Final Report: R-PACE — Runtime Preference-Aware Chunk Editing for Frozen Zero-Shot Vision-Language-Action Execution

Revanth Senthilkumaran
Carnegie Mellon University
Email: revanths@andrew.cmu.edu

***Abstract*—Vision-language-action (VLA) policies have recently shown promising zero-shot manipulation capability on real robots, but task completion alone does not guarantee acceptable physical execution. In practice, a frozen zero-shot policy may successfully grasp and place an object while still violating a soft execution preference such as *place it gently*. This gap is directly relevant to embodied AI safety: in deployment, users care not only whether the robot finishes the task, but how it physically interacts with objects and the scene while doing so. In this project, I study runtime governance for this setting through R-PACE, a lightweight prefix-based chunk editing layer for frozen VLA execution. Rather than fully trusting each predicted action chunk, R-PACE selects how much of the chunk to execute before cutting the remainder and refreshing. I instantiate fixed-horizon, heuristic, and learned governors, and evaluate them on a real-robot green soda-can pick-and-place task with gentleness as the target preference. The results show that the base policy often completes the task but frequently remains nongentle, while naive short-horizon refreshing is overly conservative and harms completion. A learned live governor further shows that calibration materially changes runtime behavior and the task-success versus gentleness tradeoff, with lower intervention thresholds producing the first clear gentle-success signal on hardware. These results suggest that runtime chunk editing is a practical control point for improving preference-faithful execution without retraining the underlying VLA.**

I. Introduction & Motivation

Embodied AI systems are increasingly expected to execute language-conditioned tasks in the physical world, often using large vision-language-action (VLA) models that directly map observations and instructions to robot actions [3–5, 9]. These systems can display surprisingly strong zero-shot task completion, but embodied deployment raises a broader safety question: should we judge a robot only by whether it finishes the task, or also by how it physically does so?

This project focuses on a narrow but practically meaningful instance of that question. Consider an instruction such as: *pick up the green soda can and place it gently*. A frozen zero-shot VLA may grasp the right object and move it to the right place, yet still drop it from a noticeable height or press it into the table during release. From a purely task-level perspective, the robot succeeded. From a deployment and safety perspective, however, the execution was misaligned with the user's preference.

This problem sits squarely within the broader goals of embodied AI safety. In many real settings, the relevant failures are not only catastrophic collisions or total task failure. Instead, they include softer but still meaningful execution failures:

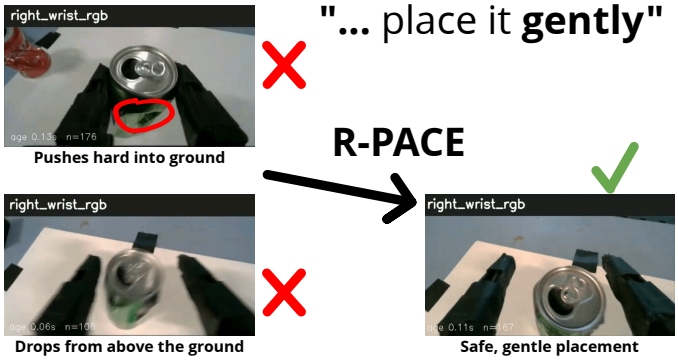

Fig. 1: **Teaser.** A frozen zero-shot VLA can complete the task while still violating a soft execution preference such as *place it gently*. Left and middle: representative base-policy failures with rough final placement. Right: R-PACE cuts the risky tail of the chunk, refreshes, and improves execution quality.

rough placement, unstable release, disturbing nearby objects, or mishandling objects in ways that degrade trust and usability. These failures are especially challenging because they do not necessarily change the high-level task goal; they change what counts as *acceptable execution*.

My project studies a lightweight runtime intervention for this problem. Modern VLAs frequently produce multi-step action chunks. This creates a natural question: once a frozen policy proposes an action chunk, how much of that chunk should actually be trusted and executed right now? The central hypothesis of this project is that many gentleness failures occur in the later part of a predicted chunk, especially during placement and release, so a runtime governor that edits the executed prefix may improve preference-faithful execution without retraining the base model.

