LARGE LANGUAGE MODELS AS POLICY OPTIMIZERS FOR COOPERATIVE MULTI-DRONE PAYLOAD TRANSPORT

Anonymous authors
Paper under double-blind review

ABSTRACT

Cooperative transport of a cable-suspended payload requires multiple drones to coordinate formation, obstacle avoidance, and stabilization under coupled dynamics. Learning and adapting these behaviors is challenging when control decisions cannot be readily inspected or revised. We develop a framework that uses pretrained large language models (LLMs) to optimize a shared control program from feedback on simulated transport tasks. The optimizer alternates code revision with local parameter search and compares candidates on the same task instances. The simulator and low-level flight controller remain fixed, and deployment executes the frozen program without online language-model inference. We construct a benchmark covering standard transport, commanded emergency setdown, and stress conditions. Our method achieves higher overall task success than proximal policy optimization (PPO) on this benchmark under the reported training budgets. Optimization-flow ablations support combining diagnostic execution feedback with local parameter search on the main benchmark.

1 INTRODUCTION

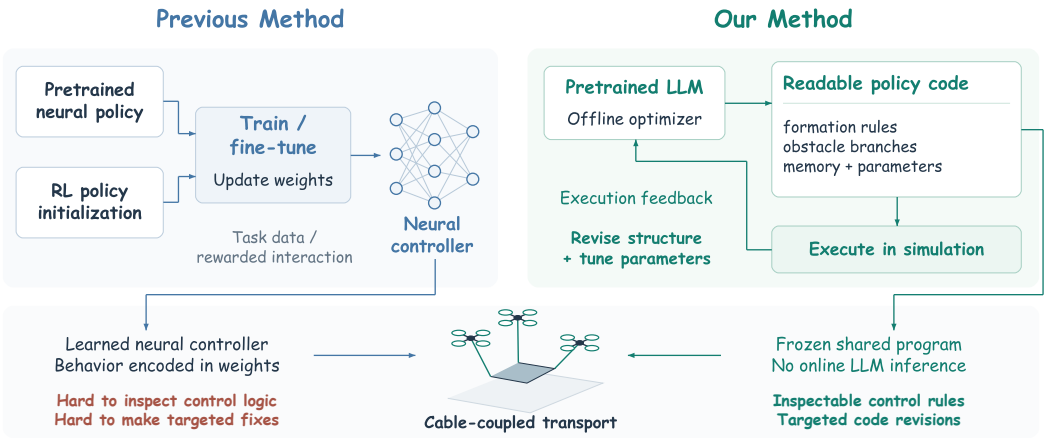

Figure 1: Neural-policy adaptation and reinforcement learning update network weights (left); our framework revises readable policy code offline using a pretrained LLM and execution feedback (right). The frozen shared program controls cable-coupled transport without online LLM inference. This schematic compares policy representations, not measured performance.

Human operation becomes more demanding when multiple drones must act together rather than independently. An operator must simultaneously track individual vehicle motion, coordinate relative positions, maintain safe separation, and react to environmental changes. Cable-suspended transport compounds this difficulty: the vehicles are connected to a shared payload, so one drone's action changes the forces experienced by its teammates. A locally reasonable command can consequently

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.

destabilize the collective motion. Coordinating formation, obstacle avoidance, and payload stabilization through vehicle-by-vehicle commands is therefore challenging, motivating learned cooperative policies that reduce the burden of manual control.

Common approaches in robot learning are adapting pretrained neural policies, such as vision-language-action models exemplified by OpenVLA (Kim et al., 2024), and training neural policies through reinforcement learning, for example with proximal policy optimization (PPO) (Schulman et al., 2017). Drone operations, particularly cooperative multi-drone tasks, can involve complex combinations of environmental disturbances, payloads with varying shapes and mass distributions, obstacles, and unexpected events such as vehicle faults or changes in the transport goal. Pretraining provides reusable knowledge, but does not guarantee that an existing model can directly accommodate the dynamics, action interfaces, and coordination requirements of these tasks. Learning reliable responses across such varied conditions may require substantial task-specific training data and environment interaction, including experience with infrequent but consequential events. Neural policies also offer limited interpretability because their control logic is distributed across network weights. When a failure occurs, it can be difficult to trace the observed behavior to a specific decision mechanism and determine how to modify the policy to address it. These considerations motivate policy optimization from simulated execution feedback, with control rules that remain available for inspection and revision.

