Deciding When to Trust the Expert: VLM-Augmented Policy switching for Safe driving (VLAPS)

Qizhao Chen, Debajyoti Chakrabarti
Department of Mechanical and Aerospace Engineering
University of California, Los Angeles, United States
qizhaoc@ucla.edu, debjyoti@ucla.edu

Keywords: Robots, Navigation, Learning, VLM, Policy Switching

1 Introduction

Mobile robots are increasingly deployed in human environments, where they must move safely around people and obstacles. In driving-style tasks, the goal is to navigate to waypoints using onboard sensing while avoiding collisions and lane violations.

Imitation learning (IL) from expert demonstrations is a common way to train such policies [1, 2], but collecting demonstrations is expensive and small test-time errors can push the agent into unseen states. Reinforcement learning (RL) for driving and navigation [3] removes the need for expert actions, yet typically depends on carefully engineered rewards. Human-in-the-loop methods mitigate these issues by letting a person intervene and using these interventions as learning signals [4], but still require a human supervisor to continuously monitor the system.

Vision–language models (VLMs) can interpret complex scenes and answer safety- or goal-related questions from images, and have been used for preference-based evaluation and reward modeling [5]. We propose VLAPS (VLM-Augmented Policy Switching), which couples an RRT* + PD expert with an online-learned policy and a VLM-based gate. At each step both expert and learner propose actions; we roll out their trajectories, overlay them on the ego-view image, and ask a VLM which one is safer. The binary label is used both to switch execution and to supervise TD-style value learning and behavior cloning.

2 Related Work

Human-in-the-loop methods such as HG-DAgger [6] improve robustness by letting a human monitor the robot and take over when it is about to fail, using these interventions as additional training data. PVP4Real [4] goes further and shows that combining human interventions and demonstrations with temporal-difference learning and behavior cloning enables fast, reward-free policy learning on real robots, but still relies on continuous human supervision. Our work follows the same proxy-value-plus-BC recipe as PVP4Real, but replaces the human with a vision–language model that compares expert and learner trajectories and acts as an automatic supervisor for both control and learning.

3 Problem Formulation

We consider a simulated autonomous driving task where an ego vehicle must follow a route while avoiding collisions and staying within lane boundaries. At each time step t, the simulator provides an observation s_t (vehicle state, LiDAR-based occupancy, lane distances), and the controller outputs a continuous action a_t (steering and longitudinal acceleration). The goal is to learn a policy $\pi_\phi(a_t \mid s_t)$ that reaches waypoints, avoids obstacles, and remains in the drivable region. We treat this as

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

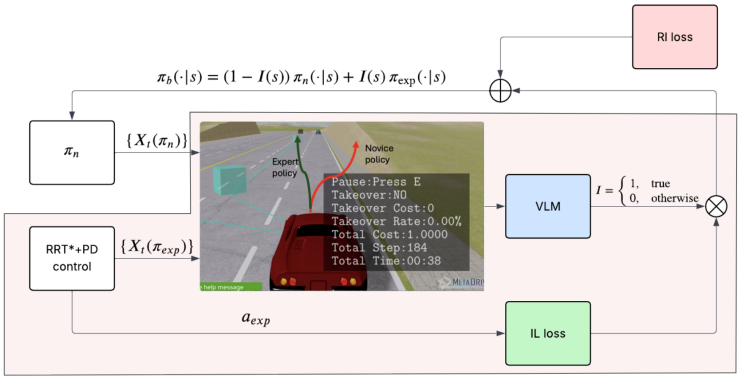

Figure 1: Overview of the VLAPS pipeline

a reward-free online learning problem: instead of a scalar reward, we assume access to a hand-engineered expert policy π_{exp} and a VLM that, given visualizations of candidate trajectories, returns a binary preference over which one better satisfies high-level safety and progress criteria. The learner must use these expert actions and VLM preferences to improve π_ϕ over time while keeping executed actions safe during training.

4 Method

VLAPS combines three components: an RRT* + PD expert controller, a novice policy, and a VLM gate that compares short-horizon rollouts from both and chooses which policy to execute. The same gating signal is used to define the behavior policy and to provide supervision for a proxy value function and a behavior cloning loss.

