execution on the cabinet-packing tasks. (a)** *Towel–unseen*: the search selects *fold* → *put in cabinet* → *close*, preferring the three-step plan over direct placement. **(b)** *Sponge–unseen* with a clock distractor, reproduced from Fig. 4(b) for completeness.

Planning latency and its scaling with w. Fig. 18 plots one-time planning time against the per-step expansion width w on unseen plug insertion: 23.9 → 26.2 → 28.4 → 30.8 → 32.5 s for $w = 1..5$, near-linear, while the corresponding success curve in §5.2 (Fig. 3) peaks at w=4. Table 9 reports the breakdown at $w = 3$ (A100, 31 trials across 6 tasks). Scene-graph generation dominates (21.66 s, 79.1%), of which 17.91 s is batched decoding of successor states. Edge generation adds 4.47 s (16.3%), the MCTS overhead itself is 1.27 s (4.6%), and System 1 action inference takes 0.14 s per chunk, separate from this one-time planning cost. All components are forward passes of our model without external API calls. Latency therefore grows with the number of node decodes rather than with the search algorithm, and faster decoding or caching of shared node representations would reduce the upfront cost. Planning is performed once at the initial state, and System 1 then runs at 20 Hz.

Table 9: **Planning latency breakdown at $w = 3$ (mean±std, seconds; A100, max batch size 5).** Values are averaged over 31 trials across 6 tasks, so they differ slightly from the plug-insertion times in Fig. 18 and Table 8. Every row is VINE's own inference, and no external perception service is queried at test time. Action inference is measured per action chunk and is not part of the one-time planning cost.

Component	Time (s)	% of planning
Scene-graph generation (System 2)	21.66 ± 4.04	79.1%
Edge generation	4.47 ± 1.69	16.3%
Tree-search overhead	1.27 ± 0.26	4.6%
Planning total	$\mathbf{27.40 \pm 5.97}$	100%
Action inference (System 1, per chunk)	0.14 ± 0.04	–

