Latent Activation Editing: Inference-Time Refinement of Learned Policies for Safer Multirobot Navigation

Satyajeet Das[*], Darren Chiu, Zhehui Huang, Lars Lindemann[†], and Gaurav S. Sukhatme[†]

Abstract— Reinforcement learning has enabled significant progress in complex domains such as coordinating and navigating multiple quadrotors. However, even well-trained policies remain vulnerable to collisions in obstacle-rich environments. Addressing these infrequent but critical safety failures through retraining or fine-tuning is costly and risks degrading previously learned skills. Inspired by activation steering in large language models and latent editing in computer vision, we introduce a framework for inference-time *Latent Activation Editing* (LAE) that refines the behavior of pre-trained policies without modifying their weights or architecture. The framework operates in two stages: (i) an online classifier monitors intermediate activations to detect states associated with undesired behaviors, and (ii) an activation editing module that selectively modifies flagged activations to shift the policy towards safer regimes. In this work, we focus on improving safety in multi-quadrotor navigation. We hypothesize that amplifying a policy's internal perception of risk can induce safer behaviors. We instantiate this idea through a latent collision world model trained to predict future pre-collision activations, thereby prompting earlier and more cautious avoidance responses. Extensive simulations and real-world Crazyflie experiments demonstrate that LAE achieves statistically significant reduction in collisions (nearly 90% fewer cumulative collisions compared to the unedited baseline) and substantially increases the fraction of collision-free trajectories, while preserving task completion. More broadly, our results establish LAE as a lightweight paradigm, feasible on resource-constrained hardware, for post-deployment refinement of learned robot policies.

I. INTRODUCTION

Advances in robot learning have significantly pushed the boundaries of autonomy, including multi robot systems, driven primarily by both reinforcement learning (RL) and imitation learning [1]–[4]. Despite these successes, most learned models function as black boxes with limited interpretability and explainability [5]. Enhancing specific behaviors or addressing edge cases typically demands expensive retraining or fine-tuning, involving substantial real-world data or large amounts of simulated interactions [6]. Furthermore, retraining carries the risk of catastrophic forgetting, where policies lose previously learned skills when adapting to new or updated task specifications [7, 8]. Beyond forgetting, a broader limitation arises from the asymptotic performance plateaus often observed in RL policies. Once a policy achieves strong average performance, further optimization usually yields only marginal gains [9, 10]. Closing this final remaining performance gap (e.g., from 95% to 99.9%) is
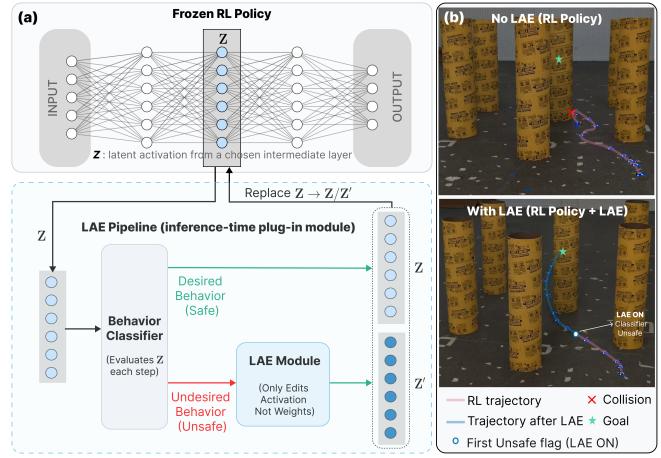
[*]Corresponding author.

[†]Equal co-advisors.

All authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA. {satyajee, chiudarr, zhehuihu, llindema, gaurav}@usc.edu

Fig. 1: **(a) Conceptual overview of LAE.** An online behavior classifier monitors the intermediate latents activation Z of a frozen RL policy. Safe activations pass unchanged, while unsafe ones are replaced by edited surrogate activation Z' generated by the activation editing module, without modifying policy weights. **(b) Real-world quadrotor navigation illustrating LAE behavior.** Without LAE, the RL policy collides with an obstacle. With LAE, the trajectory matches the base policy until the first unsafe flag, after which successive latent edits steer the quadrotor away from the unsafe zone, resulting in reaching the goal safely.

essential for robust real-world deployment [10]. These challenges motivate the need for methods that target refinement of specific policy behaviors, without incurring the costs and risks of full re-optimization.

Recent advances in natural language processing and computer vision have shown that the behavior of learned models can be modified at inference time without retraining [11]–[15]. Activation steering and representation engineering allow precise, inference-time interventions in large language models (LLMs) to guide outputs towards desired characteristics. Latent space editing in generative and diffusion models enables fine-grained control over generated content. The application of these ideas to robotics remains unexplored, primarily due to challenges associated with real-world interactions compared to text or static vision outputs.

Inspired by these advances, we propose a framework for altering the behavior of learned robot policies at inference time. Specifically, we focus on the problem of multi-quadrotor navigation in obstacle-rich environments, leveraging a pre-trained RL policy [2]. While this policy achieves strong overall performance, it continues to struggle in certain edge cases and more challenging scenarios. Further retraining or architectural changes do not alleviate these failures, underscoring the need for alternative approaches [2].

We investigate whether targeted latent activation editing

(LAE) (Figure 1) during inference can enhance safety, quantified through collision avoidance, without costly retraining or fine-tuning. We define LAE as the process of modifying hidden activations of a network during inference, without altering its trained weights. By intervening directly in the latent space, LAE temporarily adjusts the policy's internal representations to steer behavior along desired axes, such as safety (fewer collisions) in cluttered environments. Our key insight is that LAE is a promising mechanism to reduce undesirable behaviors or enhance specific desired behaviors in pre-trained models. The specific latent dimensions chosen for editing and the underlying editing logic vary depending on the behavior to be influenced.

LAE (Figure 1) operates in two stages. First, we identify undesired states by passing the selected intermediate latent activations through an online behavior classifier. Second, we perform targeted editing of these flagged latent activations using a principled strategy. To promote safer behavior, we hypothesize that artificially amplifying the robot's internal perception of environmental risk can trigger earlier and more cautious maneuvers, thereby improving collision avoidance. To realize this idea, we propose a latent collision world model (LCWM), an action-free latent world model [16]–[18] that predicts how latent activations evolve along trajectories leading to collisions, using the current activation together with a short history of past activations (Sec. IV-C). Our experiments show that among multiple baseline editing strategies, LCWM is the most effective, consistently yielding statistically superior safety performance across extensive evaluations. Finally, we demonstrate the real-world feasibility and effectiveness of our approach through deployment on Crazyflie quadrotors, establishing LAE as a practical and effective tool for enhancing the safety of pre-trained RL-based multirobot navigation policies.