As illustrated in Figure 1, we use pretrained language models to optimize an explicit control program from execution feedback, keeping its control logic directly inspectable (Liang et al., 2023; Cheng et al., 2024). Optimization uses task configurations and execution outcomes without expert action demonstrations. Given the current program and its execution records, the model proposes changes to formation rules, obstacle responses, memory updates, and control parameters. Simulations then evaluate the revised program. Because the policy is written as code, its decision rules and revisions can be inspected directly. This allows failures to be investigated in terms of specific control branches and parameters. The language model operates offline. Deployment runs the resulting program without a separate neural controller or online LLM inference.

We instantiate this idea as a programmatic policy learning framework for cable-coupled multi-drone transport, using a pretrained language model as an offline optimizer to improve a shared, executable policy program. The full procedure alternates execution-guided code revision with local parameter search, followed by paired selection in each iteration. To support policy optimization and evaluation within this framework, we construct a synthetic simulation benchmark comprising 1,000 executable task instances with fixed training, validation, and test splits. These instances specify simulation tasks rather than expert demonstrations, covering variations in team size, payload and attachment geometry, obstacles, and disturbances across standard transport, commanded emergency setdown, and stress conditions. Through controlled simulation experiments, we evaluate policy optimization by comparing how program modifications affect cooperative behavior and task performance while holding the simulation dynamics, low-level flight controller, and task verdicts fixed. Our study thus concerns policy-program optimization above a given low-level control interface, with empirical findings evaluated within that simulation setting.

The paper makes the following contributions:

- **A programmatic policy learning framework for coupled aerial teams.** We formulate policy optimization using a pretrained language model as offline improvement of a readable, shared control program. Cooperative behavior is learned through code revision and parameter search without training an additional task-specific neural control policy.

- **A synthetic simulation benchmark for cooperative aerial transport.** We construct executable simulation task instances with fixed training, validation, and test splits, covering diverse physical configurations and task conditions to support reproducible policy optimization and evaluation.

- **A baseline optimizer.** We combine policy-code revision using a pretrained language model with local numerical parameter optimization, using execution feedback and paired candidate evaluation to implement the proposed learning framework.

2 Related Work

Robot programs and execution feedback. Program-based interfaces connect language instructions to robot skills, reactive controllers, spatial value maps, and contact-aware actions (Liang et al., 2023; Singh et al., 2022; Huang et al., 2023; Burns et al., 2024). After initial generation, observation and failure feedback can guide replanning, recovery, and code correction (Huang et al., 2022; Liu et al., 2023; Ying et al., 2025). Other methods search for improved programs across trials. Execution-guided workflow optimization and controller rewriting revise programs from observed outcomes (Cheng et al., 2024; Kumar, 2026). Population-based evolution, visual behavior analysis, repository search, and memory-guided exploration have been used in games and embodied tasks (Lv et al., 2025; Hu et al., 2025; Elmaaroufi et al., 2026; Sygkounas et al., 2026). Benchmarks evaluate these iterative methods under controlled interaction budgets (Wang et al., 2026). The role of the generated code varies across methods. Reward-generation methods produce reward programs for reinforcement learning (Ma et al., 2023; Xie et al., 2023), while direct program optimization revises the acting controller. Structural and numerical search can also be combined in controller design and trajectory optimization (Cui et al., 2025; Shcherba et al., 2025). We apply program optimization to a shared feedback controller for cable-coupled aerial transport, examining how code changes affect success and regressions across task instances.