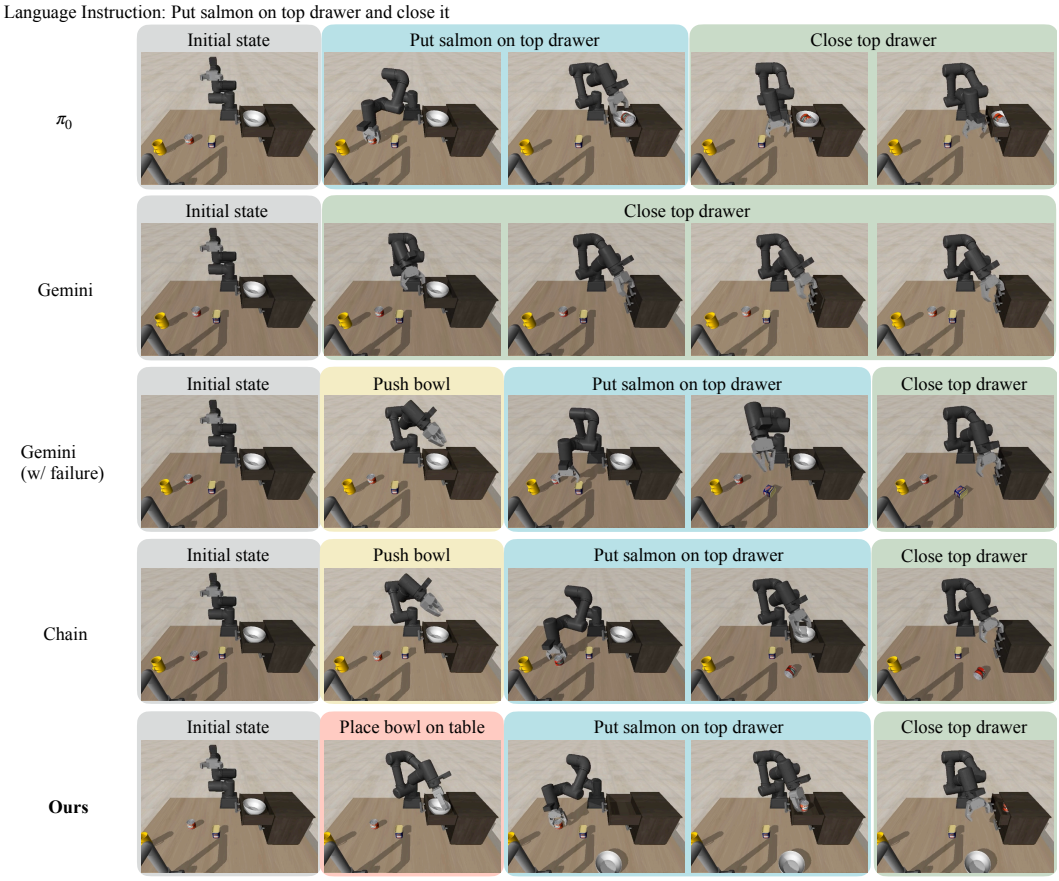


Figure 13: **Trajectory comparison on an unseen drawer-packing configuration.** We compare π_0 (Black et al., 2024), prompted Gemini-2.5-Flash (Comanici et al., 2025) in place of our trained System 2 (with and without failure examples), VINE-Chain, and VINE-Full. π_0 and Gemini without failure examples do not clear the obstructing bowl, Gemini with failure examples and VINE-Chain push it aside, and VINE-Full relocates the bowl to the table before placing the salmon.

Depth-adaptive pruning. On the long-horizon packing task we compare two search configurations of the same trained model at comparable planning time ($\sim$74 s): VINE-Fixed (w=2, fixed batch B=5 at every depth) and VINE-Prune (w=2, batch linearly narrowed from B=8 at depths 1–5 to B=2 at depths 6–10), which invests more computation where value estimates are better calibrated and less where bootstrap error accumulates. Table 10 reports both. The two reach identical aggregate unseen success (36.7%), and VINE-Prune is slightly behind on progress, so adaptive allocation does not improve aggregate performance in this setting. It is, however, more scene-adaptive: VINE-Fixed commits to the same toy$\rightarrow$stamp left ordering (55% training success rate) across scenes, whereas VINE-Prune varies its ordering per scene and selects the superior stamp left$\rightarrow$toy path (65%, Fig. 15). Confidence-based pruning that uses the value head's own uncertainty to set per-node expansion width is a natural next step.

Table 10: **Fixed vs. depth-adaptive search on the long-horizon packing task.** Progress is normalized to $[0, 1]$, and unseen columns average three unseen configurations.

Method	Progress		Success rate	
	Seen	Unseen	Seen	Unseen
π_0 (Black et al., 2024)	0.600	0.422	0.150	0.067
VINE-Fixed (B=5)	**0.683**	**0.689**	**0.300**	**0.367**
VINE-Prune (B=8$\rightarrow$2)	0.592	0.656	0.250	**0.367**

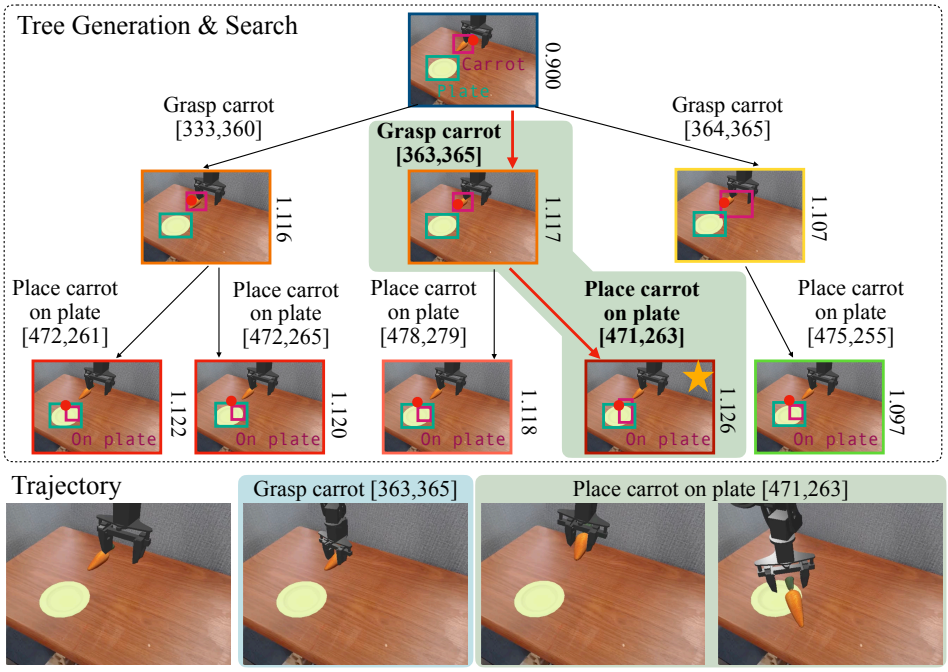


Figure 14: **Generated tree and executed trajectory in the SIMPLER environment.** Node values are raw value-head outputs (unclipped), so they can exceed 1.