This matters for at least three reasons. First, it offers a safety-relevant intervention point that does not require expensive retraining or fine-tuning of the VLA. Second, it is auditable: the governor can be understood as a separate control layer rather than a hidden change inside the base policy. Third, it forces a more realistic evaluation perspective, where task success is not treated as the sole measure of embodied competence.

At the same time, this is a deliberately narrow project. R-PACE is not intended to solve all semantic misalignment in

VLA execution. If the very first action in the chunk is already wrong, prefix editing can at best stop immediately; it cannot in general repair arbitrary semantic mistakes. The project instead targets a more specific and tractable class of failures: those where the early part of the chunk is acceptable, but the later part becomes stale, risky, or preference-violating.

II. Related Work

A. Vision-Language-Action Policies

Recent vision-language-action systems have demonstrated increasingly strong language-conditioned manipulation, including zero-shot or low-shot transfer to new tasks and embodiments [2–5, 9, 10]. This line of work motivates the setting of my project: I assume a strong frozen base policy that is already capable of meaningful real-robot pick-and-place behavior. My focus is therefore not on improving the policy itself through retraining, but on governing its execution at runtime.

B. Runtime Steering, Monitoring, and Failure Detection

A growing body of work studies runtime intervention for embodied systems, including uncertainty-aware fallback, failure detection, and test-time observation interventions [7, 11, 12]. These methods share the intuition that raw model outputs should not always be executed without scrutiny. However, they do not generally formulate the specific question I study here: given a chunk proposed by a frozen VLA, what is the largest prefix that should be executed now before refreshing?

C. Safety Filters and Embodied AI Safety

My project is also related to broader safety-filter ideas from robotics, where a nominal controller is monitored and possibly overridden when safety conditions are at risk [1, 6, 8]. The difference is that my target is not a hard certified constraint such as collision avoidance. Instead, it is a *soft execution preference*: gentleness during object placement. For that reason, I position R-PACE as a runtime governance layer rather than a formal safety guarantee.

D. Gap Addressed by This Project

The main gap addressed here is a lightweight, deployment-oriented one. Prior work shows that strong frozen VLAs exist, and prior work also explores runtime detection and intervention, but less attention has been paid to *prefix-level chunk editing* as a modular control point for preference-aware execution. This project isolates and evaluates that question on real hardware.

III. Problem Formulation

I consider a frozen zero-shot VLA policy deployed on a real robot. At time step t, the robot receives an observation o_t and language instruction x, and the frozen policy proposes an action chunk

$$A_t = (a_{t,1}, a_{t,2}, \dots, a_{t,H}),$$

where H is the chunk horizon.

The target task is a real-robot green soda-can pick-and-place instruction of the form:

"Pick up the green soda can and place it gently."

The key point is that task completion alone is not sufficient. A rollout may grasp and place the correct object, but still violate the intended preference during final execution. In this work, the focal preference is **gentleness during placement**. Operationally, gentleness is defined as avoiding visibly harsh or uncontrolled terminal placement behavior, such as a visible drop onto the surface, hard downward pressing at release, or abrupt contact near the end of placement. In the current setting, gentleness is evaluated primarily around the release / placement phase rather than across the entire episode. This reflects what was repeatedly observed on hardware: the main failures were concentrated in final placement quality rather than in the earlier transport phase.

The goal is to design a runtime governor G that takes the proposed chunk A_t and returns an executed prefix length k_t^*, where only

$$A_t^{1:k_t^*} = (a_{t,1}, a_{t,2}, \dots, a_{t,k_t^*})$$

is executed before the policy is refreshed on a new observation.