Multi-robot and aerial code policies. Language models have been used for multi-robot task decomposition and allocation, decentralized plan consensus, and code-based swarm control (Kannan et al., 2023; Li & Zhou, 2025; Strobel et al., 2024). Related systems generate formations, coordinated waypoints, or flight programs that use motion planning, tracking, and safety filtering (Jiao et al., 2023; Schuck et al., 2024; Lykov et al., 2024; Aikins et al., 2024). Aerial systems also use execution feedback to revise programs. Simulation videos and trajectory descriptions guide changes to swarm policies and UAV code, and outcome assessment guides changes to drone behavior trees (Ji et al., 2025; Wang et al., 2025; Zhang et al., 2025). Other methods generate new swarm code when execution stalls or revise cooperative-control programs asynchronously during a mission (Zhu et al., 2025; Li et al., 2026). These methods already separate language-model reasoning from rapid control and use feedback to revise distributed programs. In our setting, the vehicles are physically coupled through one cable-suspended payload. Formation expansion changes cable geometry and load support, which in turn affects obstacle avoidance and braking. We optimize a shared feedback program across training instances, then freeze it for evaluation on held-out instances with fixed flight-control and task interfaces. The program maps observations to actions across transport conditions rather than specifying a single trajectory or mission plan.

3 Programmatic Policy Optimization

3.1 Task Formulation

We optimize executable policies $Q = (P, \theta)$, comprising a program $P \in \mathcal{P}$ with a fixed interface and numerical parameters $\theta \in \Theta(P)$ within declared bounds. For transport instance c, each drone i executes the shared program with its own observation $o_{i,t}$ and persistent memory $m_{i,t}$:

$$(u_{i,t}, m_{i,t+1}) = P(o_{i,t}, m_{i,t}; \theta). \tag{1}$$

The desired velocity $u_{i,t} \in \mathbb{R}^3$ enters the fixed command filter. Each drone maintains private memory initialized with permitted team, payload, cable, and control-limit information (Appendix A). Two to eight drones share a cable-suspended payload; simulator observations cover vehicle, payload, goal, and nearby-environment states.

Let $y_c(Q)$ be 1 if policy Q succeeds on instance c and 0 otherwise, under the fixed mission verdict. On each evaluation panel or split B used in this study, the score is its success rate:

$$S_B(Q) = \frac{1}{|B|} \sum_{c \in B} y_c(Q). \tag{2}$$

Learning seeks to maximize success on the training set by revising the program and its parameters. The implementation uses category weights matching the proportions of every scheduled panel and full split, making its score exactly equivalent to Eq. 2; the full definition appears in Appendix A. The pretrained LLM, simulator, flight controller, and verdict function remain fixed.

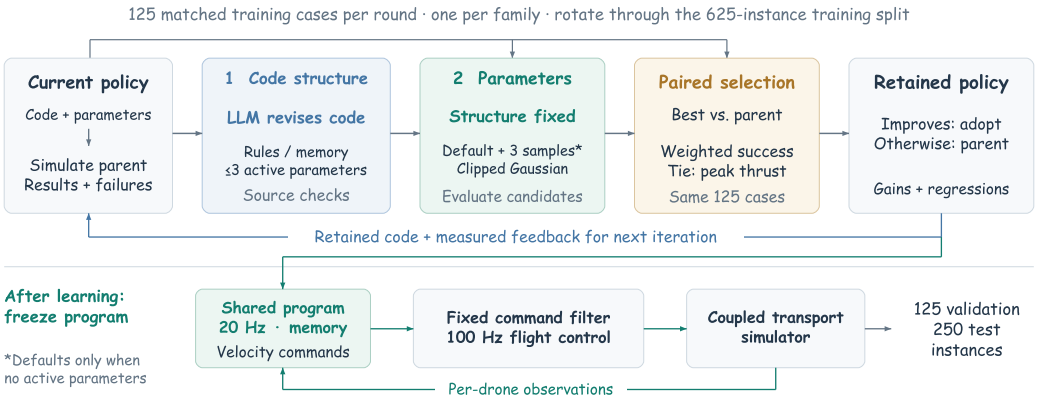

Figure 2: Each optimization iteration performs (1) code revision and (2) local parameter search, followed by paired selection against the current policy. "Best" denotes the candidate retained by sequential comparison, not a global optimum. Selection prioritizes weighted success; ties require a sufficient peak-thrust reduction on common successful deliveries. The retained policy and feedback feed the next iteration. The lower path executes the frozen shared program without online LLM calls. Optimization uses training feedback; the policy is frozen for validation and test.