Weight-merging coefficient λ_m and expectile τ_e. Table 11 reports both sensitivities. Merging performance peaks at $\lambda_m = 0.6$ (0.800 seen, 0.422 unseen) and degrades when biased toward either π_0 or PaliGemma. We therefore regard weight interpolation as a practical choice that requires tuning rather than a principled fusion mechanism. Alternatives such as concatenating embeddings and training a fusion head are not evaluated here. For the expectile, performance is stable within $\tau_e \in [0.6, 0.8]$ with $\tau_e = 0.7$ best. At $\tau_e = 0.5$ the loss becomes symmetric, can no longer prioritize the failure signal, and unseen success drops to 0.575.

Table 11: **Sensitivity analysis.** Sensitivity to (left) the weight-merging coefficient λ_m on plug insertion and (right) the expectile parameter τ_e on drawer packing. Avg. is the success rate over all seen and unseen trials (plug: 45/45, drawer: 80/40).

	λ_m (plug)			τ_e (drawer)		
Value	Seen	Unseen	Avg.	Seen	Unseen	Avg.
0.5	0.756	0.356	0.556	0.750	0.575	0.692
0.6	**0.800**	**0.422**	**0.611**	0.775	0.625	0.725
0.7	0.733	0.289	0.511	**0.800**	**0.675**	**0.758**
0.8	0.422	0.244	0.333	0.738	0.675	0.717

J FAILURE DECOMPOSITION

We manually annotate all $n = 64$ failure cases across 210 trials into three categories: *action failure* (System 1 fails during execution, 51.6%), *planning failure* (System 2 selects an infeasible sequence, 34.4%), and *planning–action misalignment* (System 1 deviates from a correct plan, 14.1%), and Fig. 19 shows the distribution. Action failures dominate in both environments, though drawer packing shows a higher share of planning failures because contact outcomes are more stochastic (pushing a distractor may or may not clear the space depending on friction). Conditional on a feasible plan (188/210 trials), System 1 succeeds in 146 cases (77.7%), suggesting that the failure-aware value identifies viable subgoal sequences in most trials and that further gains may come from improving low-level execution. This analysis relies on a manual taxonomy over 64 cases, a small base