4.1 Expert Policy

To make the entire learning fully autonomous, our new expert controller is a hand-engineered policy that combines an RRT*-based local planner operating on a LiDAR-derived occupancy grid with a simple proportional–derivative (PD) controller for tracking.

4.1.1 Occupancy Grid and RRT* Local Planner

We plan in an ego-centric top down (x, y) frame where the vehicle is at the origin, x points forward, and y is lateral. From the forward-facing LiDAR, we build a 2D occupancy grid over $x \in [0, x_{\max}]$ and $y \in [-y_{\max}, y_{\max}]$ with resolution $\Delta x = \Delta y = 0.05\,\text{m}$; in our experiments we use $x_{\max} = 30\,\text{m}$ and $y_{\max} = 10\,\text{m}$. Each LiDAR beam i provides a normalized range $d_i \in [0, 1]$, mapped to a metric range $r_i = d_i R_{\max}$ with LiDAR distance limit $R_{\max}$, and a local point.

Occupied grid cells are then inflated by a safety radius of 0.5 meters, considering the vehicle size. Lane boundaries, which are not directly observable from LiDAR. This information is provided by the simulator as distances d_{left} and d_{right} from the vehicle center to the left and right lane edges. We shrink this lateral corridor by a margin m_{lane} and mark cells outside as occupied

$$y \in [y_{\text{right}},\, y_{\text{left}}], \quad y_{\text{left}} = d_{\text{left}} - m_{\text{lane}}, \quad y_{\text{right}} = -d_{\text{right}} + m_{\text{lane}}$$

On this grid we run an RRT* planner in the local (x, y) workspace. The start state is fixed at the ego position, and the goal is obtained from the next navigation checkpoint in global coordinates transformed into the local frame. We run a fixed budget of 200 iterations. At each iteration, we sample within y range, and $[x_{\min}, x_{\max}]$ spans from $x = 0$ to x_{goal}. In free space, we find the nearest node q_{near} and extend along the line toward q_{rand} by a fixed step (2 m in our implementation). Among all nodes within radius $r^* = 3\,\text{m}$, we choose as parent the one that minimizes the cost-to-come plus edge length and perform standard RRT* rewiring. Any node that enters a goal ball of

radius $r_{\text{goal}} = 1\,\text{m}$ is treated as a candidate solution. For tracking, we extract a single lookahead waypoint from the resulting path,

$$q_{\text{LA}} = \arg\min_{q \in \mathcal{P}} \{ \|q\| \mid \|q\| \geq d_{\text{LA}} \},$$

with $d_{\text{LA}} = 2\,\text{m}$. If no point satisfies this, we use the final waypoint q_T.

4.1.2 Proportional-Derivative (PD) Tracking Controller

The expert converts the lookahead target $q_{\text{LA}} = (x_{\text{LA}}, y_{\text{LA}})$ into steering and acceleration commands via a PD controller on heading and speed. We consider two controllers: one for steering and one for speed tracking. For the lateral (steering) PD controller, the desired heading is the angle from the ego to the lookahead waypoint in the local frame

$$e_{\psi,k} = \text{atan2}(y_{\text{LA},k},\, x_{\text{LA},k}).$$

The steering command is computed as a PD term

$$u^{\text{raw}}_{\text{steer},k} = K^{\psi}_{p} e_{\psi,k} + K^{\psi}_{d} \frac{e_{\psi,k} - e_{\psi,k-1}}{\Delta t},$$

with gains $K^{\psi}_{p} = 2.0$ and $K^{\psi}_{d} = 0.001$, and controller time step $\Delta t = 0.1\,\text{s}$. We do not use an integral term in the steering loop. This is because at everytime step, the local goal from RRT* will update the tracking position for the PD controller, and the controller and planner are running at the same frequency.

For the longitudinal (speed) PD tracking we define the speed error with $v_{\text{ref}} = 5\,\text{m/s}$.

$$e_{v,k} = v_{\text{ref}} - v_k,$$

and compute the pedal command as

$$u_{\text{pedal},k} = K^{v}_{p} e_{v,k} + K^{v}_{d} \frac{e_{v,k} - e_{v,k-1}}{\Delta t},$$

with gains $K^{v}_{p} = 0.1$ and $K^{v}_{d} = 0.05$. We do not use an integral term for the same reason, and we clip $u_{\text{pedal},k}$ to the simulator's normalized range $[-1, 1]$.

