PROGRESS IS THE REWARD: UAV POLICY OPTIMIZATION VIA PROGRESS PREDICTION

Anonymous authors
Paper under double-blind review

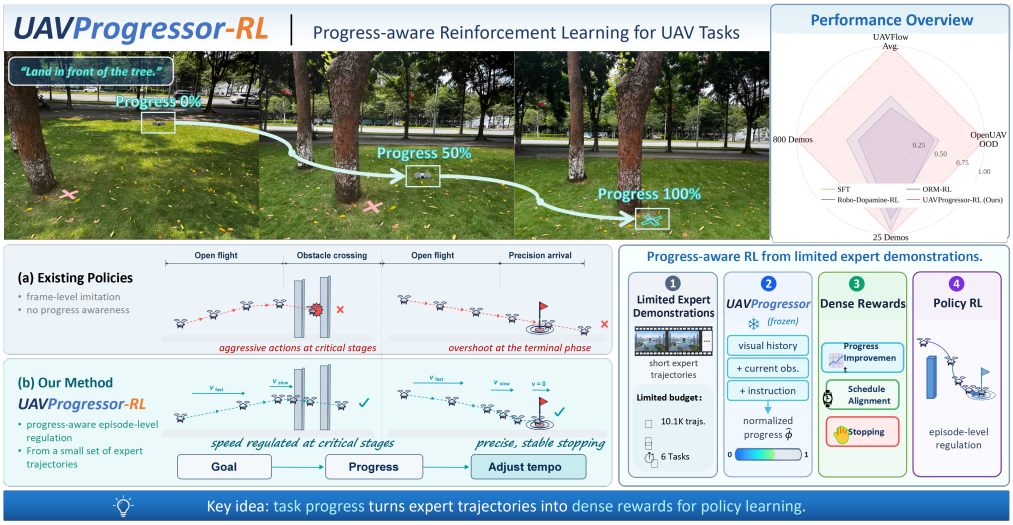


Figure 1: **UAVProgressor-RL learns task progress from a small set of expert trajectories and uses it as a dense reward for episode-level regulation.** (a) Frame-level imitation fails at critical stages such as obstacle approach and landing, whereas (b) progress-aware regulation keeps execution on schedule and stops precisely and stably under limited demonstrations.

ABSTRACT

High-quality UAV control policies rely on expert demonstrations that are costly to collect and limited in scale. More importantly, supervised imitation provides only frame-level action supervision, leaving episode-level progress regulation unlearned and making long-horizon policies prone to inappropriate pacing and unstable stopping. Yet the temporal structure required for such regulation is already embedded in complete expert trajectories. We propose **UAVProgressor-RL**, a progress-aware reinforcement learning framework that learns task progress from limited expert trajectories and converts it into dense feedback for policy optimization. The history-conditioned progress model **UAVProgressor** estimates normalized task progress from visual history, the current observation, and the task instruction, and its predictions are used to construct rewards for progress improvement, schedule alignment, and completion-aware stopping. We further introduce Schedule-Aligned Displacement Error (SADE) to evaluate whether spatial execution remains aligned with task progress. Extensive experiments on UAVFlow and OpenUAV show that UAVProgressor-RL achieves the best overall performance across five policy backbones and retains its gains on unseen maps, successfully turning limited demonstrations into effective dense supervision. Our models and training code will be publicly available.

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

1 INTRODUCTION

Vision-Language-Action (VLA) models and World Action Models (WAMs) are increasingly expected to combine fine-grained control with episode-level planning. For UAV tasks, due to the effects of vehicle dynamics, predicting geometrically correct actions no longer guarantees success: like humans adapting behavior to task progress, the policy must not only specify *where* to move but also sense **task progress** to regulate *how fast* to advance and when to stop. Overly aggressive actions near critical stages, such as obstacle approach and landing, induce overshoot or collision.

Existing paradigms hardly provide this awareness. Supervised fine-tuning (SFT) imitates frame-wise actions without evaluating the overall rate of task advancement (Chen et al., 2025a): since trajectories are split into largely independent local samples, the temporal structure encoding pacing is discarded, leaving pacing errors invisible to the per-step loss. Reinforcement learning (RL) can optimize behavior at the episode level, yet recent VLA-RL methods obtain signals from learned critics, process reward models, or large-scale rollouts (Li et al., 2026; Lu et al., 2025; Shu et al., 2025), all of which need large amounts of data that UAV settings rarely have. This raises our central question: **Is it possible to acquire progress-aware UAV policy with limited expert trajectories?**

We answer with **UAVProgressor-RL**, built on one key insight: the temporal structure discarded by SFT lies implicitly within expert trajectories, since the normalized position of each frame within its trajectory already labels how far the task has advanced. These free labels allow us to train a progress model, **UAVProgressor**, on those demonstrations without additional annotation. Built on a pretrained video world model, UAVProgressor predicts normalized progress from the visual history, current observation, and instruction. History disambiguates visually similar states at different stages, while withholding the timestep forces predictions to follow visual evidence rather than elapsed time.

Built around this progress model, UAVProgressor-RL trains the policy in two stages. The first stage trains the policy and UAVProgressor on the same expert trajectories with action and progress supervision respectively. The second stage freezes UAVProgressor and turns its predictions into a dense PPO reward for episode-level regulation. Progress improvement credits actions that advance the task, schedule alignment penalizes falling behind a reference schedule since slow yet correct actions still gain progress, and completion-aware stopping suppresses residual motion once progress signals completion. Since the two models interact only through this reward, the framework is architecture-agnostic, as verified on five VLA and world-action policies.

To quantify schedule-aligned execution, we propose **Schedule-Aligned Displacement Error (SADE)**, which evaluates whether spatial execution remains synchronized with the reference task progress. Experiments on UAVFlow (Wang et al., 2025a) show that UAVProgressor-RL achieves the best overall performance among the compared methods across five policy backbones, outperforming supervised imitation and existing reward-design baselines. Consistent improvements are further observed on the unseen-map subset of OpenUAV (Wang et al., 2025b). Our main contributions are:

1. We propose UAVProgressor-RL, a progress-aware reinforcement learning framework for UAV tasks. Its two-stage pipeline decouples progress modeling from policy learning, enabling the policy to learn frame-level actions and episode-level pacing.
2. We propose UAVProgressor, a history-conditioned task progress model, and construct from its predictions a dense progress-aware reward. Progress improvement, schedule alignment, and completion-aware stopping supervise advancement, pacing, and terminal stopping.
3. We validate UAVProgressor-RL on UAVFlow through progress prediction, online policy optimization, and learning under limited expert demonstrations. We further introduce a Schedule-Aligned Displacement Error (SADE) metric for progress-aware evaluation

2 RELATED WORK

2.1 PROCESS MODELING IN VLA TASKS

In long-horizon VLA tasks, visually similar states may correspond to different stages. Memory-augmented models make decisions history-dependent (Shi et al., 2026b; Zhou et al., 2026; Yang et al., 2026b) or preserve long-term context through keyframes, memory tokens, events, or future

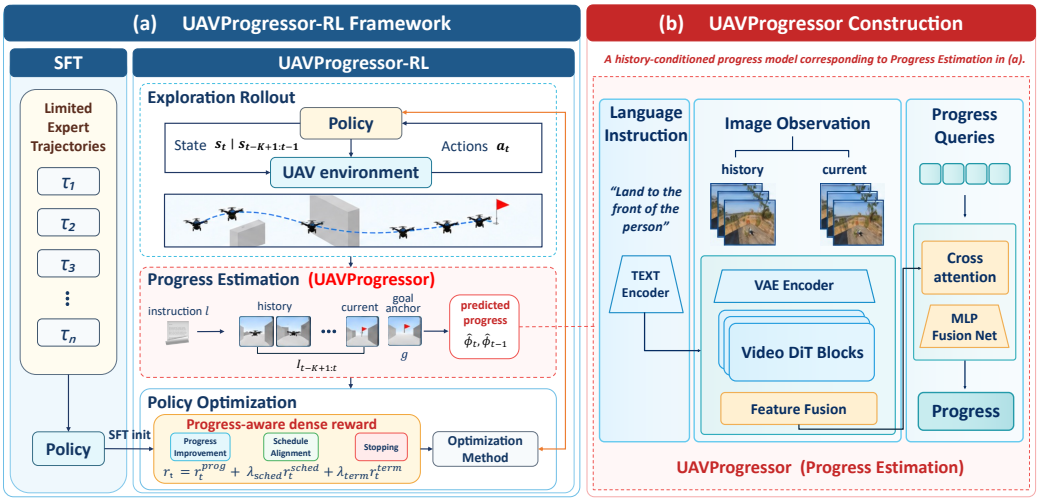


Figure 2: Overview of UAVProgressor-RL. The framework first learns a history-conditioned progress model, UAVProgressor, from expert trajectories. During reinforcement learning, UAVProgressor is frozen and converts predicted execution progress into dense process-level rewards for optimizing the UAV policy.

prediction (Sridhar et al., 2026; Wang et al., 2026; Yang et al., 2026a; Shi et al., 2026a), yet none distills this history into where the task stands. Others build explicit task-level representations: execution stages (Fan et al., 2025; Hung et al., 2025; Chen et al., 2026), values or temporal distances (Shu et al., 2025; Xia et al., 2026; Huang et al., 2026), goal-proximity representations (Ma et al., 2023b;a; Zakka et al., 2022), and step- or completion-level reward models (Liang et al., 2026; Tan et al., 2026; Lee et al., 2026). These convey quality, distance, or stage, but not the position on the reference execution process, which in visually repetitive tasks only history can resolve. UAVProgressor provides this variable: a continuous, history-conditioned estimate of task progress.