Our key contributions are as follows:

- We present LAE, a novel plug-in framework that steers pre-trained policies by modifying intermediate activations at inference time, enabling targeted refinement of specific behaviors. This is the first activation-space intervention demonstrated on learned robot policies.
- We instantiate LAE on the task of navigating multiple quadrotors in cluttered environments, focusing on improving the collision avoidance behavior of a pretrained RL policy.
- We demonstrate the efficacy of LAE through large-scale simulation studies and real-world quadrotor experiments, achieving statistically significant safety improvements while remaining feasible on highly resource-constrained robots.
- Ablation studies show that effective latent editing must preserve activations relating to the robot's own dynamics to avoid dynamically infeasible behaviors.

II. RELATED WORK

A. Latent Space Editing: Altering Learned Model Behavior

Neural networks learn semantically structured internal representations that can be located and modified. Early work in natural language processing showed linear regularities and biases in word embeddings and even a "sentiment neuron" emerging under next-token training [19, 20]. In vision, self-supervised models exhibit emergent semantics such as segmentation and depth [21], motivating linear probes [22] and concept directions [23] to identify editable features. Building on this structure, latent-space editing techniques in vision successfully steer attributes by moving along interpretable directions in generative models [14, 24] and diffusion-based guidance [15, 25]. In LLMs, behavior can be steered via (i) prompt based methods [26], (ii) decoding-time control [27], (iii) weight-level knowledge editing [28], or (iv) activation-space interventions that directly modify hidden activations during inference, such as activation addition and representation engineering [11]–[13]. Complementing these editing methods, representation factorization shows that activations can be decomposed into sparse, interpretable features: classical dictionary learning and sparse coding provide the foundation [29], and modern sparse autoencoders recover monosemantic features that enable causal ablations and steering [13, 30, 31]. Here, we present the first application of ideas from activation interventions to robotics.

B. Latent Representations for Safe Reinforcement Learning.

Recent work has explored latent representations to improve robot safety in reinforcement learning. Ls3 [32] constructs latent safe sets to constrain exploration during training. LSPC [33] uses conditional variational autoencoders (VAEs) to enforce offline latent constraints. SLAC [34] augments latent actor-critic models with a safety critic for cost-constrained training, and Latent Safety Filters [35] apply reachability analysis in learned latent spaces to avoid unsafe actions. These approaches enforce safety through training-time constraints/modified architectures; we intervene directly at inference by editing activations of a frozen policy.

C. Safe Multi-Quadrotor Navigation

Safety for multi-quadrotors has been pursued via model-based methods like (i) real-time planning / nonlinear model predictive control (NMPC) and (ii) control barrier functions (CBFs). Distributed NMPC exchanges neighbors' predicted trajectories at each control step and solves a constrained NMPC locally. This introduces communication delays and solver costs that grow with neighborhood size, horizon length, and active constraints, while assuming low-latency neighbor state information and an obstacle model [36]. Decentralized planners such as EGO-swarm [37] achieve millisecond-level onboard replanning via a broadcast/chain network and local mapping, but would require nontrivial adaptation to interface with a learned policy. CBF filters solve an online quadratic program (QP) to enforce forward invariance; the seminal multi-robot CBF [38] and graph CBF variants [39] provide guarantees but add a runtime optimization layer, rely on accurate ego/neighbor/obstacle states, and, when paired with RL, typically require a change of the policy architecture. Moreover, per-agent QPs and neighbor exchanges become a bottleneck as team size or constraints

grow. In contrast, LAE is model-free and improves safety without retraining, architectural changes, or limits on the number of robots or environment clutter.

III. PROBLEM FORMULATION

We consider a decentralized multi-quadrotor navigation task in an obstacle-rich environment. Each robot i receives local observations $O_{t,i}$ (Sec. V-B) at time t consisting of its own state $O_{t,self}$, the relative states of its nearest neighbors $O_{t,neigh}$, and a signed distance field encoding of nearby obstacles $O_{t,obst}$ (Figure 2). A pre-trained RL policy π_ϕ (specifically, the policy from [2]) maps observations to rotor thrusts, $a_{t,i} = \pi_\phi(O_{t,i})$, enabling each robot to navigate to its goal while avoiding collisions. Although π_ϕ achieves strong overall performance, there is room for improvement in the collision rate.

Our objective is to reduce collisions in π_ϕ without modifying its weights or architecture, while maintaining its goal-reaching behavior. Let $Z_t \in \mathbb{R}^d$ denote an intermediate latent representation of the policy at time t where d is the latent dimension. We seek an inference-time transformation $Z_t' = \mathbf{E}_\theta(Z_t) \in \mathbb{R}^d$ that selectively replaces unsafe activations Z_t with edited surrogates Z_t', such that the resulting actions computed by the original policy with its intermediate latent Z_t substituted by Z_t' lead to reduced collisions while preserving goal-reaching performance.

IV. METHODOLOGY

We propose an inference-time framework to modulate the behavior of a pre-trained mulit-quadrotor RL policy without altering its weights. As shown in (Figure 2), at each timestep we extract the latent activation Z_t from an intermediate encoder layer of the policy. This latent is evaluated by a behavior classifier; if it is predicted to correspond to a safe state, the policy continues unaltered. If it is predicted to be unsafe, it is replaced by an edited version Z_t' generated by the latent editing module, which is then forwarded to the downstream layers of the frozen RL policy. The policy's outputs can thus be selectively steered toward safer behaviors while retaining the original architecture and weights. The overall method consists of three components: dataset collection, behavior classifier, and latent activation editing.

A. Dataset Collection

To construct the dataset, we roll out the trained policy in the QuadSwarm simulator [40]. At each policy step, we record only the intermediate latent activation $Z_t \in \mathbb{R}^d$, together with the trajectory index τ and time index t. For each trajectory τ ($\tau = 1, \ldots, K$) of length T, we mark all collision time indices t_c in the set $\mathscr{C}^{(\tau)}$. Because collision times are precisely observable, we use a *time-to-collision* heuristic with horizon H to obtain safety labels as