4.2 Policy Visualization and VLM-gated Policy Switching

To make the effect of each control decision visually interpretable, we render not only the raw driving scene but also an approximate rollout of the vehicle under the current actions using a kinematic bicycle model. At time step k, the learned policy and the expert policy each output normalized steering and pedal commands. We map them to position command in the next time steps using the discrete-time kinematic bicycle model (see Appendix 8.4). For visualization only, we use an enlarged integration step $\Delta t_{\text{vis}} = 1.5\,\text{s}$ for the roll out.

The predicted positions for both policies are passed back to the simulator and rendered as polylines in the ego-centric view. As illustrated in Fig. 2. This image is fed into a vision–language model (VLM), which acts as an external observer that periodically decides which policy should control the vehicle. Every $T_{\text{vlm}} = 10\,\text{s}$ of simulator time we pause the simulation, render a frame with both predicted rollouts visualized as in Fig. 2, and query the VLM.

For the VLM model, we use Chat-GPT VLM API with model 4o-mini. It calls the vision model with a brief prompt, and parses a strict JSON field `"response"` from the output:

$$\texttt{"response"} \in \{0, 1\}.$$

The full text of the user prompts used for this VLM query is provided in Appendix 8.5. The prompt asks the VLM to compare the red (expert policy) and blue (agent policy) trajectories purely based on how safely and efficiently they approach the goal region. A value of `{"response": 1}` is interpreted as the red policy is better than the blue policy.

After the response of VLM, let $z_k \in \{0, 1\}$ denote the VLM decision at query k. Between two consecutive VLM calls, the executed action at time step t is

$$a_t = (1 - z_k)\, a^{\text{rl}}_t + z_k\, a^{\text{exp}}_t, \qquad t \in [kT_{\text{vlm}}, (k+1)T_{\text{vlm}}),$$

where a^{rl}_t and a^{exp}_t are the actions proposed by the learned and expert policies.

Figure 2: Policy visualization in image space. The predicted rollout of the learned policy is shown as a blue curve, while the expert policy rollout is shown as a red curve, overlaid on the ego-view.

4.3 TD Policy Learning and Behavior Cloning

We follow the reward-free human-in-the-loop framework of PVP4Real [4], but replace human interventions with a VLM-based binary preference signal. At each time step t the simulator provides an observation s_t. The novice policy π_ϕ outputs an action a_t^{rl}, while the expert controller π_{exp} outputs a_t^{exp}. Using the rollout visualization, the VLM compares the two future trajectories and returns $z_k \in \{0, 1\}$, where $z_k = 1$ means the red (expert) trajectory is preferred and $z_k = 0$ means the blue (learner) trajectory is preferred.

Behavior cloning on expert-preferred states: When the VLM prefers the expert ($z_k = 1$), we treat a_t^{exp} as a local demonstration. We add a behavior cloning loss that encourages the policy to match these expert actions:

$$\mathcal{L}_{\text{BC}}(\phi) = \mathbb{E}\big[\mathbf{1}[z_k = 1]\, \|\pi_\phi(s_t) - a_t^{\text{exp}}\|_2^2\big], \tag{1}$$

using an ℓ_2 loss on steering and pedal commands.

Critic and policy updates: We maintain a critic $Q_\theta(s, a)$ that scores how compatible an action is with the VLM preference. For each transition we encourage Q_θ to give higher values to the preferred action and lower values to the non-preferred action, with a small margin, and we add a temporal-difference term so that these local preferences propagate along the trajectory, following [4]. The policy is then updated to select actions that the critic scores highly while staying close to expert-preferred behavior:

$$\mathcal{L}_\pi(\phi) = -\mathbb{E}\big[Q_\theta(s_t, \pi_\phi(s_t))\big] + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}}(\phi). \tag{2}$$

In practice we keep two replay buffers (expert-chosen vs. learner-chosen steps) and sample from them uniformly to avoid imbalance.