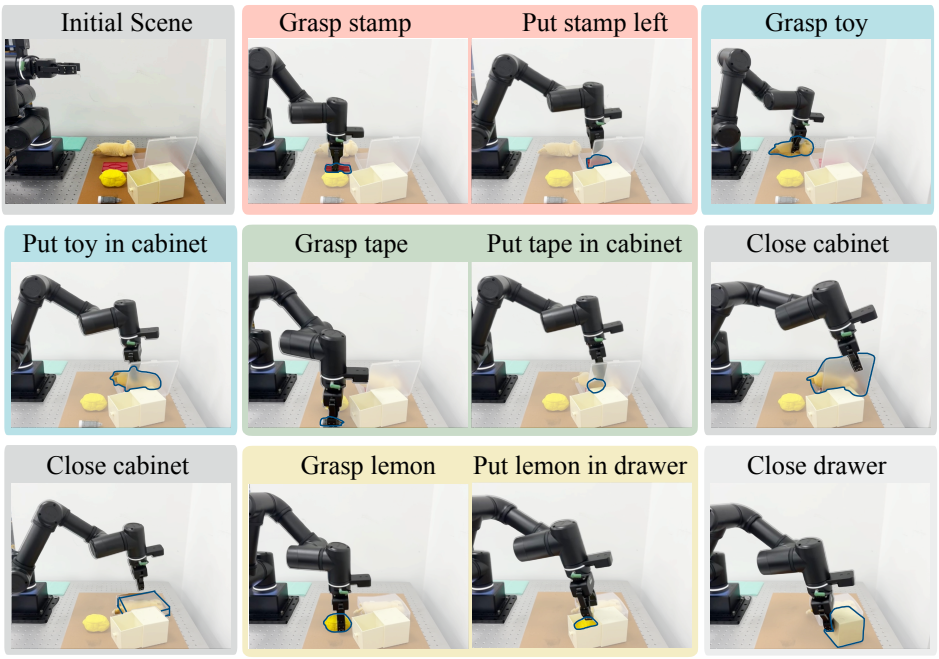


Figure 15: **VINE-Prune execution on the long-horizon packing task.** Stamp (left) → toy → tape → close cabinet → lemon → close drawer. The adaptive search selects the highest-success ordering (65% training success rate), which packs the stamp before the toy.

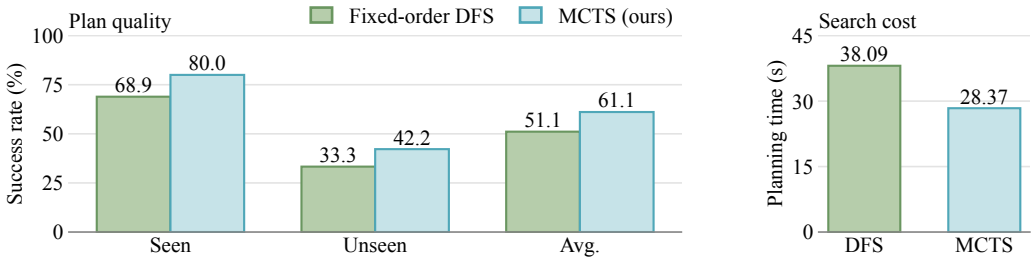


Figure 16: **Tree-search comparison on plug insertion: MCTS vs. fixed-order DFS.**

with some annotator judgement involved, so the percentages should be read as indicative rather than precise.

K LIMITATIONS

Open-loop planning. The tree is built once, at subgoal granularity, with no within-chunk feedback. We deliberately avoid a stop-and-replan loop: replanning at every chunk enforces a quasi-static assumption and interrupts continuous motion, which is costly in contact-rich manipulation. As a result, pose error, sensor latency, and contact disturbance accumulate between subgoals, so the predicted successor state can drift away from what the robot can stably execute, and the planned path and the achievable path diverge. The uncertainty-triggered replanning module of §L closes part of this gap (67.5% → 75.0% on unseen drawer packing) without retraining, but its variance threshold is calibrated manually per task and the extra computation rules out reactive settings.

Subgoal-level failure labelling. Our labels record *whether* a transition succeeded, not *why*. This makes them objective and inexpensive to collect (§E), but it also means action-level failure modes (grasp slippage, contact error, mid-motion collision) are invisible to the planner. The consequence compounds with a second choice: System 1 is trained only on successful chunks, so it has no failure-