2.2 Reinforcement Learning with Progress Signals for VLA

Offline RL (Kumar et al., 2020; Fujimoto & Gu, 2021; Kostrikov et al., 2022; Nair et al., 2020; Ball et al., 2023; Chebotar et al., 2023; Luo et al., 2025) anchors VLA post-training: ConRFT and CO-RFT combine behavior cloning with value-based or offline RL (Chen et al., 2025a; Huang et al., 2025), Robo-ValueRL uses history-conditioned value estimation (Xia et al., 2026); online, VLA-RL trains a process reward model and critic (Lu et al., 2025), SimpleVLA-RL scales outcome-feedback RL (Li et al., 2026), TGRPO and IG-RFT mix trajectory-, step-, and subtask-level signals (Chen et al., 2025b; Su et al., 2026), WMPO and WorldGym evaluate in learned world models (Zhu et al., 2026; Quevedo et al., 2026).

Closest to ours, RFTF derives dense rewards from value differences (Shu et al., 2025), RynnValue from temporal distances (Huang et al., 2026), SARM from stage-wise progress (Chen et al., 2026), and MVR, RoboMeter, and Robo-Dopamine from visual, comparative, or learned rewards (Luo et al., 2026; Liang et al., 2026; Tan et al., 2026). None provides the progress awareness episode-level regulation requires: they never align execution with a schedule or supervise stopping, and online variants need critics or large-scale rollouts. UAVProgressor-RL instead derives task progress from limited expert trajectories and converts it, via a frozen evaluator, into rewards of progress improvement, schedule alignment, and completion-aware stopping.

3 Method

As illustrated in Fig. 2, UAVProgressor-RL adopts a two-stage training framework. First, the UAV policy is trained with action supervision, while UAVProgressor learns task progress from complete expert trajectories. Then, UAVProgressor is frozen and its predicted progress is converted into dense rewards for reinforcement-learning-based policy optimization.

3.1 PROGRESS MODELING

Given a task instruction l and an expert trajectory of length T, we assign each frame a normalized progress target $\phi_t = t/(T-1) \in [0,1]$, where $\phi_0 = 0$ and $\phi_{T-1} = 1$ denote the beginning and completion of the reference execution, respectively. This provides dense supervision across trajectories of different durations without additional annotation, following prior work on progress- or value-based learning from ordered demonstrations (Zhang et al., 2025; Chen et al., 2026; Liang et al., 2026; Huang et al., 2026). The temporal index is used only to construct training labels; at inference time, UAVProgressor receives neither t, T, nor future observations.

Since a single observation may be ambiguous, progress is estimated from a visual prefix as $\hat{\phi}_t = f_\psi(I_{t-K+1:t}, l)$, where $I_{t-K+1:t}$ contains the most recent K observations. The **progress predicting model** uses Wan2.2 (Team Wan et al., 2025) as its world-model backbone. Historical and current frames are encoded with frame-role and relative temporal embeddings, and intermediate-layer features are fused before being aggregated by a lightweight progress head with learnable queries. The resulting prediction can be written as $\hat{\phi}_t = h_{\text{prog}}(F_{\text{WM}}(I_{t-K+1:t}), l)$, where F_{WM} denotes the visual-temporal representation produced by the Wan2.2 world model.

UAVProgressor is trained with trajectory-derived progress labels using $\mathcal{L}_{\text{prog}} = \frac{1}{N}\sum_{i=1}^{N} \ell(\hat{\phi}_i, \phi_i)$, together with auxiliary progress prediction and adjacent-progress consistency objectives. After training, the model is frozen and used as the progress evaluator during policy optimization.

3.2 PROGRESS-GUIDED REINFORCEMENT LEARNING

In the second stage, the policy is initialized from its SFT checkpoint while UAVProgressor remains frozen. During online rollouts, predicted progress is converted into three complementary reward signals: progress improvement, schedule alignment, and completion-aware stopping.

For dense intermediate feedback, we define the progress change as $\Delta\hat{\phi}_t = \hat{\phi}_t - \hat{\phi}_{t-1}$ and construct $r_t^{\text{prog}} = \text{clip}(\Delta\hat{\phi}_t, -r_{\max}, r_{\max})$, following dense reward shaping and progress-based robotic reward formulations (Ng et al., 1999; Tan et al., 2026; Huang et al., 2026). Positive progress therefore produces positive reward, while stagnation or regression is penalized.

Progress improvement alone does not constrain execution pace. We therefore introduce an episode-specific reference schedule

$$c_t = \text{clip}\left(\frac{t}{T_{\text{ref}}}, 0, 1\right), \tag{1}$$

where T_{ref} denotes the duration of the corresponding normal expert execution. Since reference trajectories can have different durations, c_t provides a normalized reference coordinate for the expected execution pace of each episode rather than a global timestep prior. In contrast, the actual execution progress is estimated from visual evidence,

$$\hat{\phi}t = f\psi\left(I_{t-K+1}, l\right), \tag{2}$$

without access to the absolute timestep t, the reference duration T_{ref}, or future observations. We therefore use the discrepancy between expected and visually inferred progress to identify delayed execution:

$$r_t^{\text{sched}} = -\max\left(c_t - \hat{\phi}t - \epsilon\text{sched}, 0\right). \tag{3}$$

This term acts as an asymmetric lag penalty: it penalizes substantial delay behind the reference pace, but does not directly reward matching a particular reference spatial position or penalize execution merely for being ahead of the schedule. It therefore regulates execution pace rather than directly optimizing trajectory displacement.

To stabilize task termination, a terminal reward r_t^{term} is activated when $\hat{\phi}_t \geq \phi_{\text{term}}$. In this completion phase, the reward encourages reduced residual motion and stable stopping once the task completion condition is satisfied, tying termination behavior to predicted task progress rather than an isolated episode-end signal.

The final reward is $r_t = r_t^{\text{prog}} + \lambda_{\text{sched}} r_t^{\text{sched}} + \lambda_{\text{term}} r_t^{\text{term}}$, where the three terms respectively supervise task advancement, execution pace, and stable completion.

Table 1: Policy comparison across UAVFlow backbones and OpenUAV OOD using SADE (↓).

Method	OpenVLA	StarVLA QwenPI	StarVLA QwenGR00T	π_0	FastWAM	OpenUAV OOD FastWAM
			Long-horizon			
w/o RL	0.6241	0.8932	0.8245	0.7546	0.6591	–
RoboMeter	0.4906	0.8527	0.6589	0.6125	0.6048	–
Robo-Dopamine	0.4576	0.7938	0.7287	0.5819	0.5426	–
ORM	0.4705	0.8127	0.6729	0.5908	0.4279	–
UAVProgressor-RL (Ours)	**0.3331**	**0.7069**	**0.6716**	**0.5964**	**0.1573**	–
			Short-horizon			
w/o RL	0.4217	0.8086	0.7381	0.6698	0.5703	0.9436
RoboMeter	0.3998	0.7721	0.5767	0.5313	0.5144	0.8364
Robo-Dopamine	0.3718	0.5756	0.5639	0.4873	0.3532	0.7218
ORM	0.3873	0.5891	0.5905	0.5116	0.3445	0.7456
UAVProgressor-RL (Ours)	**0.2705**	**0.5355**	**0.4908**	**0.4456**	**0.0987**	**0.5127**
			Overall			
w/o RL	0.5229	0.8509	0.7813	0.7122	0.6147	0.9436
RoboMeter	0.4452	0.8124	0.6178	0.5719	0.5596	0.8364
Robo-Dopamine	0.4147	0.6847	0.6463	0.5346	0.4479	0.7218
ORM	0.4289	0.7009	0.6317	0.5512	0.3862	0.7456
UAVProgressor-RL (Ours)	**0.3018**	**0.6212**	**0.5812**	**0.5210**	**0.1280**	**0.5127**
Δ *vs. Best Baseline*	**0.1129**	**0.0635**	**0.0366**	**0.0136**	**0.2582**	**0.2091**

We optimize the UAV policy using PPO (Schulman et al., 2017). UAVProgressor remains frozen throughout RL and receives no policy gradient, avoiding reward drift as the policy changes. The progress model and policy share no parameters and interact only through scalar rewards. We instantiate the policy with FastWAM in the main experiments, while the same reward formulation is applicable to other VLA or world-action policies that support continuous environment rollouts.

4 Experiments

We evaluate UAVProgressor-RL on UAVFlow for policy optimization, progress prediction, and data efficiency, with OpenUAV used for cross-domain evaluation. We study policy improvement across backbones, progress-estimation quality, and robustness to limited demonstrations and distribution shift. Full dataset and execution details are provided in the Appendix.

4.1 Benchmarks and Evaluation Settings

Benchmarks and metrics. UAVFlow (Wang et al., 2025a) is the primary benchmark. Policy performance is measured by **Schedule-Aligned Displacement Error (SADE)** on long- and short-horizon tasks, with their equally weighted mean reported as Overall. SADE measures spatial synchronization with reference progress and is sensitive to delay, repeated motion, and post-completion overshoot; its definition is given in Appendix C.3. Progress prediction uses MAE, PCC, and SRCC.

UAVFlow evaluation covers six representative behaviors. SurroundApproach, SurroundCircle, and SurroundLeave form the long-horizon group and require multi-stage execution, while Approach, FollowStraight, and FollowTurn form the short-horizon group and emphasize target approach and tracking. Progress prediction is evaluated on 273 held-out trajectories that are separated from the training pool at the trajectory level.