$$Y_t^{(\tau)} = \begin{cases} \text{unsafe} & \text{if } \exists t_c \in \mathscr{C}^{(\tau)} \text{ with } 0 \leq t_c - t \leq H, \\ \text{safe} & \text{otherwise.} \end{cases} \quad (1)$$
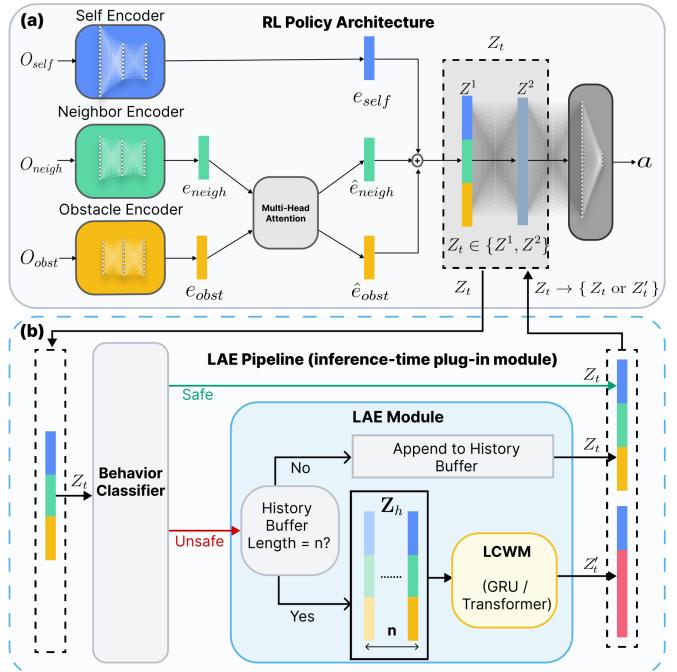

Fig. 2: Overview of LAE integrated on a pre-trained multi-quadrotor navigation policy. **(a) RL policy architecture:** observations are encoded and fused via multi-head attention to produce intermediate latent activations (Z^1, Z^2), which serve as candidates for Z_t. **(b) editing pipeline:** a behavior classifier evaluates latent activation Z_t and forwards it unchanged if safe, or routes it to the editing module if unsafe. The module maintains a short history buffer and, once filled, invokes the LCWM to generate a surrogate Z_t' that replaces the unsafe latent activation Z_t.

We label a latent *unsafe* if it lies within H steps of any logged collision, and *safe* otherwise. This results in labeled dataset

$$\mathscr{D} = \bigcup_{\tau=1}^{K} \{(Z_t^{(\tau)}, Y_t^{(\tau)})\}_{t=0}^{T-1}, \qquad Z_t^{(\tau)} \in \mathbb{R}^d, Y_t^{(\tau)} \in \{\text{safe}, \text{unsafe}\} \quad (2)$$

which is used to train the behavior classifier and to construct sequence data for training the LAE (LCWM) module.

B. Behavior Classifier

The behavior classifier $\mathbf{B}_w$ is a neural network trained with supervised learning on $\mathscr{D}$ to detect unsafe activations. It implements a mapping

$$\mathbf{B}_w : Z_t \rightarrow \{\text{safe}, \text{unsafe}\}, \quad (3)$$

and we denote its prediction by $\hat{Y}_t = \mathbf{B}_w(Z_t)$. We use a multi-layer perceptron with batch normalization, ReLU activations, and dropout. At inference, if $\hat{Y}_t = \text{safe}$ the policy continues normally, i.e., $Z_t' = Z_t$. If $\hat{Y}_t = \text{unsafe}$, the latent activation is passed to the editing module. Although we focus on collision avoidance behavior, the labeling scheme is flexible; we will explore applying the same machinery to other behavioral dimensions by re-labeling $\mathscr{D}$ in future work.

C. Latent Activation Editing (LAE)

The editing module $\mathbf{E}_\theta$ realizes our hypothesis that artificially amplifying the policy's internal perception of risk can induce earlier and safer avoidance maneuvers. To put

this into practice, we need a principled way to synthesize *risk-amplified latent activations* consistent with the policy's latent evolution, which in turn motivates learning a model capable of predicting near-future latent activations. A *world model* typically refers to a learned predictor of environment dynamics, approximating the transition function $f : (s_t, a_t) \mapsto s_{t+1}$ or, in latent form, $f : (Z_t, a_t) \mapsto Z_{t+1}$, and has been widely studied for planning and imagination in model-based RL [16]–[18]. Latent world models aim to capture the transition dynamics of the environment in the latent space. However, in the multi-quadrotor navigation domain, modeling the full latent evolution proved infeasible due to partial observability, the coexistence of static and dynamic obstacles, and highly non-stationary latent dynamics.

To address these challenges, we introduce a *latent collision world model (LCWM)* that focuses exclusively on latent transitions leading to collisions. Whereas conventional world models first encode observations and condition on actions, our policy already produces latent representations, allowing us to bypass the encoder and operate directly on these activations. Unlike action-conditioned world models, LCWM leverages short histories of latents to capture the relevant latent evolution without explicit actions. Similar "action-free" latent world models have recently been proposed [41], showing that meaningful transition structure can be inferred without ground-truth actions. To this end, we train LCWM that predicts future latents along trajectories leading to collisions, using only a short history of past activations, thereby operationalizing our safety hypothesis.

Dataset Preparation for LCWM. We train LCWM using only collision-bearing trajectories (those with $|\mathscr{C}^{(\tau)}| > 0$, Sec. IV-A). For each trajectory τ and collision time $t_c \in \mathscr{C}^{(\tau)}$, we consider the pre-collision window $[t_c - H, t_c]$. For every index t in this window, we form an n-step history buffer of latent activations $\mathbf{Z}_h = [Z_{t-n}, \ldots, Z_t] \in \mathbb{R}^{d \times n}$, where n denotes the buffer length. To avoid predicting beyond the collision instant, we clamp the forecast index to $t^\star = \min(t+m, t_c)$, where where m defines the number of steps to predict into the future. Each $\mathbf{Z}_h$ is then paired with the target $Z_{t^\star}$, where Z_{t+m} is latent activation m steps ahead of Z_t, and Z_{t_c} is latent activation at the time of collision. The LCWM is trained by minimizing the objective in Equation 4.

$$\mathscr{L}(\theta) = \sum \left\| \mathbf{E}_\theta(\mathbf{Z}_h) - Z_{t^\star} \right\|_2^2. \tag{4}$$

Model. We primarily implement $\mathbf{E}_\theta$, the LCWM, as a gated recurrent unit (GRU). Given input sequence $\mathbf{Z}_h$, the GRU evolves hidden states h_i as:

$$h_i = \text{GRU}(h_{i-1}, Z_i), \quad Z'_t = W h_t + b, \tag{5}$$

where $i = t-n, \ldots, t$ and Z'_t is the predicted future latent. During inference, when $\mathbf{B}_w$ flags Z_t as unsafe and a buffer of length n is available, the GRU predicts Z'_t, which replaces Z_t in the forward pass. If the classifier outputs safe, the buffer is reset. We also implemented LCWM with a transformer-based predictor (Table I), but the GRU variant provided slightly better empirical performance in our setting.