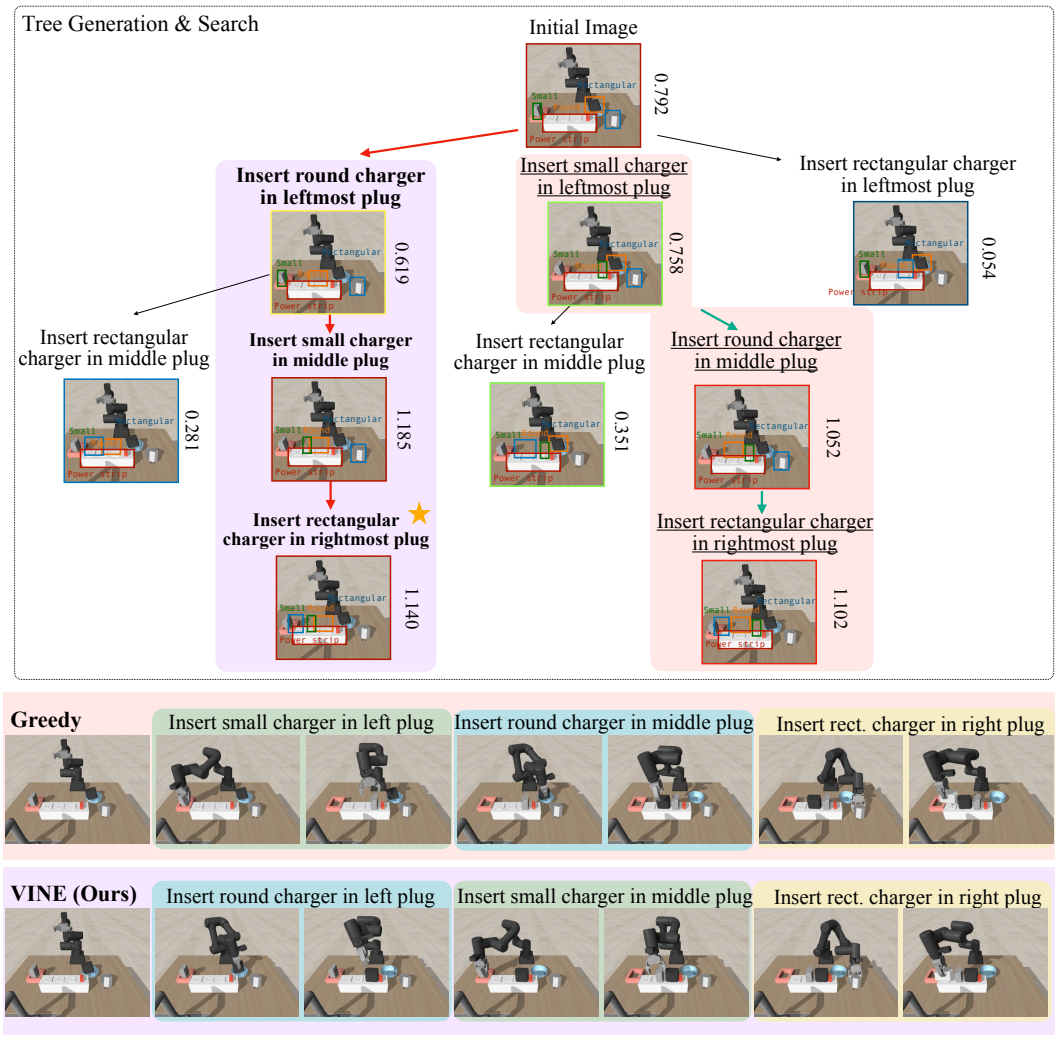


Figure 17: **MCTS vs. greedy execution on an unseen plug-insertion episode.** Grasp subgoals are omitted for clarity. **Top:** search tree with reach–avoid values. **Bottom:** execution rollouts, in which greedy creates spatial conflicts while VINE maintains clearance. Node values are raw value-head outputs (unclipped), so they can exceed 1.

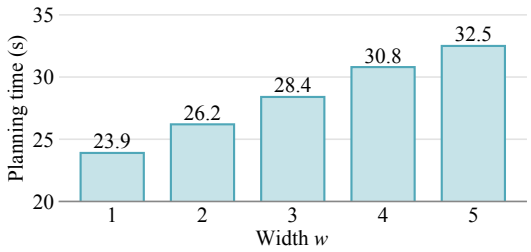


Figure 18: **One-time planning latency versus per-step expansion width w on unseen plug insertion (A100).** Growth is near-linear in w because each additional candidate costs one more successor-node decode. The success-rate counterpart is Fig. 3.

conditioned gradient with which to self-correct once perturbed off the nominal distribution. The framework therefore improves planning-level robustness and does not directly address execution-side failures, which our taxonomy puts at 51.6% of all failures. This is the largest failure category and a complementary direction for future work. We note that the taxonomy rests on a manual annotation of 64 cases, a small base with some annotator judgement involved, so those percentages are indicative rather than precise.

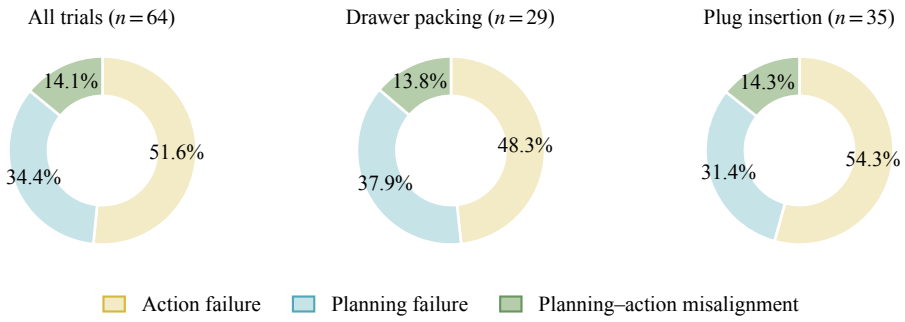


Figure 19: **Failure type distribution.** All trials (left), drawer packing (center), and plug insertion (right).