For cross-domain evaluation, we test the FastWAM-based policy on a length-matched subset of the OpenUAV (Wang et al., 2025b) Test Unseen Map split without OpenUAV training data. The subset contains 48 short-horizon trajectories whose duration distribution is matched to the UAVFlow

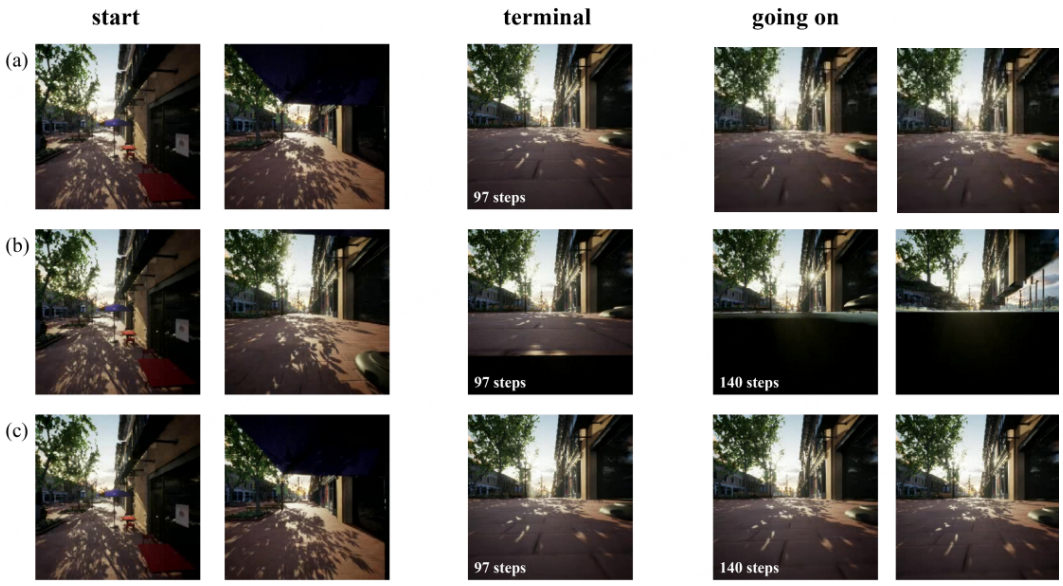


Figure 3: **Qualitative comparison of terminal stopping.** The image sequences compare SFT and UAVProgressor-RL when both policies reach the target region at approximately 97 steps and after execution continues to 140 steps. SFT keeps moving after reaching the target and eventually leaves the valid map, whereas UAVProgressor-RL remains near the terminal pose with only small local adjustments.

short-horizon evaluation set, reducing the influence of episode length when measuring cross-domain transfer. Split construction is detailed in Appendix B.

4.2 Compared Methods

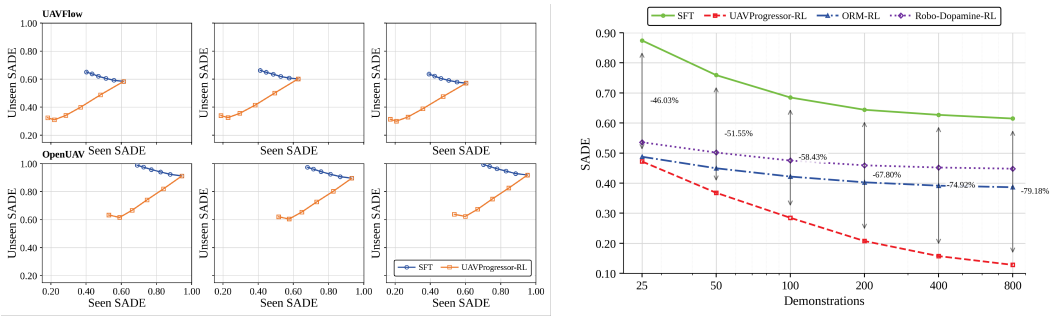
Policy and reward baselines. We evaluate UAVProgressor-RL with OpenVLA (Kim et al., 2024), StarVLA-QwenPI (StarVLA Community, 2026; Black et al., 2025), StarVLA-QwenGR00T (StarVLA Community, 2026; NVIDIA et al., 2025), π_0 (Black et al., 2025), and FastWAM (Yuan et al., 2026). Each SFT checkpoint is the no-RL baseline, and RoboMeter (Liang et al., 2026), Robo-Dopamine (Tan et al., 2026), and an outcome reward model (ORM) are compared under matched initialization and interaction conditions.

Progress prediction baselines. UAVProgressor is compared with RoboMeter (Liang et al., 2026), Robo-Dopamine (Tan et al., 2026), StarVLA-Progress (StarVLA Community, 2026), FastWAM-Progress (Yuan et al., 2026), and V-JEPA-Progress (Assran et al., 2025). All predictors use matched progress supervision and evaluation data, and each is used to construct rewards for the same downstream policy.

4.3 Training and Analysis Protocols

Policy optimization. All RL policies start from their SFT checkpoints, with UAVProgressor frozen. PPO (Schulman et al., 2017) is used by default; PPO and GRPO (Shao et al., 2024) are additionally compared under matched settings in Appendix D.1. During closed-loop execution, policies operate at 5 Hz, predict eight future actions at each query, and execute the first two before replanning. The same action limits, execution interface, task splits, and online interaction budget are used for all compared RL methods. Full training details are in Appendix A.

Data efficiency and generalization. We vary the UAVFlow demonstration budget over $\{25, 50, 100, 200, 400, 800\}$ trajectories using nested subsets, compare simulator-expert data with Expert-Inspired Trajectory Synthesis (EITS), and track SFT and UAVProgressor-RL across checkpoints on UAVFlow and OpenUAV. At each demonstration budget, competing methods use the same



(a) Generalization (b) Data scarcity

Figure 4: Generalization and data-scarcity analysis. (a) Mean performance over three selected task instances on UAVFlow and OpenUAV across six checkpoints, pairing seen-task and held-out performance. (b) Policy performance under different expert-demonstration budgets on UAVFlow. Lower SADE is better.

offline data and matched online interaction budget, isolating the effect of progress-based reward construction. Protocols are detailed in Appendix B.

5 Results

5.1 Policy Performance on UAVFlow and OpenUAV

UAVFlow performance across policy architectures. Table 1 shows lower *overall* SADE with UAVProgressor-RL across all five evaluated architectures. Relative to the strongest reward baseline, the reductions are 0.1129, 0.0635, 0.0366, 0.0136, and 0.2582 for OpenVLA, StarVLA-QwenPI, StarVLA-QwenGR00T, π_0, and FastWAM, respectively. This supports transferability at the overall level, although the advantage is not uniform across every task group: on long-horizon tasks, StarVLA-QwenGR00T and π_0 do not outperform their strongest reward baselines.

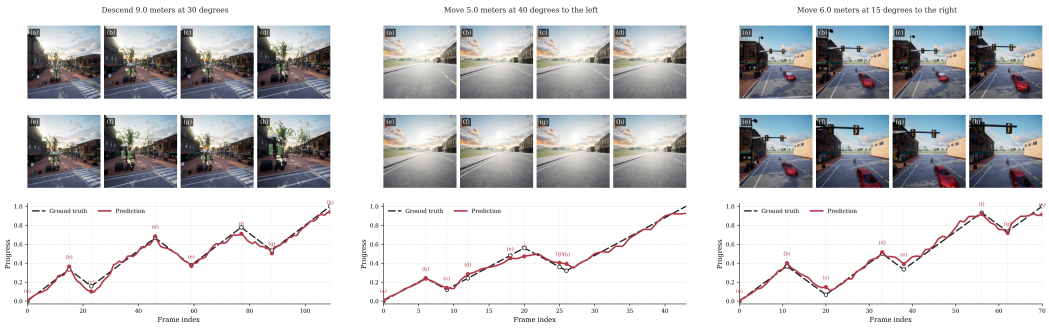
With FastWAM, overall SADE decreases from 0.6147 for SFT and 0.3862 for the strongest reward baseline to **0.1280**; long- and short-horizon SADE are **0.1573** and **0.0987**. We repeat each evaluation three times and observe the same performance ordering across runs.

The magnitude of the improvement varies across backbones. In particular, FastWAM exhibits the largest reduction over the strongest reward baseline, whereas the gains for StarVLA-QwenGR00T and π_0 are smaller at the overall level. This variation indicates that the progress reward does not simply produce a uniform offset to policy performance. Instead, its effect depends on the underlying policy initialization and the quality of the behavior already captured by supervised training. Nevertheless, all five backbones obtain lower overall SADE after progress-aware optimization.

Cross-domain transfer to OpenUAV. On the OpenUAV unseen-map subset, UAVProgressor-RL reaches **0.5127** SADE, versus 0.9436 for SFT and 0.7218 for the strongest reward baseline. Because no OpenUAV trajectory is used for training, this result evaluates zero-shot transfer across scene appearance and map layout.

Qualitative case study on terminal stopping. Figure 3 illustrates the effect of progress-aware optimization near task completion. Both SFT and UAVProgressor-RL reach the target region at approximately 97 execution steps, but their behaviors diverge afterward. SFT continues producing substantial motion and eventually leaves the valid map, whereas UAVProgressor-RL remains close to the terminal pose up to 140 steps with only small local adjustments.

This behavior reflects the role of completion-aware reward shaping. Reaching the target is insufficient if subsequent actions immediately move the UAV away from the completed state. By coupling terminal behavior with predicted progress, UAVProgressor-RL learns to suppress unnecessary post-completion motion, which also reduces schedule-aligned displacement error.



(a) Backward motion (b) Repeated observations (c) Variable-speed execution

Figure 5: **Robustness of UAVProgressor to temporal perturbations.** We perturb the execution process while keeping the task instruction unchanged. Reference progress is inherited from the corresponding states of the original trajectories rather than recomputed from the perturbed frame index. UAVProgressor captures local stagnation and regression and recovers when task advancement resumes.