Algorithm 1 Inference-time LAE
___
1: Predict $\hat{Y}_t \leftarrow \mathbf{B}_w(Z_t)$
2: **if** $\hat{Y}_t =$ safe **then**
3:     Forward latent activation, Z_t; reset history buffer
4: **else**
5:     Append Z_t to history buffer
6:     **if** buffer length is n **then**
7:         $Z'_t \leftarrow \mathbf{E}_\theta(\mathbf{Z}_h)$
8:         Forward new latent activation, (Z'_t)
9:     **end if**
10: **end if**
___

Inference Loop The overall procedure is summarized in Algorithm 1. At each step, the Z_t is classified and either passed through or edited depending on predicted safety.

While demonstrated for collision avoidance, the framework is general: expert labeling defines the axis of intervention, the classifier acts as a representation reader, and the editor serves as a representation controller. Thus, our method can extend beyond safety to modulate other behaviors of pre-trained robot policies.

V. EXPERIMENTS

A. Experimental Setup

We evaluate our LAE pipeline through a combination of simulation experiments and real world deployment. In simulation, we use the QuadSwarm simulator [40] with 8 quadrotors navigating a $10 \times 10 \times 10$ m room containing static obstacles (20% density, 0.6 m diameter). To ensure fair comparisons, we modified the simulator to be deterministic, such that identical initializations yield identical trajectories. This prevents stochastic variation and allows improvements to be attributed directly to editing. In the default setting, major sources of stochasticity include motor & sensor noise in QuadSwarm [40] and action sampling in Sample Factory [42]. These components are important during training to promote policy robustness and sim-to-real transfer, but for evaluation we fix these sources to obtain reproducible comparisons (for further details see [40, 42]). Unless otherwise noted (Fig. 4), all simulation experiments are conducted in the deterministic simulator setting. We construct a set of 2,600 environment configurations where the base policy collides at least once, and replay each configuration with and without LAE for evaluation. The behavior classifier $\mathbf{B}_w$ achieves high accuracy (overall $\sim$98%, and 99.3% on safe trajectories), and with the additional history-buffer requirement, LAE rarely activates on already safe trajectories, where its performance is effectively identical to the baseline RL policy. Evaluation is therefore restricted to these 2600 configurations to avoid inflating results with already safe scenarios. Performance is assessed using three metrics: **total collisions**, the cumulative number of total collisions across all 2600 trajectories; **zero collision trajectories**, the number of trajectories completed without any collision; and **average success rate**, the fraction of robots that reach their goals without collision, capturing task completion alongside safety.
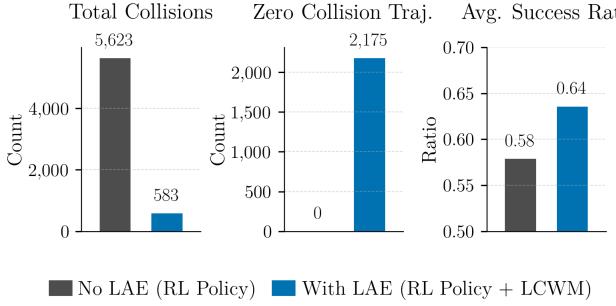
Fig. 3: Quantitative comparison of the base RL policy with and without LAE on 2,600 environmental configurations. We report total collisions, zero collision trajectories, and average success rate. LAE implementation uses LCWM (GRU).

The remainder of this section is organized as follows. Sec. V-B describes the architecture and pre-training of the base RL policy. Sec. V-C presents the core comparison between the base RL policy and our best LAE strategy - LCWM. Sec. V-D evaluates alternative baseline LAE strategies. Sec. V-E investigates two key design questions: *which latent components should be edited* and *when editing should be triggered* (i.e., how many steps before a collision). Sec. V-F shows real-world deployment on Crazyflie quadrotors.

B. Base RL Policy (Architecture and Pre-training)

We adopt the end-to-end, decentralized RL policy of Huang et al. [2] as the base RL controller for the multi-quadrotor navigation task. Each robot (Figure 2) observes its own state and goal ($O_{t,self}$), the relative positions and velocities of its two nearest neighbors ($O_{t,neigh}$), and a compact 3×3 signed-distance field encoding of nearby obstacles ($O_{t,obst}$). These observations are encoded by three two-layer MLPs (self, neighbor, obstacle), with neighbor and obstacle embeddings fused through a multi-head attention module. The concatenated embedding produces a latent representation $Z^1 = [e_{self}, e_{neigh}, e_{obst}] \in \mathbb{R}^d$, which is then passed through downstream MLP layers to yield another latent $Z^2 \in \mathbb{R}^d$. Finally, Z^2 is fed into the actor's action parameterization head to output four normalized rotor thrusts. In our experiments, we instantiate Z_t as either Z^1 or Z^2, i.e., $Z_t \in \{Z^1, Z^2\}$, which serve as candidate editing points for LAE (see Sec. V-E.1). The frozen RL policy used in our experiments has latent dimension $d = 30$.

Training uses the asynchronous version of decentralized, independent PPO (IPPO) implemented in Sample Factory [42], in a randomized 10 m × 10 m × 10 m simulated room with static obstacles and goal curricula in the QuadSwarm simulator [40]. The reward encourages goal progress and penalizes robot–robot and robot–obstacle collisions and near-misses together with control regularization. A key ingredient is a collision-focused replay curriculum that buffers 1.5 s pre-impact windows at elevated sampling rates while capping long failure episodes, which reduces collision rates versus baseline variants. The learned policy scaled in simulation to 32 robots at obstacle densities up to 80 percent and transfers zero-shot for real-world deployment with Crazyflie 2.1. For
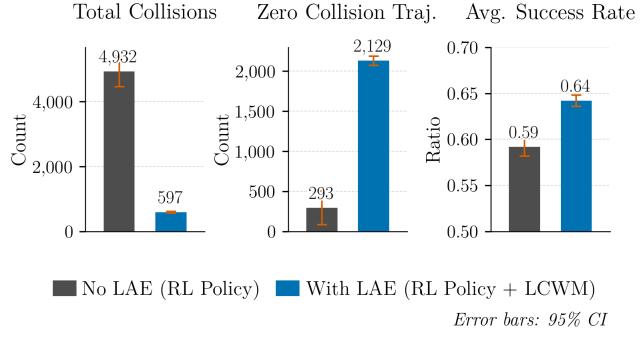
Fig. 4: Quantitative comparison of the base RL policy with and without LAE in the non-deterministic simulator setting. We evaluate on the same 2,600 configurations with identical start–goal and obstacle locations as in the deterministic setting, but report averages over 10 stochastic runs. Error bars denote 95% confidence intervals across runs. LAE implementation uses LCWM (GRU).

more details on architecture, training setup and base-policy performance refer to Huang et al. [2].