Relational, not metric, abstraction. The 2D scene graphs the model generates omit metric geometry, contact cues, and short-horizon dynamics that matter in contact-rich manipulation.

Measured against ground truth they are relationally accurate (0.85, §G), but relational accuracy does not reflect geometry that the serialization does not encode, and this omission is by design. This can miscalibrate feasibility when objects are partially occluded or slightly mislocalized, or when frictional effects dominate, causing the search to misrank viable branches or prune plans that the true dynamics would support. Our evidence that the design tolerates this is observational (a correlation between naturally occurring graph quality and success over $n = 210$ trials) rather than a controlled study that injects bounded corruption and measures the induced drop. The real-world clock example (§5.4) also marks the boundary of the mitigation: the value function can account for a physical detail the text serialization omits only when that detail produced failures in the training data, so a novel object whose fragility was never demonstrated remains a failure mode.

Depth and latency. Value calibration degrades with search depth. At depth 6 (plug insertion, w=3) the planner selects the highest-success path and reaches 80% seen success. At depth 10 (long-horizon packing, w=2) accumulated bootstrap error can reverse the ranking between closely matched alternatives (it selects a viable ordering with 55% training success instead of the 65% one) while still placing catastrophic orderings (0% training success) at the bottom. Errors therefore appear when discriminating among viable paths, not when separating viable from infeasible ones, and reliable *ranking* extends to roughly 6 subgoal steps in our setup. On cost, planning takes ∼27 s upfront at w=3 (Table 9) and ∼74 s at depth 10, paid once before the first action. The node count is bounded by $O(w^D)$, so scaling to 15–20 steps would require a smaller w, more aggressive pruning, or parallel node decoding. Since 79% of the latency is System 2 decoding scene graphs (§I), faster decoding or node caching is a more direct remedy than a different search algorithm.

L Uncertainty-triggered replanning

To mitigate the drift caused by executing long action chunks without feedback, we attach a lightweight uncertainty monitor (Agia et al., 2024) that enables test-time replanning without retraining. When the within-chunk variance becomes high, the monitor down-weights the score of the currently committed branch and switches to the next-best alternative already present in the search tree, turning quasi-static open-loop execution into an instance-adaptive procedure that can abandon deteriorating motions. On unseen drawer packing this raises success from 67.5% to 75.0%, above the π_0 baseline's 57.5%, with all learned parameters unchanged (Fig. 21). Fig. 20 shows the mechanism: the planner first selects the green path (push the obstructing box aside), detects high uncertainty during the first step, and switches to the second-best plan (place the box on the table). The calibration is currently manual and task-specific and the added computation limits real-time use, so we present this as an orthogonal, practical mitigation rather than part of the main method.