3.2 Overview

Figure 2 shows the full procedure: two optimization steps followed by paired selection in each iteration. *Current policy* evaluates incumbent Q_k on training panel B_k for execution feedback. In *1 Code structure*, the fixed pretrained LLM uses this evidence to revise the program and declare adjustable parameters. In *2 Parameters*, local settings are evaluated with the proposed structure fixed. *Paired selection* compares the selected candidate and incumbent on the same instances. The *Retained policy* and feedback feed the next iteration: these steps form an alternating loop, not separate training phases.

The full procedure uses 125-instance panels, cycling through five training settings per scenario family. Program-search runs start from the same handwritten controller. After optimization, the lower part of Figure 2 executes only the frozen shared program: 20 Hz velocity commands pass through a fixed command filter and 100 Hz flight controller, without online LLM calls. Optimization uses training feedback; the final policy is frozen before validation and test evaluation.

3.3 Code Revision

The code-revision step, labelled *1 Code structure* in Figure 2, can modify formation rules, obstacle responses, state memory, and numerical constants. It therefore includes changes beyond those expressible by numerical tuning alone. Given the incumbent program, execution feedback, and recent attempt history, the fixed pretrained LLM proposes a replacement program, default parameter values within declared bounds, and at most three active parameters for local search. Disabling external numerical search still permits the LLM to revise constants; the subsequent local-search step samples numerical settings with the proposed program fixed.

Feedback contains per-instance outcomes and selected diagnostic windows; history records previous edits and their effects, including gained and lost successes. The model proposes a code revision and explains its support in the feedback. *Source checks* in Figure 2 verify the interface and imports before simulation. Valid proposals proceed to local parameter search, followed by paired selection to determine incumbent replacement.

3.4 LOCAL PARAMETER SEARCH

The *2 Parameters* block holds the proposed structure fixed. Within one iteration, let θ^0 be its default parameter vector. If any parameters are active, we draw three complete parameter vectors, perturbing each active coordinate j with bounds $[l_j, h_j]$ as

$$\theta_j^{(r)} = \mathrm{clip}_{[l_j,h_j]}\left(\theta_j^0 + \sigma_j \epsilon_j^{(r)}\right), \qquad \epsilon_j^{(r)} \sim \mathcal{N}(0,1). \tag{3}$$

Here $r = 1, 2, 3$ indexes samples and $\sigma_j = 0.12(h_j - l_j)$ sets the perturbation standard deviation to 12% of the allowed range. Out-of-range values are clipped to the bounds, rather than resampled; inactive parameters retain their defaults. Thus, the search evaluates four settings (the default and three samples), or only the default if no parameters are active. All candidates and the incumbent use the same training panel B_k.

3.5 PAIRED SELECTION AND POLICY RETENTION

The *Paired selection* and *Retained policy* blocks use the same comparison, denoted BETTER: a candidate passes if its success rate is higher. If success rates tie, it passes only when mean peak commanded-thrust fraction decreases by at least 0.002 (0.2 percentage points), averaged over deliveries successful under both policies. This secondary comparison excludes emergency setdown and uses the episode-level worst-drone peak with the recorded 0.5 s moving window. A lower success rate, insufficient thrust reduction, or a success tie with no common successful delivery retains the current comparison incumbent. Exact definitions appear in Appendix A.

The full procedure applies this comparison twice: first among the proposed parameter settings, then between the retained proposal and the incumbent. The following steps specify one iteration; every evaluation and comparison uses the same B_k.

One optimization iteration

1. Evaluate incumbent Q_k and obtain execution feedback.
2. Ask the LLM for a program, defaults, and active parameters; check the source. If no valid proposal is obtained, retain Q_k and end this iteration.
3. Evaluate the valid proposal at its defaults and, if applicable, three sampled parameter vectors.
4. Initialize the retained proposal R to the default setting.
5. For samples 1, 2, and 3 in order, replace R with that sample only if BETTER(sample, R) holds. Skip this step if no parameters are active.
6. Set Q_{k+1} to R if BETTER(R, Q_k) holds; otherwise retain Q_k.