I model runtime governance as preference-aware prefix selection. Let $r_t(k)$ denote the risk of executing the first k actions of the current chunk with respect to the target preference. The problem is to choose a prefix length that maximizes trusted execution while keeping prefix risk below threshold:

$$k_t^* = \max\{k \in \{1, \dots, H\} : r_t(j) \le \tau \ \forall j \le k\}.$$

The intervention is therefore not over the whole rollout and not over a single low-level control step, but over the trusted prefix of the predicted chunk.

The project scope is intentionally narrow:

- the base VLA is frozen, with no policy retraining,
- intervention is purely runtime,
- the studied preference is gentleness during placement,
- the current setting is a single real-robot soda-can placement task,
- the governor is external to the base policy, *i.e.*, black-box with respect to the VLA internals,
- and no formal safety guarantees are claimed.

The central problem is therefore:

> *Can a lightweight runtime governor improve preference-faithful execution on frozen VLA chunks, especially when violations emerge in the later part of execution rather than immediately?*

This formulation also makes the project boundary clear. If the earliest action is already misaligned, chunk editing alone may not help except to stop or shorten execution. Thus, the intended mechanism is tail-risk mitigation, not arbitrary semantic correction.

IV. Proposed Approach

A. R-PACE Overview

R-PACE is a runtime governance layer that sits between the frozen VLA and the robot controller. The frozen policy

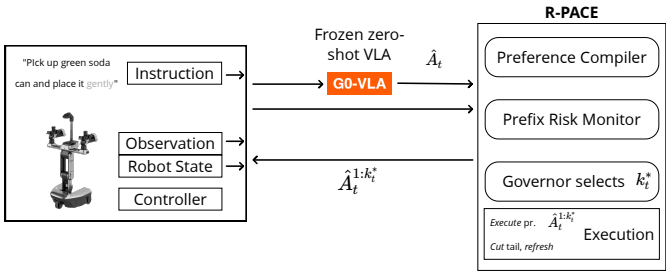

Fig. 2: **Runtime architecture.** Inputs are instruction, observation, and robot state. The frozen VLA proposes action chunk A_t. A preference compiler and prefix risk monitor estimate risk over candidate prefixes. The governor selects k_t^*. The robot executes $A_t^{1:k_t^*}$, cuts the tail, and refreshes.

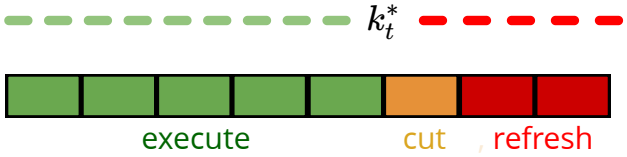

Fig. 3: **Prefix-based chunk editing.** R-PACE executes the trusted prefix, cuts the risky tail at k_t^*, and refreshes before the tail executes.

proposes an action chunk; R-PACE decides how much of that chunk to trust and execute.

Chunk-predicting VLAs create a distinctive runtime problem: later actions in the predicted chunk are committed under earlier observations and may become stale near contact or release. This makes execution horizon itself a meaningful runtime decision variable. This observation is empirical rather than universal: I do not claim that all VLA failures occur in the tail, only that in the current setting many of the gentleness failures did.

The overall runtime loop is

$$\begin{aligned}(x, o_t, s_t) &\xrightarrow{\text{frozen VLA}} A_t \\ &\xrightarrow{\text{preference compiler + prefix risk monitor}} k_t^* \\ &\xrightarrow{\text{execute } A_t^{1:k_t^*},\ \text{cut tail, refresh}} (o_{t+1}, s_{t+1}),\end{aligned} \tag{1}$$

where x is the language instruction, o_t is the observation, s_t is the robot state, and A_t is the proposed action chunk.

The important design choice is the intervention variable. Rather than steering observations, resampling many candidate plans, or verifying full imagined trajectories, R-PACE uses the current chunk itself as a candidate plan and decides how much of that plan should be executed before refresh. This is desirable when the base policy is already competent at the coarse task and the primary failure lies in the terminal execution details.

B. Preference Compilation

The instruction contains both a task goal and an execution preference. In this project, I focus on the specific preference *place it gently*. Rather than attempting to learn a full semantic grounding of gentleness from scratch, I compile this preference into a small set of runtime-monitored signals that are physically relevant to placement quality.