5 Experiments

We evaluate VLAPS in the MetaDrive driving simulator using the built–in urban scenarios with intersections, curved segments, and surrounding traffic. An episode is counted as a success only if the ego vehicle reaches the predefined goal without collision or going out of road boundaries.

We train a single policy π_ϕ while a fixed expert controller (RRT* + PD) runs in parallel and provides supervision through VLAPS. During training, the system interacts with the environment and updates π_ϕ every 200 environment steps. After each update block, we *evaluate* the current policy for 50 full episodes without intervention. This train–evaluate loop continues until a total of 50 000 environment steps are collected. All experiments are fully autonomous; no human actions are recorded.

6 Results and Discussion

Figure 4 (shown in Appendix 8.6) compares the learning curves reported for PVP4Real in the original paper with those obtained from our VLM-gated MetaDrive implementation. In both cases the success rate increases steadily over training, while crash and out-of-road rates decrease and the cumulative intervention curve gradually saturates. Our VLM-gated variant reproduces the same qualitative behaviour: the policy moves from near-zero success to a high-success regime, and the expert is queried frequently early on and then relied on less as the learned policy improves.

Table 1 summarizes final evaluation statistics for our run at $49\,950$ environment steps and for the approximate values read from the PVP4Real curves. Over 50 evaluation episodes without intervention, VLAPS achieves a success rate of 0.86 with zero crashes, an out-of-road rate of 0.08, and a route-completion rate of 0.935. These values are within a few percentage points of the ~ 0.90 success, near-zero crash, and $\lesssim 0.10$ out-of-road rates reported for PVP4Real. The total intervention cost in our run, 2.7×10^5, is of the same order as in the original human-supervised setting and slightly higher, consistent with a somewhat more conservative switching behaviour early in training.

Overall, these results indicate that replacing a human overseer with a VLM-based supervisor preserves the main learning dynamics of the PVP4Real pipeline: rapid improvement to high success with very low failure rates, together with a decreasing reliance on the expert controller over time. The training process is, however, fully automatic in our case: no human actions are recorded and supervision is provided entirely through batched VLM queries on rendered trajectory visualizations.

Metric	**Ours (50 episodes)**	**Baseline paper***
Success rate	0.86	~ 0.90
Crash rate	0.00	≈ 0
Out-of-road rate	0.08	$\lesssim 0.10$
Route completion	0.935	~ 0.95
Mean episode length	1.01×10^3 steps	n/a
Total intervention cost	2.7×10^5	slightly higher

Table 1: Final evaluation statistics of our MetaDrive run at $49\,950$ timesteps. Baseline values are read from Fig. 4 of [4]. Overall, our implementation matches reported performance trend.

7 Conclusion

We studied whether a vision–language model can stand in for the human supervisor used in reward-free, human-in-the-loop learning for safe driving. Building on the proxy value plus behavior cloning framework of PVP4Real, we introduced VLAPS, which combines an RRT* + PD expert controller, a learned policy, and a VLM that compares short-horizon trajectory rollouts and selects which controller to trust. The same VLM preferences are used to train a critic and to regularize the policy toward expert-preferred actions. In MetaDrive urban scenarios, our implementation achieves high success with very low crash and off-road rates, closely matching the performance and learning dynamics reported for the original human-supervised pipeline.

Our system has several practical limitations. The visualized waypoint rollouts use a relatively large integration step and approximate camera intrinsics, which can introduce numerical error, slow rendering, and occasionally mislead the VLM. Supervision quality is sensitive to prompt design, model choice, and query frequency: stronger VLMs and more frequent queries tend to improve decisions but also increase latency, cost, and the risk of bias if the expert identity is exposed. On the control side, we ultimately rely on an RRT* + PD expert because a non-convex MPC expert was too slow and sometimes infeasible in our setup. Future work includes improving the expert (e.g., kinodynamic RRT*, better tuning, and multi-rate execution of simulator, planner, and controller) and the supervisor (more accurate trajectory rendering, and simple state-estimation filters that reduce the need to re-simulate long rollouts under both policies).