Complementary nDTW evaluation. We additionally report nDTW (Ilharco et al., 2019) as a conventional trajectory-similarity metric. UAVProgressor-RL improves average nDTW from 0.3409 for SFT to **0.3712**, slightly above ORM at 0.3675. The smaller margin is expected because the baselines already use heuristic termination rules, including small-motion stopping and maximum-step termination, while dynamic temporal alignment makes nDTW less sensitive to pacing and exact stopping time. The consistent improvement nevertheless shows that the gain is not specific to SADE.

Generalization as seen-task performance improves. Figure 4(a) examines whether continued optimization on seen tasks sacrifices performance on held-out environments. For SFT, held-out SADE increases to 0.6503 on UAVFlow and 0.9860 on OpenUAV as seen-task fitting improves. UAVProgressor-RL instead reaches 0.3273 and 0.6317, respectively, and exhibits only a small deterioration at the final checkpoint.

The different trends suggest that progress-guided optimization is less prone to over-specializing to the seen trajectories. Rather than only reinforcing individual action targets, the reward constrains whether the execution continues to advance through the expected task process, providing a more task-level optimization signal.

5.2 Learning with Limited Expert Demonstrations

Figure 4(b) varies the expert-demonstration budget. With 25 demonstrations, UAVProgressor-RL obtains 0.4691 SADE, close to ORM-RL at 0.4728 and below SFT at 0.8726. It improves to 0.2816 with 100 demonstrations and **0.1280** with 800, versus 0.3862 for ORM-RL and 0.4479 for Robo-Dopamine-RL. The gain over SFT is therefore already substantial in the low-data regime, while the margin over ORM-RL increases as additional demonstrations improve progress-based reward construction.

5.3 Progress Prediction and Reward Quality

Table 2 compares progress estimation and downstream reward quality. UAVProgressor achieves **0.1003** MAE, **0.9808** PCC, **0.9957** SRCC, and **0.1280** downstream SADE. Low MAE provides accurate calibration for schedule alignment, while high correlation preserves the ordering of advancement and regression. Both properties are important because progress errors are directly converted into online reward noise.

Robustness to temporal perturbations. To test whether progress is inferred from visual state rather than elapsed time, we perturb trajectories with backward motion, repeated observations, and variable-speed execution. Each observation retains the progress label of its corresponding state in the original trajectory, so reference progress may stall or decrease even as time advances. Figure 5 shows that $\hat{\phi}_t$ decreases under regression, remains nearly unchanged under repetition, and rises when

Table 2: Progress-estimator comparison on UAVFlow.

Model (Size)	MAE ↓	PCC ↑	SRCC ↑	RL SADE ↓
RoboMeter (4B)	0.2772	0.4735	0.4830	0.5596
Robo-Dopamine (8B)	0.2886	0.4729	0.6417	0.4479
StarVLA-Progress (4B)	0.4728	0.3866	0.3728	0.6468
FastWAM-Progress (5B)	0.4615	0.3982	0.3846	0.6327
V-JEPA-Progress (1B)	0.2827	0.1889	0.1072	0.6038
UAVProgressor (5B)	**0.1003**	**0.9808**	**0.9957**	**0.1280**

Table 3: Demonstration-source ablation on UAVFlow.

Source	Method	SADE ↓
EITS	SFT	0.6813
EITS	UAVProgressor-RL	0.2089
UAVFlow Expert	SFT	0.6147
UAVFlow Expert	**UAVProgressor-RL**	**0.1280**

EITS: Expert-Inspired Trajectory Synthesis.

advancement resumes. This behavior differs from the schedule c_t, which necessarily increases with elapsed time, confirming that expected and visually observed progress are distinct signals on the same normalized coordinate.

5.4 Ablation Analysis

Progress-modeling configuration. The progress-model comparison shows that progress supervision alone is not sufficient to produce an effective reward evaluator. Although all evaluated models estimate task progress, their downstream effects differ substantially. StarVLA-Progress and FastWAM-Progress obtain SADE values of 0.6468 and 0.6327, respectively, both worse than the shared SFT initialization of 0.6147, while V-JEPA-Progress provides only a small improvement to 0.6038. In contrast, UAVProgressor reaches **0.1280**.

This result shows that progress-guided RL is sensitive to evaluator quality. Poorly calibrated or weakly ordered progress predictions can introduce reward noise even when trained with similar progress supervision, whereas UAVProgressor provides sufficiently accurate and temporally consistent estimates to support policy optimization.

Reward components. Removing r_t^{term}, r_t^{prog}, or r_t^{sched} increases SADE from **0.1280** to 0.3547, 0.2986, and 0.2568, respectively. The terminal term is most critical for suppressing residual motion, while the progress and schedule terms provide direct advancement and pacing supervision. Importantly, the variant without r_t^{sched} still substantially outperforms SFT at 0.6147. Because this variant does not use the reference schedule in its reward, the improvement cannot be attributed solely to the schedule shared with SADE. Full results are in Appendix Table 5.

Demonstration source. Table 3 shows that UAVProgressor-RL reduces SADE from 0.6813 to 0.2089 with EITS and from 0.6147 to **0.1280** with UAVFlow expert demonstrations. The improvement therefore holds across both demonstration sources.

Reinforcement-learning optimizer. Under the same SFT initialization, reward, and interaction budget, PPO improves SADE from 0.6147 to **0.1280**, whereas GRPO reaches 0.5578. Thus, optimizer choice remains important even with the same dense progress signal. Full curves are provided in Appendix D.1.

6 Conclusion

We presented UAVProgressor-RL, a progress-aware reinforcement-learning framework that converts trajectory-derived task progress into dense feedback for UAV policy optimization. UAVProgressor uses visual history to estimate execution progress without access to the absolute timestep at inference, and its frozen predictions support progress improvement, schedule alignment, and completion-aware stopping rewards. We also introduced SADE to evaluate whether a policy remains spatially synchronized with the reference execution schedule rather than only following a geometrically plausible path. Across five policy backbones, UAVProgressor-RL reduces overall SADE; with the main world-action policy, it reaches 0.1280 compared with 0.3862 for the strongest reward baseline. The method also remains effective under limited demonstrations and transfers to the evaluated OpenUAV unseen-map subset. These results support task progress as a compact process representation for data-efficient, long-horizon UAV policy optimization.

AI USE STATEMENT

Generative AI tools were used to assist with language polishing and to improve the clarity, grammar, and readability of the manuscript. All AI-assisted edits were reviewed by the authors, who take full responsibility for the final content.

REFERENCES

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025. 6

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594, 2023. 3

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, June 2025. doi: 10.15607/RSS.2025.XXI.010. 6

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, Keerthana Gopalakrishnan, Julian Ibarz, Ofir Nachum, Sumedh Anand Sontakke, Grecia Salazar, Huong T. Tran, Jodilyn Peralta, Clayton Tan, Deeksha Manjunath, Jaspiar Singh, Brianna Zitkovich, Tomas Jackson, Kanishka Rao, Chelsea Finn, and Sergey Levine. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, pp. 3909–3928, 2023. 3

Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Yide Shentu, and Philipp Wu. Sarm: Stage-aware reward modeling for long horizon robot manipulation. In *International Conference on Learning Representations*, 2026. 3, 4

Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. Conrft: A reinforced fine-tuning method for vla models via consistency policy. In *Proceedings of Robotics: Science and Systems*, 2025a. doi: 10.15607/RSS.2025.XXI.019. 2, 3

Zengjue Chen, Runliang Niu, He Kong, and Qi Wang. Tgrpo: Fine-tuning vision-language-action model via trajectory-wise group relative policy optimization. *arXiv preprint arXiv:2506.08440*, 2025b. 3

Yiguo Fan, Pengxiang Ding, Shuanghao Bai, Xinyang Tong, Yuyang Zhu, Hongchao Lu, Fengqi Dai, Wei Zhao, Yang Liu, Siteng Huang, Zhaoxin Fan, Badong Chen, and Donglin Wang. Longvla: Unleashing long-horizon capability of vision language action model for robot manipulation. *arXiv preprint arXiv:2508.19958*, 2025. 3

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021. 3

Dongchi Huang, Zhirui Fang, Tianle Zhang, Yihang Li, Lin Zhao, and Chunhe Xia. Co-rft: Efficient fine-tuning of vision-language-action models through chunked offline reinforcement learning. *arXiv preprint arXiv:2508.02219*, 2025. 3

Dongchi Huang, Hongyin Zhang, Bohan Hou, Siteng Huang, Zhian Su, Hang Guo, Tong Lu, Zhaofeng Xu, Jiahao Tang, Jianfei Yang, Donglin Wang, Peixi Peng, Mingxiu Chen, Deli Zhao, and Xin Li. Rynnvalue: Scaling robotic value foundation models with temporal distance. *arXiv preprint arXiv:2608.09853*, 2026. 3, 4

Kuo-Han Hung, Pang-Chi Lo, Jia-Fong Yeh, Han-Yuan Hsu, Yi-Ting Chen, and Winston H. Hsu. Victor: Learning hierarchical vision-instruction correlation rewards for long-horizon manipulation. In *International Conference on Learning Representations*, 2025. 3

Gabriel Ilharco, Vihan Jain, Alexander Ku, Eugene Ie, and Jason Baldridge. General evaluation for instruction conditioned navigation using dynamic time warping. *arXiv preprint arXiv:1907.05446*, 2019. 8

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In *Proceedings of the Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2024. 6

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. 3

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020. 3

Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. Roboreward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026. 3

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. Simplevla-rl: Scaling vla training via reinforcement learning. In *International Conference on Learning Representations*, 2026. 2, 3