C. Core Results

Figure 3 presents the quantitative comparison between the base RL policy and LCWM (GRU), our best-performing instantiation of the LAE. The base policy incurs 5,623 collisions across the set of 2600 configurations, with no zero collision trajectories and an average success rate of 0.58. With LCWM, collisions reduce to 583 (-89.6%), 2,175 trajectories (82.7%) are collision-free , and the average success rate increases to 0.64 ($+10.3\%$ relative). This reduction in collisions is statistically significant: a paired t-test over 2,600 configurations shows a mean per-run reduction of 1.94 (95% CI [1.86, 2.01]), $p < 10^{-300}$, Cohen's $d = 1.0$. These results demonstrate that inference-time LAE substantially improves safety while preserving task performance. In our experiments, we found $n = 3$ to suffice, although it is task dependent. The specific choices of Z_t, horizon H, and prediction horizon m follow the best-performing configurations identified in the ablations (Sec. V-E).

While all primary evaluations in this paper are conducted in a deterministic simulator setting to ensure fair comparisons (Sec. V-A), we also performed a complementary test in the non-deterministic (stochastic) setting. This evaluation confirms that the observed improvements persist, with Figure 4 showing similar trends: LAE substantially reduces total collisions and increases zero collision trajectories and average success rate, with error bars (95% CI) indicating consistent performance across 10 independent runs.

To complement these aggregate statistics, Figure 5 shows representative trajectory comparisons. Because editing is only triggered when unsafe states are detected, trajectories remain identical to the base policy whenever the robots operate in safe regions. When a collision is imminent, LCWM intervenes to adjust the latent state, allowing the robots to avoid the obstacle. Importantly, the agents still reach their goals, showing that the safety improvements do not come at the expense of goal-reaching behaviour.

Method	Model	Total Collisions ↓	Zero Collision Traj. ↑	Avg. Success Rate ↑
Base RL Policy (No Editing)	–	5,623	0	0.58
KD-Tree Retrieval	–	2,678	1,307	0.61
Sparse Autoencoders (SAE)	–	2,896	1,320	0.62
Encoder–Decoder Projections	UMAP	3,766	491	0.59
	Barlow-Twins	3,355	919	0.61
	AE	37,749	215	0.59
Latent Collision World Model (LCWM)	Transformer	612	2,062	0.63
	GRU	**583**	**2,175**	**0.64**

TABLE I: Comparison of alternative editing strategies on 2600 diverse environment configurations. LCWM (GRU) achieves fewest collisions and the most zero collision trajectories while maintaining success rate. Other approaches offer partial gains but are less effective.
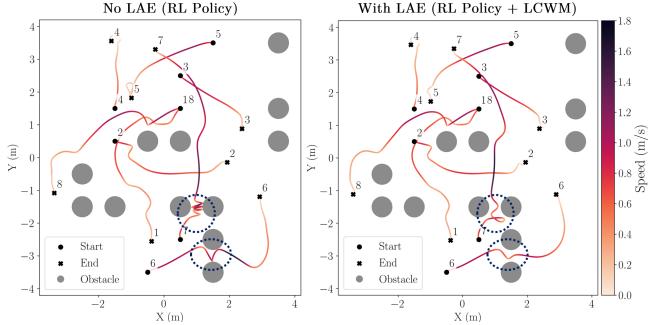

Fig. 5: Representative trajectory comparison. **Left**: base RL policy (no LAE), which collides with obstacles. **Right**: RL policy with LAE, which avoids collisions while remaining identical to the base policy in safe regions and still reaching the goals. Trajectories are coloured by speed. Numbers denote quadrotor index.

D. Comparative Evaluation

We next compare LCWM against a range of alternative baseline LAE strategies. For each baseline, we conducted reasonable hyperparameter sweeps and report the best-performing setting we identified. While we do not claim these are globally optimal, weaker configurations performed substantially worse, and we found no settings that altered the qualitative trends reported here. The complete results are summarized in Table I, and we briefly discuss the motivation, design, and outcomes of each method.

1) KD-Tree Retrieval: As a non-parametric test of our LAE hypothesis for safer behavior, we built KD-trees over latent activations indexed by time-to-collision and used them to replace unsafe latents activations with their nearest neighbors drawn from trajectories leading up to a crash. This serves as a dataset-driven approximation of LCWM, rather than predicting future unsafe latents with a model, it simply recalls them from a dictionary of previously observed unsafe progressions. KD-tree retrieval reduces collisions compared to the base policy (2,678 vs. 5,623), showing that even simple memorization of unsafe evolution provides some benefit. However, its reliance on stored examples prevents it from generalizing to unseen states, leaving it well behind the learned LCWM (583 total collisions).

2) Sparse Autoencoders (SAE): SAE have become popular in AI safety [13, 31], where they decompose activations into interpretable units that can be selectively modulated. Following this idea, we trained SAEs on unsafe latents to identify neurons correlated with unsafe behavior and applied standard steering interventions on these units [13, 31]. While this improved safety relative to the base policy (2,896 vs. 5,623 collisions), it remained way less effective than LCWM. A key limitation is that in robotics, unsafe behaviour is distributed across entangled latent dimensions, and may not be cleanly separable into individual sparse units as sometimes observed in LLMs or vision models. As a result, SAE performance was highly sensitive to hyperparameters such as such as dictionary size, sparsity penalty (λ), and the choice of units targeted for editing. This fragility manifested in large performance variance, with some configurations yielding modest gains while others produced unstable or infeasible behaviour. One likely contributor is that some of the units selected for steering also encode aspects of robot's own dynamics; modifying them can disrupt the underlying stability of the controller and lead to catastrophic behaviour (Sec. V-E.1). Overall, SAEs demanded extensive tuning yet consistently underperformed the LCWM approach, highlighting their limited suitability.

3) Encoder–Decoder Projections: We next explored whether projecting latents into a lower-dimensional representation could act as a form of editing by implicitly mapping unsafe states closer to the manifold of safe activations. We tested parametric UMAP (structure-preserving) [43], Barlow Twins (self-supervised disentanglement) [44], and a standard autoencoder. UMAP (3,766 collisions) and Barlow Twins (3,355 collisions) provided modest improvements, but the autoencoder collapsed completely (37,749 collisions). Overall, these results indicate that while encoder–decoder projections reshape the representation space, compression alone is insufficient for effectively reducing unsafe behavior.