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[3] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 45(3):3461–3475, 2022.

[4] Z. Peng, Z. Liu, and B. Zhou. Data-efficient learning from human interventions for mobile robots. *arXiv preprint arXiv:2503.04969*, 2025.

[5] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[6] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

8 Appendix

8.1 Team Contributions

Please note that all team members collaborated on all aspects of the project. The following descriptions only highlight each person's primary areas of focus.

Qizhao Chen: Implemented the policy visualization and VLM interface, integrated VLM-based gating into the training loop.

Debajyoti Chakrabarti: Designed and implemented the expert controller (RRT* + PD) and simulation setup.

This work is also in collaboration with Yuanhong Zeng, who provides feedback to our pipeline implementation as well as simulator setup.

8.2 Discrepancy Discussion

We completed all of the core promised components of the project, including the VLM-based gating mechanism, the expert policy, and the simulation evaluation of the learned agent.

However, there is one deviation from our original plan: in the project proposal, we intended to use a model predictive control (MPC) expert with explicit obstacle avoidance. In the final implementation, we instead use an RRT* planner combined with a PD tracking controller as our expert.

This change was caused by practical issues with the MPC setup. The MPC expert is model-based, and we found that model mismatch in the simulator. Together with the computational budeget of solving an MPC problem online, it introduced significant latency. This lag was further amplified when combined with VLM calls, making the overall control pipeline noticeably slower and less reliable. In contrast, the RRT* + PD expert is faster, more robust to modeling errors, and easier to integrate with our VLM-based supervision.

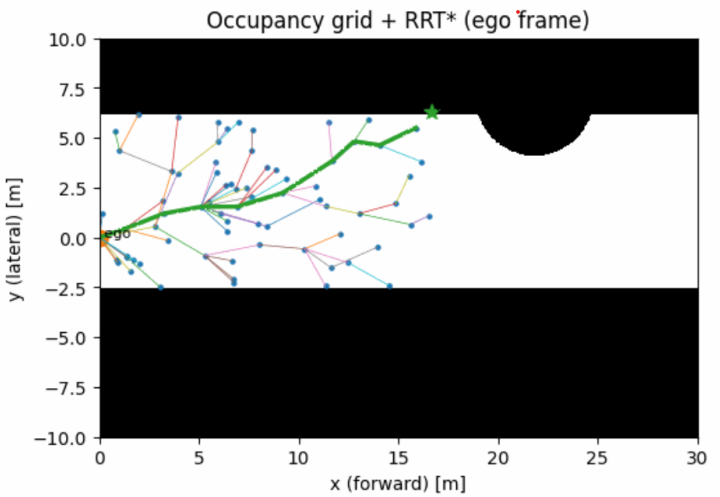

Figure 3: RRT* in the ego-centric occupancy grid. The ego vehicle starts at the origin on the left; the planner grows a tree (colored branches) inside the free space and returns a collision-free path (green) to the local goal (green star).

8.3 Result of Occupancy Grid and RRT* Plan

Fig. 3 illustrates the result of RRT* planner operating on the top-down ego-centric occupancy grid described in Sec. 4.1.1. The white region is the drivable lane, black cells are occupied or outside the lane, blue dots and colored line segments show the sampled tree, and the green polyline with star marks the final path from the ego vehicle to the local goal. The planning will happen every time step, which is every 0.1 seconds.

8.4 Discrete-Time Kinematic Bicycle Model

For completeness, we summarize the discrete-time kinematic bicycle model used for both control and visualization. The vehicle state at time step k is

$$s_k = (x_k, y_k, \psi_k, v_k)^\top,$$

where (x_k, y_k) is the position in the world frame, ψ_k is the yaw angle, and v_k is the longitudinal speed. The control inputs are the steering angle δ_k and longitudinal acceleration a_k, obtained from normalized actions $u_k = (u_{\text{steer},k}, u_{\text{pedal},k}) \in [-1, 1]^2$ via fixed scale factors:

$$\delta_k = u_{\text{steer},k}\, \delta_{\max}, \qquad a_k = u_{\text{pedal},k}\, a_{\max}.$$