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S. Huang, Luke Zettlemoyer, Dieter Fox, Yu Xiang, Anqi Li, Andreea Bobu, Abhishek Gupta, Stephen Tu, Erdem Biyik, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. In *Robotics: Science and Systems*, 2026. 3, 4, 6

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 16

Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. Vla-rl: Towards masterful and general robotic manipulation with scalable reinforcement learning. *arXiv preprint arXiv:2505.18719*, 2025. 2, 3

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105), 2025. doi: 10.1126/scirobotics.ads5033. 3

Lirui Luo, Guoxi Zhang, Hongming Xu, Yaodong Yang, Cong Fang, and Qing Li. Mvr: Multi-view video reward shaping for reinforcement learning. In *International Conference on Learning Representations*, 2026. 3

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *International Conference on Machine Learning*, pp. 23301–23320, 2023a. 3

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations*, 2023b. 3

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020. 3

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, pp. 278–287, 1999. 4

NVIDIA, Johan Bjorck, Fernando Castaneda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. *arXiv preprint arXiv:2503.14734*, 2025. 6

Julian Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. Worldgym: World model as an environment for policy evaluation. In *International Conference on Learning Representations*, 2026. 3

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 5, 6

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 6

Hao Shi, Weiye Li, Bin Xie, Yulin Wang, Renping Zhou, Tiancai Wang, Xiangyu Zhang, Ping Luo, and Gao Huang. Memoryvla++: Temporal modeling via memory and imagination in vision-language-action models. *arXiv preprint arXiv:2606.09827*, 2026a. 3

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, 2026b. 2

Junyang Shu, Zhiwei Lin, and Yongtao Wang. Rftf: Reinforcement fine-tuning for embodied agents with temporal feedback. *arXiv preprint arXiv:2505.19767*, 2025. 2, 3

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Memer: Scaling up memory for robot control via experience retrieval. In *International Conference on Learning Representations*, 2026. 3

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026. 6

Zhian Su, Weijie Kong, Haonan Dong, and Huixu Dong. Ig-rft: An interaction-guided rl framework for vla models in long-horizon robotic manipulation. *arXiv preprint arXiv:2602.20715*, 2026. 3

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, Shaoxuan Xie, Guocai Yao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. General process reward modeling for robotic reinforcement learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. 3, 4, 6

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. 4

Xiangyu Wang, Donglin Yang, Yue Liao, Wenhao Zheng, Wenjun Wu, Bin Dai, Hongsheng Li, and Si Liu. Uav-flow colosseo: A real-world benchmark for flying-on-a-word uav imitation learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. 2, 5

Xiangyu Wang, Donglin Yang, Ziqin Wang, Hohin Kwan, Jinyu Chen, Wenjun Wu, Hongsheng Li, Yue Liao, and Si Liu. Towards realistic UAV vision-language navigation: Platform, benchmark, and methodology. In *The Thirteenth International Conference on Learning Representations*, 2025b. 2, 5

Ziye Wang, Modi Shi, Chaojun Ni, Jiazhi Yang, Mengdi Li, Zhizhong Su, Tianwei Lin, and Hongyang Li. Nativemem: Native memory compression for long-horizon robotic manipulation. *arXiv preprint arXiv:2607.06678*, 2026. 3

Wenke Xia, Pei Ren, Wenbo Yu, Yizhuo Zhang, Jifan Li, Yixue Zhang, Yinuo Zhao, Qingyang Gao, Jianlong Fu, Jian Tang, Ji-Rong Wen, Zhengping Che, and Di Hu. Robo-valuerl: Reliable value estimation for offline-to-online reinforcement learning. *arXiv preprint arXiv:2607.09866*, 2026. 3

Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, Sitong Mao, Junyi Dong, Tianxing Chen, Jiaqi Peng, Jing Xiong, Jiafei Cao, Jifeng Dai, Wengang Zhou, Yao Mu, and Tai Wang. Eventvla: Event-driven visual evidence memory for long-horizon vision-language-action policies. *arXiv preprint arXiv:2606.20092*, 2026a. 3

Jiarui Yang, Yehao Lu, Yuning Su, Yu Zhong, Yufeng Xie, Yazhou Zhang, Haiyu Lan, Kaixiang Lu, Peiwen Lin, Chuang Wang, Junwei Liang, and Enyu Li. Temporalflow-vla: Learning physically grounded execution history for long-horizon robot manipulation. *arXiv preprint arXiv:2608.26821*, 2026b. 2

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026. 6

Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, pp. 537–546, 2022. 3

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. In *Conference on Robot Learning*, 2025. 4

Yulin Zhou, Yimeng Wang, Nengyu Wang, Shaojia Xing, Shiyun Tu, Xiang Li, Jingkai Zhang, Ningbo Jiang, Yuankai Lin, Hua Yang, Xiangrui Zeng, and Zhouping Yin. Chronos: A physics-informed full-history framework for non-markovian long-horizon manipulation. *arXiv preprint arXiv:2606.30318*, 2026. 2

Fangqi Zhu, Zhengyang Yan, Zicong Hong, Quanxin Shou, Xiao Ma, and Song Guo. Wmpo: World model-based policy optimization for vision-language-action models. In *International Conference on Learning Representations*, 2026. 3

A Experimental Setup and Implementation Details

This section provides additional implementation details of the simulation platform, policy execution protocol, training configuration, and online interaction pipeline used throughout our experiments. Unless otherwise stated, all policy comparisons use the same simulation interface, action constraints, and online execution protocol.

A.1 Simulation and Computing Platform

All experiments are conducted in an Unreal Engine-based UAV simulation environment. The simulation environment is deployed on a Windows workstation equipped with a single NVIDIA GeForce RTX 5060 GPU. The workstation is used primarily for environment rendering, physics simulation, UAV state updates, and action execution.

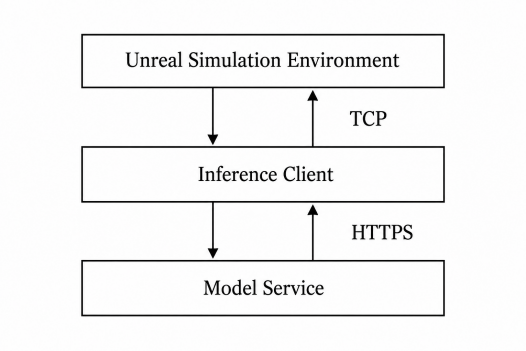


Figure 6: **Distributed simulation and training platform.** Five Unreal Engine environments are executed in parallel on a Windows simulation workstation equipped with an NVIDIA GeForce RTX 5060 GPU. Visual observations and UAV states are transferred to a remote computing server, where policy inference, progress estimation, reward computation, and policy optimization are performed. Predicted actions are then returned to the corresponding simulation environments for execution, forming a closed-loop interaction pipeline.

To accelerate online interaction and rollout collection, we launch five independent Unreal simulation instances in parallel on the same workstation. Each instance maintains an independent UAV environment and executes a separate rollout. This configuration allows multiple trajectories to be collected simultaneously while sharing the same simulation host.

Model computation is separated from environment simulation. The Windows simulation workstation communicates with a remote computing server through a network connection. All policy inference, UAVProgressor inference, reward computation, and model optimization are performed on the remote server. Consequently, the local workstation is responsible mainly for simulation and environment interaction, whereas the remote server performs the computation-intensive components of training and inference.

During online training, observations and UAV states collected from the five parallel environments are transferred to the remote server. The server processes observations from multiple environments, performs policy inference, and returns the predicted actions to the corresponding simulation instances. After these actions are executed, the environments produce the next observations and UAV states, which are sent back to the remote server for the next decision step. This process forms a distributed closed-loop interaction pipeline.

Separating simulation rendering from model computation provides two practical advantages. First, it prevents the computational load of the policy and progress models from directly competing with Unreal Engine rendering on the same GPU. Second, parallelizing environment interaction across five simulation instances increases rollout throughput during reinforcement learning and online evaluation.

Figure 6 illustrates the communication architecture of the experimental platform.

A.2 Policy Execution Protocol

We use a unified execution protocol for all policy baselines and UAVProgressor-RL variants. The UAV state at timestep t is represented as

$$\mathbf{s}_t = (x_t,\, y_t,\, z_t,\, \psi_t)\,, \tag{4}$$

where (x_t, y_t, z_t) denotes the UAV position and ψ_t denotes its yaw angle.

The policy predicts relative translational and rotational actions,

$$\mathbf{a}_t = (\Delta x_t,\, \Delta y_t,\, \Delta z_t,\, \Delta\psi_t)\,, \tag{5}$$

where Δx_t, Δy_t, and Δz_t denote translational displacements along the three spatial axes, and $\Delta\psi_t$ denotes the change in yaw.

The trajectory data are recorded at 10 Hz, while policies are executed at 5 Hz during closed-loop evaluation and reinforcement learning. At each policy query, the model predicts an action sequence with a horizon of eight steps,

$$\hat{\mathbf{A}}_t = [\hat{\mathbf{a}}_t, \hat{\mathbf{a}}_{t+1}, \ldots, \hat{\mathbf{a}}_{t+7}]\,. \tag{6}$$

Rather than executing the complete predicted sequence, only the first two actions are applied to the environment before a new observation is acquired and the policy is queried again. The resulting receding-horizon execution scheme can be written as

$$\hat{\mathbf{A}}_t \rightarrow \{\hat{\mathbf{a}}_t, \hat{\mathbf{a}}_{t+1}\} \rightarrow I_{t+2}, \mathbf{s}_{t+2} \rightarrow \hat{\mathbf{A}}_{t+2}, \tag{7}$$

where I_t denotes the visual observation at timestep t. Replanning after every two executed actions allows the policy to continuously incorporate new visual and state feedback while maintaining short-term action consistency.