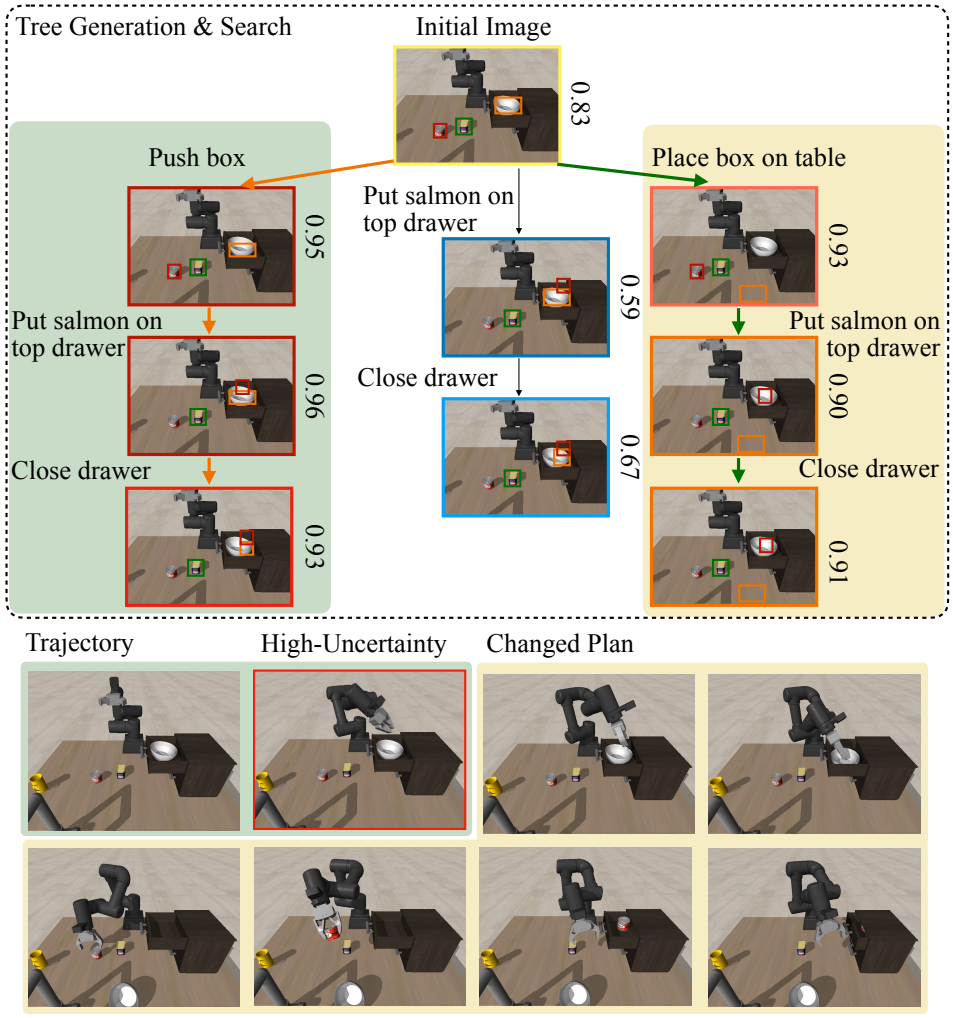


Figure 20: **Qualitative replanning.** The planner first selects the green plan (*push box* → *place item* → *close*), detects high uncertainty during the first step, and switches to the yellow alternative (*place box on table* → *place item* → *close*), completing the task. Shaded panels mark the uncertainty trigger and the plan switch, and bottom rows show the executed trajectories.

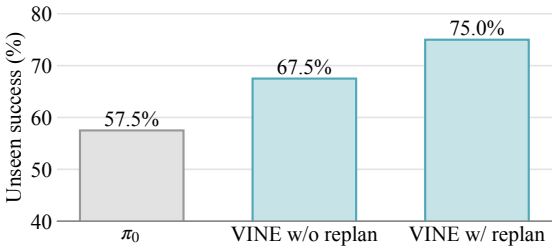


Figure 21: **Success rates on unseen drawer packing with and without uncertainty-triggered replanning.** Replanning raises success from 67.5% to 75.0% with all learned parameters unchanged, and π_0 is shown for reference.

M HYPERPARAMETERS

System 1 and System 2 are trained independently with separate LoRA adapters on the merged backbone.

Table 12: **Hyperparameters for VINE.**

Group	Parameter	Value
System 1 (policy)		
	Optimizer	AdamW (β=$(0.9, 0.95)$)
	Learning rate	2.5×10^{-5}, cosine decay
	Batch / steps	12 / 20–30 K
	λ_{act} / λ_{done} / κ_{done}	1.0 / 0$\rightarrow$0.05 (10 K warmup) / 0.3
System 2 (LM / value)		
	Learning rate	2–3$\times10^{-5}$, cosine decay
	Batch / steps	2–3 / 1.5–3 K
Value head		
	γ / τ_e / EMA rate	0.99 / 0.7 / 0.9
	Value weight / warmup	1.2 / 200 steps
Focal loss (done expert)		
	γ_d / α_d / weight	2.0 / 0.5 / 0.1
LoRA		
	Rank / α / dropout	256 / 128 / 0.05
	Target modules	q/k/v/o/gate/down/up_proj
Tree search (MCTS)		
	M / B / w / α	4 / 5 / 3 / 0.8
	Max tokens (node / edge)	200 / 50
	Decoding	beam search (edge proposals), greedy (successor node)