Let L denote the wheelbase. With a control period Δt, the discrete-time update is

$$x_{k+1} = x_k + \Delta t\, v_k \cos \psi_k, \tag{3}$$
$$y_{k+1} = y_k + \Delta t\, v_k \sin \psi_k, \tag{4}$$
$$\psi_{k+1} = \psi_k + \Delta t\, \frac{v_k}{L} \tan \delta_k, \tag{5}$$
$$v_{k+1} = v_k + \Delta t\, a_k. \tag{6}$$

In our experiments we use $\Delta t = 0.1\,\text{s}$ for the actual environment dynamics, and a larger step $\Delta t_{\text{vis}} = 1.5\,\text{s}$ only for the image-space visualization.

8.5 VLM Gating Prompt

For reproducibility, we include the text prompt used when querying the VLM for policy switching. The user prompt is:

```
"Compare the expert policy (green path) and novice policy (red path) shown in the image.\n\n"
"First, output your decision strictly in this JSON format:\n"
"{{\"response\": <1 or 0>}}\n\n"
"Where:\n"
    " - 1 → The expert policy is **significantly better** than the novice policy.\n"
    " - 0 → The expert policy is **similar** to the novice policy.\n\n"
"Immediately after the JSON object, provide a short one-sentence explanation"
"for your decision.\n"
"Do not add any extra formatting or text before the JSON.".
```

8.6 Plot of Experiment Results

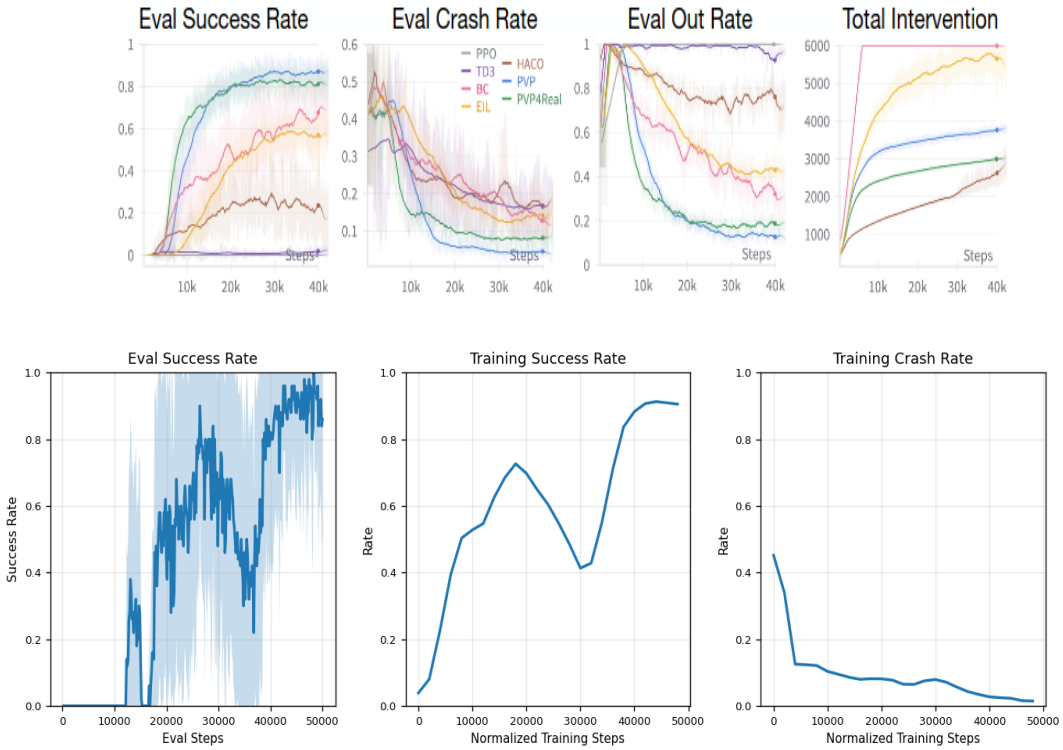

Figure 4: Top: performance of the original human-in-the-loop PVP4Real pipeline (Baseline paper [4]). Bottom: results from our VLM-gated MetaDrive implementation.