In the current prototype, these signals include end-effector vertical speed, end-effector vertical acceleration, and action-magnitude-based proxies near placement and release. These signals do not fully capture human judgment of gentleness, but they provide a practical first mechanism for runtime governance.

C. Heuristic Prefix Risk

For each proposed action chunk A_t, I score candidate prefixes according to their estimated risk with respect to the target preference. In the current soda-can setting, the heuristic governor uses threshold-normalized placement signals. Let

$$z_{t,k}^{\text{speed}}, \qquad z_{t,k}^{\text{accel}}$$

denote monitored vertical speed and vertical acceleration over prefix k, respectively. Using current gentleness thresholds

$$\bar{z}^{\text{speed}} = 0.15, \qquad \bar{z}^{\text{accel}} = 0.75,$$

the heuristic prefix risk is defined as

$$r_{t,k}^{\text{heur}} = \max\left(\frac{z_{t,k}^{\text{speed}}}{\bar{z}^{\text{speed}}}, \frac{z_{t,k}^{\text{accel}}}{\bar{z}^{\text{accel}}}\right).$$

More generally, if a compiled preference p activates a set of monitored signals $S(p)$, then

$$r_{t,k}^{\text{heur}} = \max_{m \in S(p)} s_m\left(z_{t,k}^{(m)}\right),$$

where each s_m is a threshold-normalized signal score. The heuristic cut point is then

$$k_t^* = \max\left\{k \in \{1, \dots, H\} : r_{t,j}^{\text{heur}} < \tau \ \forall j \le k\right\}.$$

D. Which Methods Are Actually Used

I evaluate four main runtime conditions on hardware:

- **Base policy**: execute the full chunk,
- **Fixed horizon**: execute a fixed prefix length,
- **Fixed short refresh**: execute a very short prefix before refresh,
- **Heuristic governor**: choose k_t^* using the risk rule above.

I then separately evaluate a **learned live governor** via a threshold sweep. This means the main comparison table is between base / fixed / heuristic execution strategies, while the learned governor appears in the later calibration study.

E. Learned Governor and Dense Prefix Supervision

The learned governor is intentionally lightweight. Rather than retraining the VLA, I train a small prefix-risk calibrator on logged chunk-prefix features and dense deviation-onset labels. The goal is to learn *when* the predicted chunk should stop being trusted, not to learn a new low-level policy.

The logging pipeline stores rollout metadata such as task identifier, condition identifier, episode outcome, and preference violation types. For a dense-labeled subset of episodes, I also annotate the onset of undesirable behavior at the prefix level. These dense annotations are derived from manual review of rollout videos and aligned to the chunk timeline so that the onset of harsh placement behavior can be localized to a prefix range rather than only to a whole-episode label.

Concretely, given a chunk-prefix feature vector $u_{t,k}$, the learned model predicts an unsafe-prefix probability

$$p_{t,k} = g_\theta(u_{t,k}),$$

where g_θ is a small supervised model. In the current implementation, the feature vector includes:

- focal preference signal values at prefix k,
- focal risk score and top overall risk at prefix k,
- prefix position features such as k, k/H, and remaining horizon,
- simple trend features such as differences between current and previous risk values,
- and lightweight metadata such as task or preference type.

Training labels are derived from dense deviation annotations: prefix k is labeled unsafe when the annotated first deviation occurs at or before k, and safe otherwise. The learned cut-point rule is

$$k_t^* = \max\{k \in \{1, \dots, H\} : p_{t,j} < \tau \ \forall j \le k\}.$$

For dense supervision, I manually annotate the onset of visibly bad terminal placement behavior in a subset of rollouts. For example, in one rollout with a successful grasp but bad final placement, the right-wrist RGB stream had 157 total frames and the visibly bad placement event occurred approximately over frames 102–108. This segment was marked with a dense note such as *drop on the ground happened in this frame*, and then mapped back to a prefix onset label for learned supervision.