To prevent unstable exploration or excessively large actions during online optimization, translational displacement is clipped to 150 cm and yaw change is clipped to 0.8 rad. The same limits are applied to all compared policies:

$$\|\Delta\mathbf{x}_t\| \leq 150\ \text{cm}, \qquad |\Delta\psi_t| \leq 0.8\ \text{rad}, \tag{8}$$

where $\Delta\mathbf{x}_t = (\Delta x_t, \Delta y_t, \Delta z_t)$.

These execution settings are fixed across supervised baselines, reward-model baselines, and UAVProgressor-RL so that performance differences are not caused by different control frequencies, prediction horizons, or action constraints.

A.3 Training Configuration

UAVProgressor-RL follows the two-stage training procedure described in Section 3. The first stage independently establishes a supervised UAV policy and a progress predicting model. The second stage uses the frozen progress model to provide dense feedback for online policy optimization.

In the first stage, the UAV policy is trained from demonstration trajectories using action supervision. Given the observed visual history, UAV state, and task instruction, the policy predicts a sequence of future UAV actions. The supervised checkpoint obtained in this stage serves as the common initialization for subsequent reinforcement-learning experiments.

UAVProgressor is trained separately using complete demonstration trajectories. For a trajectory containing T observations, the normalized progress target at timestep t is defined as

$$\phi_t = \frac{t}{T-1}. \tag{9}$$

The normalized temporal index is used only to construct the training target. Neither the absolute timestep t nor the total trajectory duration T is provided to UAVProgressor during inference. Instead, progress is estimated from the task instruction and the observed visual trajectory prefix.

At inference time, UAVProgressor predicts

$$\hat{\phi}_t = f_\psi\left(I_{t-K+1:t}, l\right), \tag{10}$$

where $I_{t-K+1:t}$ denotes the available visual history and l denotes the task instruction. Future observations are never used during online execution.

After progress-model training, UAVProgressor is frozen throughout policy optimization. The policy and UAVProgressor do not share trainable parameters, and no policy gradient is propagated into

the progress predicting model. Consequently, the evaluator remains fixed while the policy changes during reinforcement learning, avoiding reward drift caused by simultaneously updating the reward estimator.

For the primary experiments, PPO is used for policy optimization. GRPO is additionally evaluated as an alternative optimizer under the same SFT initialization, progress model, reward construction, environment interaction budget, and evaluation protocol. Detailed reward coefficients and optimizer hyperparameters are provided in Appendix A.4.

A.4 Reward and Optimization Hyperparameters

Reward configuration. We use a fixed reward configuration for all main PPO experiments. The progress-improvement reward uses a clipping threshold of $r_{\max} = 0.2$. The schedule-alignment coefficient is $\lambda_{\text{sched}} = 1.0$, with a schedule tolerance of $\epsilon_{\text{sched}} = 0.05$. The completion-aware reward uses $\lambda_{\text{term}} = 1.0$.

Completion is detected when the predicted progress enters the terminal region and remains stable for $H = 3$ consecutive execution steps. The implementation uses a terminal progress tolerance of 0.02 from full completion; equivalently, this corresponds to $\phi_{\text{term}} = 0.98$ when the terminal condition is expressed in the threshold form used in Section 3.2.

Visual-history configuration. We set $K = 10$ for UAVProgressor input, corresponding to nine historical observations and the current observation. The historical observations consist of six anchor frames and three recent frames, with a recent-frame stride of one. The completion-hold parameter $H = 3$ is independent of the visual-history length K.

Progress-model objective. The loss ℓ in Section 3.1 is implemented as mean-squared error for current progress prediction. The complete progress-training objective is

$$\mathcal{L}_{\text{prog}} = \mathcal{L}_{\text{cur}} + 0.2\,\mathcal{L}_{\text{aux}} + 0.2\,\mathcal{L}_{\Delta}, \tag{11}$$

where

$$\mathcal{L}_{\text{cur}} = \text{MSE}\left(\hat{\phi}_t, \phi_t\right), \tag{12}$$

$$\mathcal{L}_{\text{aux}} = \text{MSE}\left(\hat{\phi}_t^{\text{aux}}, \phi_t\right), \tag{13}$$

and

$$\mathcal{L}_{\Delta} = \text{SmoothL1}\left((\hat{\phi}_t - \hat{\phi}_{t-1}) - (\phi_t - \phi_{t-1})\right). \tag{14}$$

UAVProgressor is optimized using AdamW (Loshchilov & Hutter, 2019) with a learning rate of 3×10^{-5} and weight decay of 0.01. We use gradient clipping of 1.0, a cosine learning-rate schedule, a warmup ratio of approximately 5%, and bf16 mixed precision.

PPO configuration. Table 4 summarizes the PPO hyperparameters used in the main experiments.

Chunk-level optimization. As described in Appendix A.2, each policy query predicts eight future actions and executes the first two before replanning. Accordingly, one PPO transition corresponds to two physical actions.

For the two physical-action rewards $r_{t,0}$ and $r_{t,1}$, the chunk-level reward is

$$R_t^{\text{chunk}} = r_{t,0} + \alpha r_{t,1}, \qquad \alpha = \sqrt{0.99} \approx 0.99499. \tag{15}$$

GAE and PPO optimization are then performed at the chunk level using $\gamma = 0.99$ and $\lambda_{\text{GAE}} = 0.95$.

Table 4: PPO hyperparameters used in the main experiments.

Hyperparameter	Value
Discount factor γ	0.99
GAE coefficient λ_{GAE}	0.95
PPO clip ratio	0.20
Value clip ratio	0.20
Value Huber delta	1.0
PPO update epochs	2
Global batch size	64
Micro batch size	1
Number of minibatches	1
Advantage normalization	Yes
Action entropy coefficient	0.0
PPO KL coefficient	0.0
Actor learning rate	1×10^{-6}
Value-head learning rate	3×10^{-5}
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1×10^{-8}
Weight decay	0.01
Gradient clipping	1.0

Online interaction budget. The main PPO experiments use five parallel simulation environments. Each optimization epoch contains two rollout epochs, with at most 32 chunk-level interaction steps per environment. With 500 optimization epochs, the configured upper bound is

$$B_{\mathrm{chunk}} = 5 \times 2 \times 32 \times 500 = 160{,}000 \tag{16}$$

chunk-level environment transitions.

Since each chunk executes two physical actions, this corresponds to

$$B_{\mathrm{action}} = 2B_{\mathrm{chunk}} = 320{,}000 \tag{17}$$

physical actions. Early episode termination, rendering failures, and environment resets may reduce the realized interaction count. Evaluation interactions are excluded. All compared RL methods use the same online interaction budget.

A.5 Online Rollout and Communication

Online policy optimization is implemented as a closed-loop interaction between the five parallel Unreal environments and the remote computing server. At each decision step, environment e provides its current observation and UAV state,

$$\mathcal{O}_t^{(e)} = \left(I_t^{(e)}, \mathbf{s}_t^{(e)}\right), \qquad e \in \{1, \ldots, 5\}. \tag{18}$$

The observations collected from active environments are transmitted to the remote server. The server performs policy inference to obtain the predicted action sequence for each environment,

$$\hat{\mathbf{A}}_t^{(e)} = \pi_\theta \left(I_{t-K+1:t}^{(e)}, \mathbf{s}_t^{(e)}, l^{(e)}\right). \tag{19}$$

For UAVProgressor-RL, the same observed trajectory prefix is also evaluated by the frozen progress predicting model,

$$\hat{\phi}_t^{(e)} = f_\psi\left(I_{t-K+1:t}^{(e)}, l^{(e)}\right). \tag{20}$$

The resulting progress predictions are used to compute the progress improvement, schedule alignment, and completion-aware stopping rewards. The reward signal and rollout transitions are then accumulated on the remote server for policy optimization.

The first two predicted actions are returned to each corresponding Unreal environment and executed sequentially. After execution, new observations and states are collected and transmitted to the server, closing the perception, prediction, execution, and feedback loop.

For the five parallel environments, one interaction cycle can therefore be summarized as

$$\begin{aligned} \text{Unreal environments} &\rightarrow \text{observations and states} \\ &\rightarrow \text{policy and progress inference} \\ &\rightarrow \text{reward computation} \\ &\rightarrow \text{predicted actions} \\ &\rightarrow \text{Unreal environments.} \end{aligned} \tag{21}$$

Policy optimization is performed on the remote server using the rollout data collected through this interaction pipeline. Because all methods share the same parallel simulation platform and communication protocol, differences in reported policy performance reflect the learning method rather than differences in environment throughput or execution interfaces.

A.6 Generalization Analysis Protocol

To examine how cross-task generalization changes during policy optimization, we evaluate SFT and UAVProgressor-RL at six training checkpoints. For UAVFlow, three selected task instances are used to compare performance on optimized seen tasks with held-out tasks. For OpenUAV, three additional task instances from unseen environments are evaluated using the same checkpointed policies.

Each comparison starts from the same initial SFT policy. We plot seen-task SADE against the corresponding held-out-task or unseen-map SADE, so that the analysis measures whether continued improvement on the optimization distribution is accompanied by degradation in transfer performance. This checkpoint-based analysis is separate from the final evaluation on the 48-trajectory OpenUAV subset reported in the main paper.

B Dataset Construction and Evaluation Splits

This section provides additional details on the organization of UAVFlow, the held-out split used for progress prediction, the construction of the OpenUAV out-of-distribution evaluation subset, and the demonstration subsets used in the data-scarcity experiments. Unless otherwise stated, all dataset splits are constructed at the trajectory level rather than at the individual frame level.

B.1 UAVFlow Data Organization

UAVFlow is used as the primary benchmark for both policy learning and progress prediction. We reorganize the original data into a unified trajectory-based format so that action learning, progress supervision, and online evaluation share the same temporal representation.