Paired evaluation controls for within-round task variation. The retained policy and newly successful and failed cases feed the next iteration (*Gains + regressions* in Figure 2); regressions do not veto a candidate that passes the aggregate rule.

Because common successful deliveries depend on the compared pair, the comparison need not be transitive. Sequential selection is order dependent and can discard a proposal that would pass against Q_k even when the final retained proposal does not. Appendix A discusses this limitation and a possible alternative. Success cannot decrease on the current panel B_k, but rotating panels provide no monotonic-improvement guarantee on the full training set or held-out splits.

4 EXPERIMENTS

4.1 BENCHMARK

Task instances and coverage. The benchmark comprises 1,000 frozen task configurations, not demonstrations, in 125 scenario families. Each family contributes five training, one validation, and two test instances (625/125/250; Table 1). Families vary team size (two to eight drones), payload shape, attachment arrangement, layout, and disturbance combination. Coverage includes four shapes, four attachment arrangements, four layouts, and 16 event types spanning disturbances, faults, sensing and decision errors, and payload or goal changes. Standard transport combines zero to four overlapping or sequential events. Episodes last 20 s; cables carry tension only, allowing prescribed

Table 1: Benchmark allocation. Every family contributes instances to all three splits.

Category	Train	Validation	Test	Total
Standard transport	545	109	218	872
Commanded setdown	40	8	16	64
Stress transport	40	8	16	64
Total	625	125	250	1,000

cuts but no tension-triggered breaks. Coverage is structured, not exhaustive; Appendix B documents generation and configuration counts (Table 9).

Objective-specific verdicts. Delivery requires 3 s within 0.3 m of the current goal without post-lift grounding. Both objectives forbid recorded collisions, mechanical cable breaks, and invalid commands; functioning drones must remain airborne. At fault onset, setdown tasks reveal the mission mode and ground target beneath the payload, without an advance fault schedule. Success requires prior lift without pre-command grounding, downward contact-point speed below 0.5 m/s, first contact within a 1 m horizontal radius, and continuous stable support during the final 1 s: center-of-mass speed at most 0.1 m/s, angular speed at most 0.2 rad/s, and positive support inside the landing region. Contact force and impulse are diagnostic only. Setdown is commanded, not autonomously chosen; simulator verdicts do not certify physical safety.

Splits, input checks, and scope. Splits vary settings, event parameters, and seeds within shared families, measuring within-family generalization rather than transfer to unseen team sizes or event combinations. We report overall and category success because standard transport constitutes 87.2% of instances. Static screening checks initial overlap and event-wise thrust demand, including attached failed drones: standard demand is capped at 68% (below 70% acceptance), while stress demand exceeds 70% and is at most 95%. Screening certifies neither dynamic reachability nor full payload/cable swept-volume clearance; admission is independent of policy success. Versioned configurations, metadata, and rejection records support verified deterministic regeneration. The 180 calibration candidates are excluded. Only 16 test instances each for setdown and stress support coarse comparisons, not precise fault-subtype rankings.

4.2 Implementation Details

Training Settings. All LLM-based program-search methods use `gpt-6-astra` with medium reasoning effort through the `codex_cli` backend. The full method starts from handwritten code and runs for 16 optimization rounds, each combining program revision with local parameter search. Candidate policies implement `decide(view, mem, theta)`, sharing code across drones while maintaining private per-drone memory. Program-search comparisons share the initialization, interface, and proposal budget where applicable; actual computation differs, with recorded costs reported in Table 5.

Evaluation Settings. Main-benchmark evaluation uses the common 250-case test split (internally versioned V3), covering standard transport, commanded emergency setdown, and stress transport; we report overall and category-specific success under fixed mission verdicts. The combined-disturbance evaluation uses separate matched scenes across disturbance subsets (Table 7). Main-benchmark simulation evaluation uses four workers without GPU acceleration, with each episode lasting 20 s. Policies issue velocity commands at 20 Hz through the same fixed command filter and 100 Hz flight controller, without online LLM calls. Task allocation appears in Table 1, with additional implementation and evaluation settings in Appendix A.