This design keeps the base VLA frozen and focuses learning on a small, auditable calibration layer rather than end-to-end policy retraining. Refreshing after a cut does *not* guarantee that the next chunk will satisfy the preference; it simply re-queries the base policy under updated observations and creates a new opportunity for intervention.

V. Results

A. Experimental Setting

I evaluate on a real-robot tabletop pick-and-place task involving a green soda can. The robot is given a language instruction asking it to pick up the can and place it gently. The frozen zero-shot VLA is responsible for the nominal manipulation behavior, while R-PACE optionally edits the predicted action chunk at runtime.

The compared execution strategies are:

1) base frozen zero-shot policy,
2) fixed horizon,
3) fixed short refresh,
4) heuristic governor,
5) learned live governor with threshold sweep.

The first four conditions are the main real-robot comparison. The learned live governor is evaluated separately in the threshold ablation, because its role in this project is to study whether a lightweight learned calibrator can improve cut-point selection over simpler runtime rules.

To avoid reducing all behavior to one scalar success number, I evaluate each rollout with multiple labels:

- **Success**: the task is completed.
- **Recovered success**: the rollout initially deviates or partially fails but eventually completes the task.
- **Failure**: the task is not completed.
- **Gentleness violation**: the placement or release is harsher than desired.
- **Gentle success**: a rollout that succeeds while not violating gentleness.

I also report runtime-behavior statistics for the governed conditions, such as intervention rate, refresh count, and executed prefix ratio, to understand whether the governor actually changes execution behavior rather than merely changing final outcome counts.

B. Labeling Procedure

Rollouts are manually reviewed and annotated using both episode-level and dense prefix-level labels. Episode-level labels capture success, recovered success, failure, and whether a gentleness violation occurred. For the learned governor, a subset of episodes is additionally given dense temporal annotation identifying where undesirable behavior begins.

Concretely, I review rollout videos and identify the onset of visibly bad terminal placement behavior in the final placement chunk. The annotation is coarse but operationally useful: the goal is to locate the prefix region where the behavior starts becoming clearly ungentle, not to recover exact contact physics. For example, in one rollout with a successful grasp but bad final placement, the right-wrist RGB stream had 157 total frames and the visibly bad placement event occurred approximately over frames 102–108. This segment was marked with a dense note such as *drop on the ground happened in this frame*, and then mapped back to a prefix onset label for learned supervision.

These dense labels serve two purposes. First, they provide a more precise explanation of *when* the execution became problematic, which is important for a chunk-based intervention method. Second, they allow conversion of whole-rollout supervision into prefix-level labels suitable for learning g_θ. Invalid startup logs and malformed trials are excluded from the main analysis.

TABLE I: Main real-robot comparison across execution strategies.

Method	Succ./Rec.	Viol.	Gentle
Base policy	80%	80%	20%
Fixed horizon	58%	50%	17%
Fixed short refresh	17%	50%	0%
Heuristic governor	47%	53%	13%

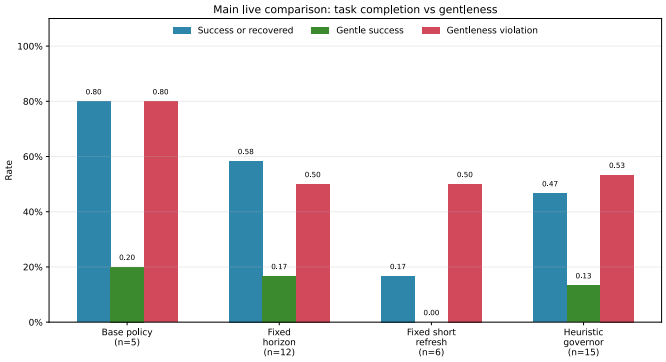

Fig. 4: **Main live comparison.** The base policy often succeeds but is frequently nongentle. Fixed short refresh is overly conservative and hurts completion, while fixed horizon and the heuristic governor occupy intermediate tradeoff regimes.