The reorganized UAVFlow dataset contains 10,109 trajectories as the training pool and reserve 273 trajectories as a held-out test set. All splits are constructed at the trajectory level, such that observations from the same trajectory never appear in both training and evaluation. The 10,109 training trajectories serve as the source pool for policy learning, progress supervision, and the demonstration-budget experiments, whereas the 273 held-out trajectories are used only for evaluation.

Each trajectory is represented as an ordered sequence

$$\tau = \{(I_t, \mathbf{s}_t, \mathbf{a}_t)\}_{t=0}^{T-1}, \tag{22}$$

where I_t denotes the first-person visual observation, $\mathbf{s}_t$ denotes the UAV state, and $\mathbf{a}_t$ denotes the corresponding control action. The UAV state is represented by position and yaw,

$$\mathbf{s}_t = (x_t, y_t, z_t, \psi_t), \tag{23}$$

and the action is represented by relative translation and yaw displacement,

$$\mathbf{a}_t = (\Delta x_t, \Delta y_t, \Delta z_t, \Delta \psi_t). \tag{24}$$

For progress-model training, an additional normalized progress annotation is generated automatically from the temporal ordering of each demonstration. The stored trajectory can therefore be written as

$$\tau = \{(I_t, \mathbf{s}_t, \mathbf{a}_t, \phi_t)\}_{t=0}^{T-1}, \tag{25}$$

where ϕ_t denotes the reference execution progress.

The evaluated UAVFlow tasks cover six representative UAV behaviors. We group them into long-horizon and short-horizon categories according to their execution characteristics. The long-horizon group contains *SurroundApproach*, *SurroundCircle*, and *SurroundLeave*, which require sustained temporal reasoning and multi-stage motion. The short-horizon group contains *Approach*, *Follow-Straight*, and *FollowTurn*, which primarily involve target approaching and tracking behaviors.

All policy methods are evaluated on the same task splits and trajectories. Consequently, differences reported in the main paper do not result from different evaluation data or task compositions.

B.2 Progress Prediction Split

Progress prediction is evaluated on a held-out set of 273 UAVFlow trajectories that is not used to optimize UAVProgressor. The evaluation is performed over complete trajectories rather than randomly sampled independent frames, so that both absolute progress accuracy and temporal consistency can be measured over the full execution process.

For a trajectory containing T observations, the reference progress at timestep t is generated as

$$\phi_t = \frac{t}{T-1}, \qquad t \in \{0, \ldots, T-1\}, \tag{26}$$

which maps trajectories of different durations into a common normalized interval $[0, 1]$. The first observation is assigned $\phi_0 = 0$, while the final observation is assigned $\phi_{T-1} = 1$.

Importantly, the normalized frame index is used only for constructing the supervision target. During inference, UAVProgressor is not given the absolute timestep t, the trajectory length T, the remaining trajectory duration, or any future observation. Its prediction is based only on the task instruction and the visual observations available up to the current timestep,

$$\hat{\phi}_t = f_\psi(I_{t-K+1:t}, l). \tag{27}$$

This distinction is important because the goal of UAVProgressor is not to recover a trajectory clock, but to infer the current execution stage from observed visual evolution.

All compared progress predictors are evaluated on the same 273 trajectories with the same reference progress labels. MAE, PCC, and SRCC are computed over the held-out evaluation data using a shared metric implementation. Additional temporal perturbation analyses are used to examine whether the predicted progress responds to actual visual state changes rather than simply increasing with elapsed time.

B.3 OpenUAV OOD Subset

We additionally use OpenUAV to evaluate cross-domain generalization beyond the UAVFlow training distribution. No OpenUAV trajectory is used for supervised policy training, UAVProgressor training, reward-model adaptation, or reinforcement-learning optimization.

The OpenUAV evaluation set is constructed from the official *Test Unseen Map* split. We select 48 short-horizon trajectories and match their trajectory lengths to the short-horizon evaluation distribution used in UAVFlow. Length matching is applied to reduce the influence of trajectory duration when comparing cross-domain performance. Without this constraint, a substantial difference in episode length could affect trajectory-level evaluation independently of visual-domain generalization.

Formally, if $\mathcal{D}_{\mathrm{UAV}}^{\mathrm{short}}$ denotes the short-horizon UAVFlow evaluation set and $\mathcal{D}_{\mathrm{OU}}^{\mathrm{unseen}}$ denotes the candidate OpenUAV unseen-map trajectories, the selected subset $\mathcal{D}_{\mathrm{OU}}^{48}$ is constructed such that

$$\mathcal{D}_{\mathrm{OU}}^{48} \subset \mathcal{D}_{\mathrm{OU}}^{\mathrm{unseen}}, \qquad \left|\mathcal{D}_{\mathrm{OU}}^{48}\right| = 48, \tag{28}$$

with its trajectory-duration distribution chosen to be comparable to that of the UAVFlow short-horizon evaluation set.

This evaluation is intended specifically as a targeted out-of-distribution test rather than a complete evaluation of the OpenUAV benchmark. The primary domain shift therefore comes from unseen environments, scene appearance, and map configuration, while the effect of substantially different trajectory lengths is reduced.

For the main cross-domain comparison, the same trained policy is transferred directly from UAVFlow to the selected OpenUAV trajectories without fine-tuning. The OpenUAV SADE reported in the main paper therefore measures zero-shot transfer of schedule-consistent policy execution to the evaluated unseen environments.

B.4 Demonstration-Budget Subsets

To evaluate learning under limited expert demonstrations, we construct a series of UAVFlow training subsets containing

$$25,\ 50,\ 100,\ 200,\ 400,\ 800 \tag{29}$$

expert demonstration trajectories.

Rather than independently resampling a new dataset for each demonstration budget, we construct the subsets in a nested manner:

$$\mathcal{D}_{25} \subset \mathcal{D}_{50} \subset \mathcal{D}_{100} \subset \mathcal{D}_{200} \subset \mathcal{D}_{400} \subset \mathcal{D}_{800}. \tag{30}$$

This design ensures that increasing the demonstration budget adds new trajectories to the previous subset instead of replacing the data already used. It therefore reduces additional variance caused by independently resampling training sets at different data scales.

The subsets are selected to preserve the task composition of the available UAVFlow training data as closely as possible. At each demonstration budget, all compared methods use exactly the same demonstration subset. The SFT policy is first trained using the corresponding subset, and reinforcement-learning methods are initialized from this matched SFT checkpoint.

For methods that require UAVFlow-specific reward-model training or adaptation, the available demonstration data are restricted to the same budget. In particular, the progress predicting model and competing learned reward models are not allowed to access additional UAVFlow demonstrations beyond those available under the corresponding data-scarcity setting.

We treat expert demonstrations and online environment interaction as two separate resources. The demonstration budget controls only the amount of offline expert data available to the learning

method, while the online interaction budget used during reinforcement learning is matched across the compared RL methods at each data scale. This prevents an RL method from compensating for fewer demonstrations by using additional environment interaction.

All demonstration-budget models are evaluated on the same fixed UAVFlow test set. The resulting comparison therefore measures how policy performance changes as the amount of expert supervision is reduced, while keeping the evaluation distribution and online interaction conditions unchanged.

C Evaluation Definitions

This section provides the mathematical definitions used in the experimental evaluation. The main paper describes the interpretation of each metric and evaluation setting, while the corresponding formal definitions are collected here for completeness.

C.1 UAVFlow Action Representation

At timestep t, the UAV action is represented as

$$\mathbf{a}_t = (\Delta x_t, \Delta y_t, \Delta z_t, \Delta \psi_t)\,, \tag{31}$$

where Δx_t, Δy_t, and Δz_t denote translational displacements along the three spatial axes, and $\Delta \psi_t$ denotes the change in yaw.

C.2 Reference Progress

For an evaluation trajectory containing N frames, the reference Progress of frame t is defined as

$$p_t = \frac{t}{N-1}, \tag{32}$$

where the first frame corresponds to Progress 0 and the final frame corresponds to Progress 1.

This normalized definition provides a common execution coordinate for trajectories of different durations. The same $[0, 1]$ convention is used for progress supervision, but the progress model does not receive the frame index, episode duration, or reference schedule during inference. Instead, UAVProgressor must infer the current execution stage from the task instruction and the available visual history.

C.3 Schedule-Aligned Displacement Error

Trajectory-similarity metrics based on dynamic time warping allow temporal realignment between an executed trajectory and its reference. This is useful for measuring geometric path similarity, but can under-penalize progress-related execution errors. For example, a policy may follow a reasonable spatial path while progressing too slowly, repeatedly moving forward and backward, or continuing beyond the intended completion stage.

We therefore introduce **Schedule-Aligned Displacement Error (SADE)** to measure whether the spatial execution of a policy remains synchronized with the reference task progress. Let the reference trajectory be

$$\tau_{\mathrm{gt}} = \{\mathbf{g}_i\}_{i=0}^{N-1}\,, \tag{33}$$

where

$$\mathbf{g}_i = \left(x_i^{\mathrm{gt}}, y_i^{\mathrm{gt}}, z_i^{\mathrm{gt}}\right), \tag{34}$$

and let the closed-loop policy rollout be

$$\tau_{\text{pred}} = \{\mathbf{x}_t\}_{t=0}^{H-1}, \tag{35}$$

where $\mathbf{x}_t$ denotes the actual UAV position reached after policy execution.