4) Latent Collision World Model Variants: The LCWM can be instantiated using different temporal latent predictors, but the input output structure and underlying hypothesis remain the same. We evaluate two widely used temporal architectures: GRUs, a popular and lightweight variant of recurrent neural networks (RNNs), and transformers. The GRU-based LCWM achieved the best overall performance (583 collisions, 2,175 zero-collision episodes), reflecting the suitability of recurrent models for short horizons under tight compute budgets. A Transformer-based LCWM achieved comparable safety (612 collisions) but incurred substantially higher computational cost, limiting its practicality for real-
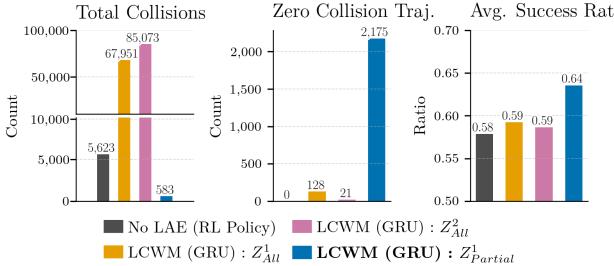
Fig. 6: Choice of Z_t: Comparing Z^1_{All}, Z^2_{All}, and $Z^1_{Partial}$. $Z^1_{Partial}$ offers the best safety–performance trade-off.

time deployment on Crazyflie hardware. These results highlight that while both architectures can realize the LCWM hypothesis, lightweight RNN such as GRUs provide the best balance of effectiveness and efficiency in our setting.

In summary, while alternative strategies offer partial gains, only LCWM consistently delivered strong safety improvements at a feasible computational cost.

E. Ablation Studies

1) Which Latent to Edit?: Since LAE operates by modifying an intermediate latent activation Z_t, the choice of Z_t is critical and a key design question. As described in Sec. V-B, we consider two candidate latents: Z^1, the fused latent obtained by concatenating the self embedding with the attention-modulated neighbor and obstacle embeddings, and Z^2, a downstream latent before the action head. We evaluate three editing strategies: (i) Z^1_{All}: editing the entire Z^1 latent activation, (ii) $Z^1_{Partial}$: editing only the neighbor and obstacle components of Z^1 while leaving the self-dynamics components untouched, and (iii) Z^2_{All}: editing the entire Z^2 latent activation. The rationale for $Z^1_{Partial}$ is that Z^1 retains a clean separation between self-dynamics and environment features, making selective editing possible. In contrast, Z^2 has already passed through a feedforward transformation and nonlinearity, which entangles these features and prevents a comparable split. As shown in Figure 6, restricting edits to $Z^1_{Partial}$ yields the best results, reducing collisions to 583, producing 2,175 zero-collision episodes, and raising success to 0.64. In contrast, editing the entire Z^1 (Z^1_{All}) leads to unsafe and dynamically infeasible trajectories, with performance degrading to 67,951 collisions and only 128 zero-collision episodes. An even stronger failure mode occurs when editing the downstream latent activation Z^2 (Z^2_{All}), the subsequent hidden layer after a feedforward transformation and nonlinearity. These results establish a clear guideline: effective latent editing must preserve neurons carrying self–dynamics information, as indiscriminate modification of robot's own dynamics can produce unsafe or infeasible behavior.

2) Effect of Editing Horizon H: The classifier horizon H determines how many steps before a collision are labeled as unsafe and thus trigger editing. Figure 7 shows a clear trend. Short horizons ($H=50$) intervene too late, leaving many collisions unresolved (1,124 total). Increasing to $H=100$ reduces collisions substantially (714 total). For $H \in [150, 300]$, the safety and success metrics stabilize
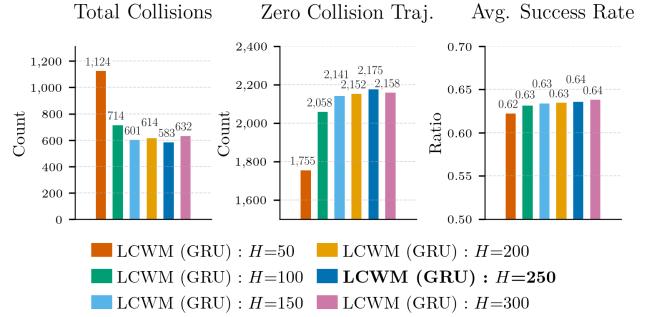

Fig. 7: Effect of editing horizon H for LAE with LCWM (GRU).

with only marginal changes, with $H=250$ offering the best overall trade-off (lowest collisions at comparable and highest success rate), whereas $H=300$ shows a slight increase in collisions. Concretely, total cumulative collisions drop from 1124 at $H=50$ to 583 at $H=250$ and total zero collision trajectories increase from 1755 at $H=50$ to 2175 at $H=250$ with a success rate of 0.64. We therefore adopt $H=250$ as the default value. These results confirm that the timing of editing is an essential hyperparameter: even with the same model, choosing H incorrectly can limit effectiveness, and the optimal value will depend on the behavior being edited.
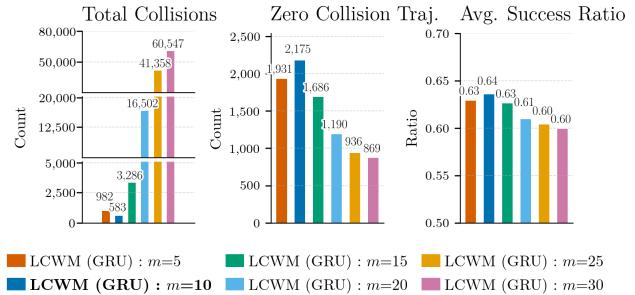

Fig. 8: Choice of prediction horizon m for LCWM.

3) Choice of Prediction Horizon m: The LCWM operates by predicting future latent activations over a fixed prediction horizon m, which specifies how many steps into the future the model predicts. A sufficient lookahead is required to realize our safety hypothesis: by anticipating the near-future evolution of unsafe states, the LCWM can amplify collision-related activations and provide the inflated risk signal needed to trigger earlier avoidance. However, predicting too far ahead risks departing from the policy's latent dynamics, leading to unreliable or destabilizing edits.