C. Main Real-Robot Comparison

The clearest pattern is that the base frozen zero-shot policy often completes the task, but frequently remains nongentle. In contrast, fixed short refresh is too aggressive: it reduces trust in the chunk so strongly that task completion drops substantially. Fixed horizon and heuristic governance occupy an intermediate regime.

These results support the core claim that task success alone is not sufficient, and that the intervention level matters. They do not show that simple refresh is always good; rather, they show that naive refresh can be too conservative.

D. Failure-Mode Breakdown

To better understand the behavior, I break rollouts into finer failure modes rather than only reporting final task success.

This matters because multiple methods can have similar top-level success while differing substantially in whether the object was dropped, placed harshly, or mishandled at the final interaction.

E. Learned Governor Calibration

The learned live governor provides the clearest evidence that calibration changes real-robot behavior. I sweep the intervention threshold τ and observe a meaningful tradeoff between aggressiveness and gentleness. Higher thresholds preserve more of the chunk and therefore intervene less; lower thresholds intervene earlier and shorten the executed prefix.

In the current experiments:

- $\tau = 0.40$: $4/5$ success, $4/5$ gentleness violations,

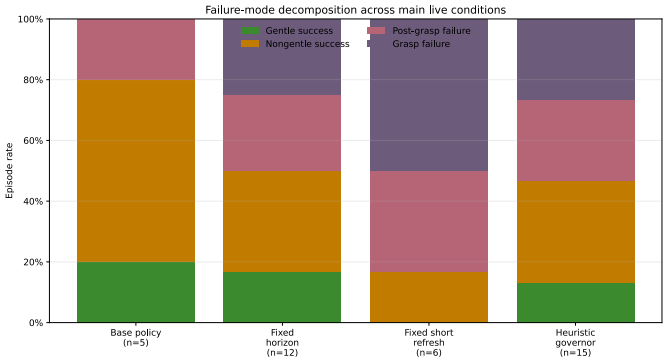

Fig. 5: **Failure-mode breakdown.** Reporting only final success obscures important differences in behavior. The base policy often succeeds but is frequently nongentle, while governed conditions shift the distribution of gentle success, nongentle success, and failure types.

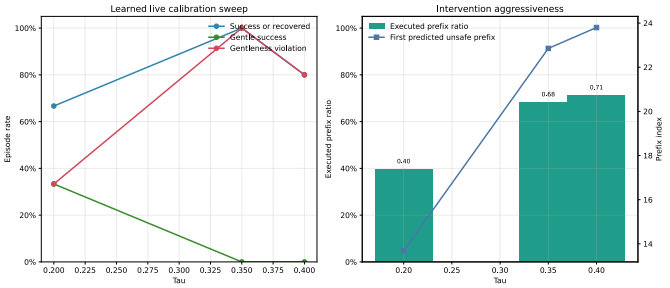

Fig. 6: **Learned live governor threshold sweep.** Varying τ changes the intervention regime. Lower thresholds intervene earlier and produce the first clear gentle-success signal in the current learned-governor setting.

- $\tau = 0.35$: $3/3$ success, $3/3$ gentleness violations,
- $\tau = 0.20$: $2/3$ success, $1/3$ gentleness violations.

The most important point is that $\tau = 0.20$ is the first learned-live setting in this study with a clear gentleness improvement signal. It does not dominate all other metrics, but it demonstrates that a learned runtime governor can materially change execution behavior on hardware.

F. Runtime Behavior Analysis

The outcome statistics above are more convincing when paired with direct evidence that the governor changes execution. For this reason, I also analyze runtime behavior metrics such as intervention rate, refresh count, and executed prefix ratio. Lower thresholds reduce the mean executed prefix ratio, indicating that the robot is trusting a smaller portion of the predicted chunk before refreshing.

This supports the intended mechanism: the learned governor is not merely correlated with different outcomes; it changes the amount of chunk executed online. In this sense, calibration affects both *how* the robot executes and *what* outcomes result.