For each policy timestep t, the corresponding reference schedule is

$$c_t = \text{clip}\left(\frac{t}{T_{\text{ref}}}, 0, 1\right), \tag{36}$$

The same normalized schedule is used by the schedule-alignment reward as a reference execution coordinate, but the reward and SADE operate on different quantities. During policy optimization, r_t^{sched} compares the expected progress c_t with the visually inferred progress $\hat{\phi}_t$ and does not use the reference spatial position $\mathbf{g}_i$. In contrast, SADE uses c_t only to locate the corresponding position on the reference trajectory and then measures the spatial displacement between this reference position and the actual UAV position. Therefore, the schedule reward regulates whether execution falls behind the expected task pace, whereas SADE evaluates whether the resulting spatial trajectory remains synchronized with the reference execution.

where T_{ref} denotes the duration of the reference expert execution expressed in policy timesteps. The corresponding continuous index on the reference trajectory is

$$u_t = c_t(N-1). \tag{37}$$

Let

$$k_t = \lfloor u_t \rfloor, \qquad \alpha_t = u_t - k_t. \tag{38}$$

Because the expert trajectory and online rollout may use different sampling rates, the reference position at the corresponding schedule progress is obtained by linear interpolation:

$$\tilde{\mathbf{g}}_t = (1-\alpha_t)\mathbf{g}_{k_t} + \alpha_t \mathbf{g}_{\min(k_t+1, N-1)}. \tag{39}$$

The raw schedule-aligned displacement is

$$D_{\text{SADE}} = \frac{1}{H}\sum_{t=0}^{H-1} \|\mathbf{x}_t - \tilde{\mathbf{g}}_t\|_2 . \tag{40}$$

To remove the effect of trajectory scale, we normalize this displacement by the total spatial length of the reference trajectory,

$$L_{\text{gt}} = \sum_{i=0}^{N-2} \|\mathbf{g}_{i+1} - \mathbf{g}_i\|_2 , \tag{41}$$

and define

$$\text{SADE} = \frac{D_{\text{SADE}}}{L_{\text{gt}}}. \tag{42}$$

SADE is dimensionless and lower-bounded by zero. A value of zero indicates perfect agreement with the reference execution schedule. The metric is not upper-bounded; values larger than one are possible when the mean schedule-aligned displacement exceeds the total spatial length of the reference trajectory. No clipping is applied to the final SADE value.

For an evaluation set containing K trajectories, we first compute SADE independently for each trajectory and then report the unweighted mean,

$$\mathrm{SADE}_{\mathrm{dataset}} = \frac{1}{K}\sum_{j=1}^{K}\mathrm{SADE}^{(j)}. \tag{43}$$

Unlike nDTW, SADE does not search for an optimal temporal correspondence between the two trajectories. Each executed UAV position is evaluated against the position prescribed by the reference schedule at that timestep. Consequently, insufficient advancement, excessive advancement, repeated reversals, and motion beyond the intended completion stage directly increase SADE. Lower SADE indicates more consistent progress-aware trajectory execution.

C.4 Progress Trend Direction Analysis

To further examine whether a progress model responds to actual visual progress changes rather than simply increasing with elapsed time, we perform a segment-level progress-trend analysis. Instead of evaluating only the absolute progress value, this analysis focuses on whether the model predicts the *direction* of local progress change correctly.

Given a progress sequence $\{\phi_t\}$ or its prediction $\{\hat{\phi}_t\}$, we compute the progress change over a short temporal window of length w:

$$\Delta\phi_t = \phi_{t+w-1} - \phi_t, \qquad \Delta\hat{\phi}_t = \hat{\phi}_{t+w-1} - \hat{\phi}_t. \tag{44}$$

Following the setting used in Fig. 7, we use a window size of $w = 5$, stride 1, and a small threshold $\epsilon = 0.02$ to categorize the local trend into three classes:

$$y_t = \begin{cases} \text{up}, & \Delta\phi_t > \epsilon, \\ \text{stall}, & |\Delta\phi_t| \le \epsilon, \\ \text{rewind}, & \Delta\phi_t < -\epsilon, \end{cases} \tag{45}$$

and similarly for the predicted trend $\hat{y}_t$ using $\Delta\hat{\phi}_t$.

Figure 7 shows the row-normalized confusion matrices of three progress evaluators: RoboMeter, Robo-Dopamine, and UAVProgressor. Each row corresponds to the ground-truth local progress trend, and each column corresponds to the predicted trend. A stronger diagonal concentration indicates that the model is more sensitive to local trajectory changes and better captures whether the task is advancing, pausing, or temporarily reversing.

Compared with RoboMeter and Robo-Dopamine, UAVProgressor exhibits a much clearer diagonal pattern, especially for the *up* and *rewind* classes. In contrast, the two baseline evaluators show substantially more confusion across classes, particularly when distinguishing local reversals or near-stationary segments. This result provides complementary evidence that UAVProgressor does not merely fit a global trajectory clock. Instead, it tracks visually grounded execution changes and can respond to local progress perturbations such as slowing down, pausing, or moving backward. This property is important for our reinforcement-learning framework, because the resulting reward must reflect the actual execution tendency of the policy rather than only the passage of time.

D Additional Results

D.1 PPO and GRPO Optimization Dynamics

Figure 8 compares PPO and GRPO under the same SFT initialization, progress-aware reward, and online interaction budget. Both optimizers start from the same SFT checkpoint with a SADE of 0.6147. PPO reduces SADE to **0.1280**, whereas GRPO reaches 0.5578, consistent with the optimizer comparison reported in the main text.

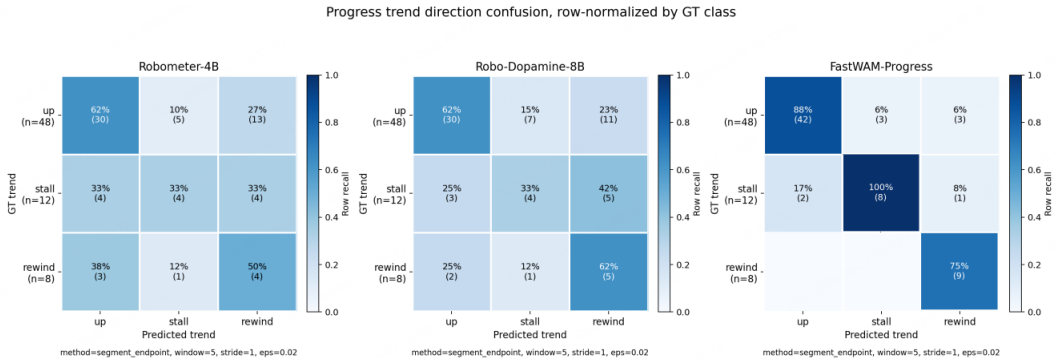


Figure 7: **Progress trend direction confusion matrices.** We evaluate whether the predicted progress follows local execution changes by classifying short trajectory segments into three trend categories: *up*, *stall*, and *rewind*. Each confusion matrix is row-normalized by the ground-truth class. Compared with RoboMeter and Robo-Dopamine, UAVProgressor shows a clearer diagonal pattern, indicating stronger sensitivity to local visual progress changes rather than a simple monotonic dependence on elapsed time. The trend labels are computed using the segment-endpoint method with window size 5, stride 1, and threshold $\epsilon = 0.02$.

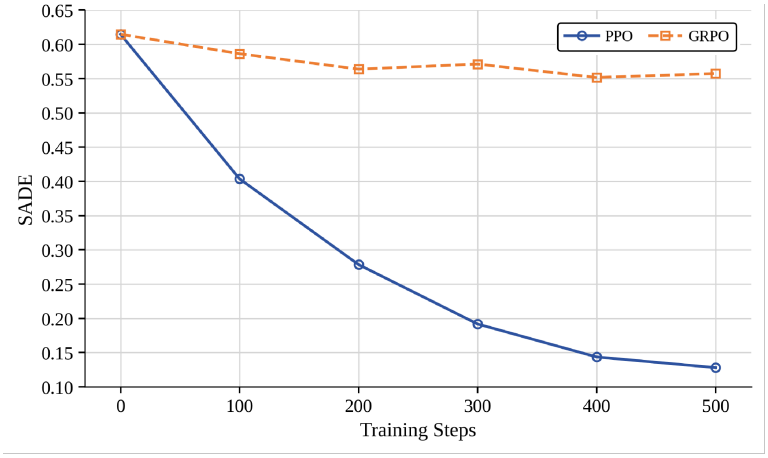


Figure 8: **Optimization dynamics of PPO and GRPO under the same progress-aware reward.** Both optimizers start from the same SFT checkpoint with SADE 0.6147 and use the same UAVProgressor reward and online interaction budget. The curves track SADE during policy optimization, where lower values indicate better schedule-aligned execution. PPO converges to 0.1280, while GRPO reaches 0.5578.

D.2 Reward-Component Ablation

To quantify the contribution of each reward component, we independently remove the progress-improvement, schedule-alignment, and completion-aware terminal terms while keeping all other training settings unchanged. As shown in Table 5, removing any component degrades policy performance. The largest degradation occurs without r_t^{term}, followed by removing r_t^{prog} and r_t^{sched}, indicating complementary roles in stable stopping, task advancement, and execution pacing, respectively. Notably, the variant without r_t^{sched} still achieves a SADE of 0.2568, substantially better than the SFT initialization at 0.6147. This confirms that the improvement cannot be explained solely by the normalized reference schedule shared between the schedule reward and SADE.

Table 5: Ablation of individual reward components on UAVFlow. Each variant removes one component from the full reward while keeping the remaining training settings unchanged. Lower SADE is better.

Reward setting	**SADE ↓**	**Main capability removed**
SFT	0.6147	No episode-level regulation
w/o r_t^{term}	0.3547	Stable terminal stopping
w/o r_t^{prog}	0.2986	Direct task advancement
w/o r_t^{sched}	0.2568	Execution-pace regulation
Full reward	**0.1280**	Advancement + pacing + stopping