Figure 8 shows the effect of varying $m \in \{5, 10, 15, 20, 25, 30\}$. Among the tested values, $m=10$ yields the best overall performance, with 583 total collisions, 2,175 zero-collision trajectories, and an average success rate of 0.64. A shorter horizon such as $m=5$ also improves performance (1,931 zero-collision trajectories, success rate 0.63), but longer horizons degrade results consistently: starting at $m=15$ collisions increase and success decreases, with the effect becoming pronounced at $m \geq 25$ (e.g., 60,547 collisions at $m=30$).
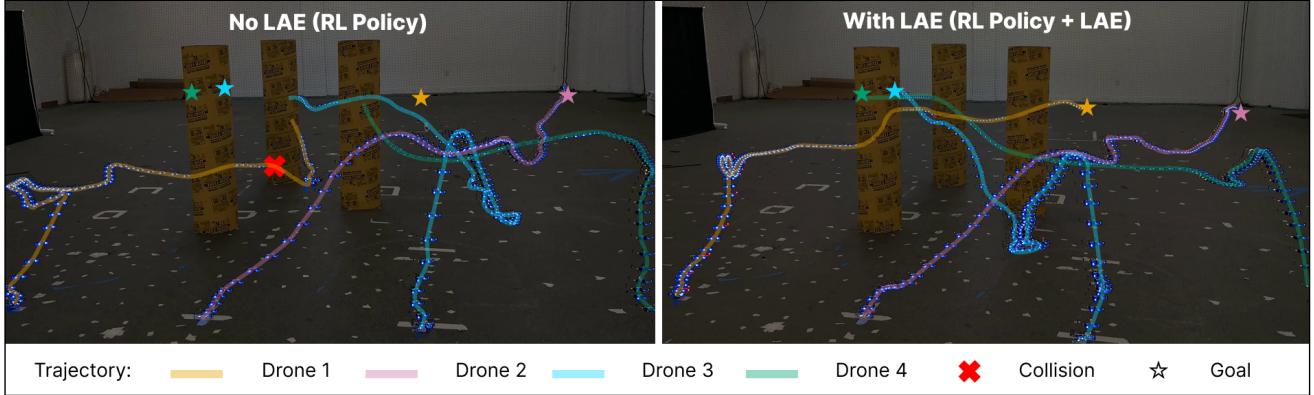
Fig. 9: Real-world deployment with 4 Crazyflie quadrotors navigating among cylindrical obstacles (**bilateral crossing**). **Left:** with the baseline RL policy, Drone 1 collides with an obstacle, leading to task failure. **Right:** with LAE enabled, all drones avoid collisions and reach their goals.
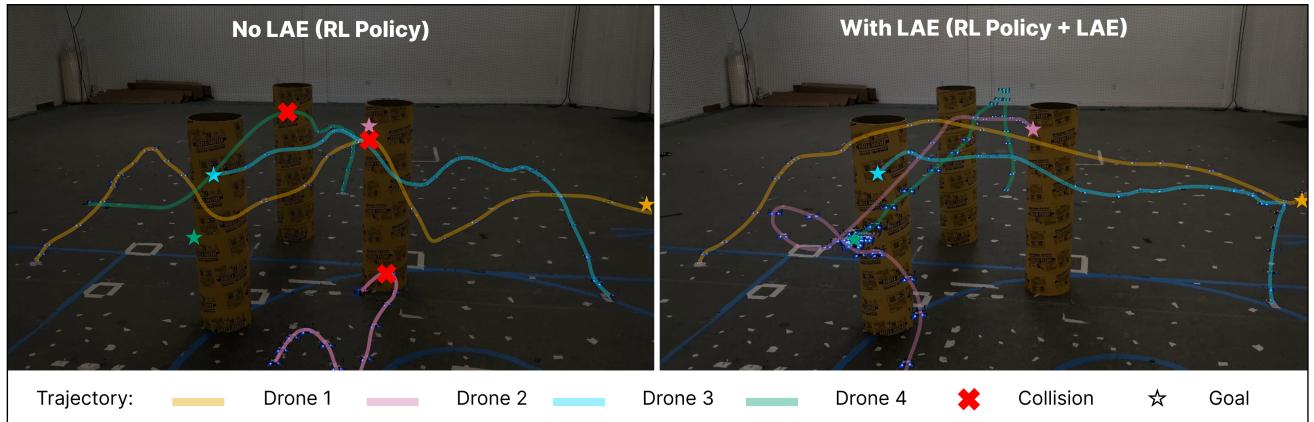

Fig. 10: Real-world deployment with 4 Crazyflie quadrotors navigating among cylindrical obstacles (**four-way crossing**). **Left:** with the baseline RL policy, Drones 1, 2, and 4 collide with obstacles, leading to task failure. **Right:** with LAE enabled, all drones avoid collisions and reach their goals.

These results indicate that the prediction horizon m is a critical hyperparameter: very short horizons limit foresight, while overly long horizons destabilize predictions. A moderate setting around $m=10$ provides the most favorable trade-off, and we adopt it as the default in for all experiments.

F. Real-World Deployment

To validate whether LAE remains effective beyond simulation, we deploy the RL policy together with our LAE module (trained exclusively in simulation) on multiple Crazyflie 2.1 quadrotors. The LAE module consists of a compact two-layer MLP classifier with 64 hidden size ($\sim$2k parameters) and a lightweight GRU editor with hidden size 32 ($\sim$7k parameters), both re-implemented in C for real-time execution on the Crazyflie's STM32 microcontroller. Together, these networks contain fewer than 10k parameters (under 40 kB in float32) and add less than 1 ms latency per step, making them fully compatible with the 1 kHz stabilization loop and demonstrating the feasibility of LAE on a severely resource-constrained platform. Each quadrotor performs onboard localization through optical flow and broadcasts its estimated state to neighbors over a low-latency radio link. Obstacles are known *a priori*, from which a local SDF (2*m*

range) is generated online. All computation, including state estimation, inter-quadrotor communication, LAE, and policy inference, runs fully onboard at 100 hz.

The quadrotors are deployed in an indoor environment with cylindrical obstacles, using identical start and goal positions across trials to ensure fair comparison. We first illustrate the LAE mechanism on a single quadrotor (Figure 1 b). The baseline RL policy collides with an obstacle, whereas with LAE the trajectory remains identical to the base policy until the first unsafe state is flagged; from that point onward, successive latent edits steer the quadrotor away from the unsafe zone while still reaching the goal safely. We then demonstrate scalability in multi-agent settings. In the 4-quadrotor bilateral crossing task (Figure 9), the baseline RL policy produces collisions, whereas the LAE-augmented policy consistently steers the robots safely around obstacles while still reaching their goals. Additional experiments (Figure 10) validate robustness in a more challenging four-way crossing setup, where quadrotors approach from all sides of the arena and must swap positions without collision. Across both scenarios, our LAE framework preserves the base policy's goal-reaching performance while significantly reducing collisions. These results confirm that inference-

time LAE is not only effective in large-scale simulation but also deployable as a practical, real-time safety layer for multi-quadrotor navigation on severely resource-constrained hardware.