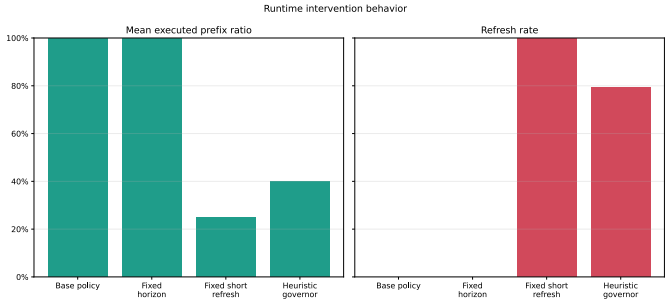

Fig. 7: **Runtime behavior under governance.** As the governor becomes more conservative, the robot executes a smaller fraction of the predicted chunk before refreshing. This supports the intended mechanism behind the observed outcome shifts.

VI. Conclusion

This project studied a narrow but important embodied-AI-safety problem: a frozen zero-shot VLA can complete a task while still violating a soft execution preference such as gentleness. I proposed R-PACE, a runtime preference-aware chunk editing layer that treats the predicted action chunk as a candidate plan and selects how much of that chunk to execute before refreshing.

The main lessons are:

- task completion alone is not a sufficient deployment metric,
- prefix-based chunk editing is a practical runtime intervention point for frozen VLA execution,
- and learned calibration changes the real-robot completion-versus-gentleness tradeoff.

More broadly, this project suggests that embodied AI safety should include not only catastrophic-failure prevention, but also runtime governance for softer execution preferences that matter in real deployment.

A. Future Work

The current work is intentionally narrow in scope. The experiments cover a single real-robot task, one object class, and one soft preference. The gentleness monitor still relies on proxy signals rather than semantic visual or tactile judgment of release quality. The learned governor is trained from limited dense supervision, and the dataset size is modest. As a result, this should be read as evidence for a promising runtime intervention point, not as a complete solution to general preference alignment for robot execution.

Future work should extend the method to broader side constraints, such as not disturbing nearby objects, keeping objects upright, or respecting clearance around protected objects. Another important direction is richer semantic monitoring, including visual or multimodal learned critics that better capture the meaning of soft execution preferences. It would also be valuable to compare more directly against explicit sample-then-verify steering systems on the same real-robot tasks, in order to better characterize when lightweight chunk editing is preferable to plan-verification pipelines.

B. Contributions of Each Team Member

This project was completed individually. I was responsible for the problem formulation, implementation of the runtime governance and learned calibrator pipeline, data collection and annotation design, experimental analysis, figure generation, video production, presentation, and report writing.

References

[1] Aaron D. Ames et al. Control barrier functions: Theory and applications. In *European Control Conference*, 2019.

[2] Anthony Brohan, Noah Brown, Justice Carbajal, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022.

[3] Anthony Brohan, Noah Brown, Justice Carbajal, et al. Rt-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems*, 2023.

[4] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, 2023.

[5] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, et al. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning*, 2023.

[6] Jaime F. Fisac et al. General safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 2019.

[7] Asher J. Hancock, Allen Z. Ren, and Anirudha Majumdar. Run-time observation interventions make vision-language-action models more visually robust. *arXiv preprint arXiv:2410.01971*, 2024.

[8] Kai-Chieh Hsu, Heng Hu, and Jaime F. Fisac. The safety filter: A unified view of safety-critical control in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 2024.

[9] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag 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. *arXiv preprint arXiv:2406.09246*, 2024.

[10] Jacky Liang et al. Open x-embodiment: Robotic learning datasets and rt-x models. In *IEEE International Conference on Robotics and Automation*, 2024.

[11] Allen Z. Ren et al. Robots that ask for help: Uncertainty alignment for large language model planners. In *Conference on Robot Learning*, 2023.

[12] Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. *arXiv preprint arXiv:2503.08558*, 2025.