VI. CONCLUSION

We introduce LAE, an inference-time framework for refining the behavior of pre-trained policies without retraining or architectural modifications. Focusing on multi-quadrotor navigation, we show that LAE substantially reduces collisions by intervening directly in intermediate latent representations. We hypothesize that amplifying collision-related activations induces more cautious maneuvers and instantiate this idea through an LCWM that predicts and replaces unsafe activations. Across large-scale simulations and real-world Crazyflie deployments, LAE yields statistically significant reductions in collisions while preserving task performance, establishing activation editing as a lightweight, effective, and practically feasible paradigm for post-deployment refinement of robotic control policies toward safer behavior.

While we instantiate LAE for reducing collisions for multi-quadrotor navigation, the framework is not inherently limited to this setting. In future work, we plan to extend the approach to other behavioral axes, different platforms, and varied task domains.

REFERENCES

[1] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual review of control, robotics, and autonomous systems*, vol. 3, no. 1, 2020.

[2] Z. Huang, Z. Yang, R. Krupani, B. Şenbaşlar, S. Batra, and G. S. Sukhatme, "Collision avoidance and navigation for a quadrotor swarm using end-to-end deep reinforcement learning," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024.

[3] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, "Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning," in *Conf. on robot learning*, 2022.

[4] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," *arXiv preprint arXiv:2408.03539*, 2024.

[5] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, no. 3, pp. 31–57, 2018.

[6] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv preprint arXiv:1709.10087*, 2017.

[7] T. Schmied, M. Hofmarcher, F. Paischer, R. Pascanu, and S. Hochreiter, "Learning to modulate pre-trained models in rl," *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[8] M. Wolczyk, B. Cupial, M. Ostaszewski, M. Bortkiewicz, M. Zajac, R. Pascanu, L. Kucinski, and P. Milos, "Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem," *arXiv preprint arXiv:2402.02868*, 2024.

[9] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[10] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv:1904.12901*, 2019.

[11] A. M. Turner, L. Thiergart, G. Leech, D. Udell, J. J. Vazquez, U. Mini, and M. MacDiarmid, "Steering language models with activation engineering," *arXiv preprint arXiv:2308.10248*, 2023.

[12] A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, A. Pan, X. Yin, M. Mazeika, A.-K. Dombrowski *et al.*, "Representation engineering: A top-down approach to ai transparency," *arXiv preprint arXiv:2310.01405*, 2023.

[13] A. Templeton, *Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet.* Anthropic, 2024.

[14] E. Härkönen, A. Hertzmann, J. Lehtinen, and S. Paris, "Ganspace: Discovering interpretable gan controls," *Advances in neural information processing systems*, vol. 33, pp. 9841–9850, 2020.

[15] C. Meng, Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon, "Sdedit: Guided image synthesis and editing with stochastic differential equations," *arXiv preprint arXiv:2108.01073*, 2021.

[16] D. Ha and J. Schmidhuber, "World models," *arXiv:1803.10122*, 2018.

[17] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *Proc. ICML*, 2019.

[18] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv:1912.01603*, 2019.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[20] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to generate reviews and discovering sentiment," *arXiv:1704.01444*, 2017.

[21] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proc. IEEE/CVF int. conf. on computer vision*, 2021.

[22] G. Alain and Y. Bengio, "Understanding intermediate layers using linear classifier probes," *arXiv preprint arXiv:1610.01644*, 2016.

[23] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas *et al.*, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)," in *Proc. ICML*, 2018.

[24] Y. Shen, C. Yang, X. Tang, and B. Zhou, "Interfacegan: Interpreting the disentangled face representation learned by gans," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 4, 2020.

[25] A. Hertz, R. Mokady, J. Tenenbaum, K. Aberman, Y. Pritch, and D. Cohen-Or, "Prompt-to-prompt image editing with cross attention control.(2022)," *URL https://arxiv. org/abs/2208.01626*, vol. 3, 2022.

[26] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks," *arXiv:2110.07602*, 2021.

[27] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu, "Plug and play language models: A simple approach to controlled text generation," 2020.

[28] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," 2023.

[29] M. Elad, *Sparse and redundant representations: from theory to applications in signal and image processing.* Springer, 2010.

[30] N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen *et al.*, "Toy models of superposition," *arXiv preprint arXiv:2209.10652*, 2022.

[31] H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey, "Sparse autoencoders find highly interpretable features in language models," *arXiv preprint arXiv:2309.08600*, 2023.

[32] A. Wilcox, A. Balakrishna, B. Thananjeyan, J. E. Gonzalez, and K. Goldberg, "Ls3: Latent space safe sets for long-horizon visuomotor control of sparse reward iterative tasks," in *Conf. on Rob. Learn.*, 2022.

[33] P. Koirala, Z. Jiang, S. Sarkar, and C. Fleming, "Latent safety-constrained policy approach for safe offline reinforcement learning," *arXiv preprint arXiv:2412.08794*, 2024.

[34] Y. Hogewind, T. D. Simao, T. Kachman, and N. Jansen, "Safe reinforcement learning from pixels using a stochastic latent representation," *arXiv preprint arXiv:2210.01801*, 2022.

[35] K. Nakamura, L. Peters, and A. Bajcsy, "Generalizing safety beyond collision-avoidance via latent-space reachability analysis," *arXiv:2502.00935*, 2025.

[36] B. Lindqvist, P. Sopasakis, and G. Nikolakopoulos, "A scalable distributed collision avoidance scheme for multi-agent uav systems," in *IEEE/RSJ int. conf. on int. robots and systems (IROS)*, 2021.

[37] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, "Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021.

[38] L. Wang, A. Ames, and M. Egerstedt, "Safety barrier certificates for heterogeneous multi-robot systems," in *Amer. cont. conf. (ACC)*, 2016.

[39] S. Zhang, O. So, K. Garg, and C. Fan, "Gcbf+: A neural graph control barrier function framework for distributed safe multi-agent control," *IEEE Transactions on Robotics*, 2025.

[40] Z. Huang, S. Batra, T. Chen, R. Krupani, T. Kumar, A. Molchanov, A. Petrenko, J. A. Preiss, Z. Yang, and G. S. Sukhatme, "Quadswarm:

A modular multi-quadrotor simulator for deep reinforcement learning with direct thrust control," *arXiv preprint arXiv:2306.09537*, 2023.

[41] D. Schmidt and M. Jiang, "Learning to act without actions," *arXiv preprint arXiv:2312.10812*, 2023.

[42] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun, "Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning," in *Proc. ICML*, 2020.

[43] T. Sainburg, L. McInnes, and T. Q. Gentner, "Parametric umap embeddings for representation and semisupervised learning," *Neural Computation*, vol. 33, no. 11, pp. 2881–2907, 2021.

[44] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *Proc. ICML*, 2021.