4.3 Experiment Results

Policy Comparisons. To assess policy optimization, we compare the full procedure with handwritten control, one-shot and best-of-N independent code generation, and velocity-action PPO (Schulman et al., 2017) on the same test tasks. The full procedure outperforms the reported fresh references overall, although PPO remains stronger on commanded setdown (Table 2). These results

Table 2: Policy performance on the common test split; entries are success percentages. Percentages in the category headers indicate each category's share of all test cases; bold marks the best result in each column. The full method is the frozen 16-round handwritten-start run. See Appendix A.1 for program-search settings.

Method	Standard (87.2%)	Setdown (6.4%)	Stressed (6.4%)	Overall
Handwritten	22.5	0.0	18.8	20.8
One-shot	22.5	6.2	18.8	21.2
Best-of-N	26.6	0.0	18.8	24.4
PPO (velocity) (Schulman et al., 2017)	35.3	**37.5**	12.5	34.0
Full procedure (ours)	**40.8**	25.0	**25.0**	**38.8**

Table 3: Feedback and optimization-stage ablations from the same handwritten initialization and 16-round budget; entries are test success percentages. The full-method reference is the same frozen policy as in Table 2. No external numerical search still permits model edits to code constants.

Method	Standard	Setdown	Stress	Overall
Fixed-code parameter search	15.6	0.0	18.8	14.8
Scalar feedback	30.7	0.0	18.8	28.0
Per-case, no trace	27.5	6.2	18.8	25.6
No external numerical search	33.0	0.0	18.8	30.0
Full procedure (ours)	**40.8**	**25.0**	**25.0**	**38.8**

Table 4: Mechanism removals on the frozen full policy over 250 paired test cases, with no retraining or retuning. Δ is relative to Full; "w/" denotes removal.

Variant	Success (%)	Δ (pp)
Full	38.8	0.0
w/ near-goal braking	29.6	-9.2
w/ carry vertical damping	31.2	-7.6
w/ attachment-motion feedback	41.2	$+2.4$
w/ local obstacle bypass	40.4	$+1.6$
w/ velocity reanchoring	41.6	$+2.8$

support execution-guided program optimization in this setting, without establishing equal-compute superiority. Feedback and optimization-stage variants are reported separately in Table 3. Baseline definitions and the excluded incomplete adaptation are documented in Appendix A.1.

Feedback-content ablation. To test which feedback helps revision, we compare scalar scores, per-case outcomes, and full feedback with diagnostic trajectory windows, using a common handwritten start and removing ablated information from both feedback and revision history. Full feedback performs best, while per-case outcomes alone do not improve overall success over scalar scores (Table 3). The result supports trajectory diagnostics in this setting.

Optimization-stage ablation. To examine the roles of structural revision and parameter search, we either keep code fixed or disable external numerical sampling while allowing the model to edit constants. The handwritten-start full procedure outperforms both variants overall, whereas parameter search alone falls below the handwritten reference (Table 3). Combining the stages is beneficial on the main benchmark, although different simulation usage prevents an equal-cost attribution.

Optimization Cost. Table 5 reports model calls, evaluation requests, simulation attempts, and token use for the fresh training runs and references. Requests include recorded evaluation work and may hit caches; attempts are neither environment steps nor unique valid episodes. PPO interaction counts are listed in Table 8 and are not directly comparable to program proposals. One-shot reuses the predesignated first valid Best-of-N default candidate rather than adding a separate search. Its 0.375K additional evaluation requests and 375 simulation attempts (125 validation and 250 test episodes) are accounted for separately and excluded from the Best-of-N row; generating the shared candidate is already included in that row.

Mechanism removals. To examine which implemented terms support the frozen policy, Table 4 compares five removals on paired main-benchmark test cases, with other settings fixed and no retraining or retuning. Appendix A.1.1 defines the interventions. Removing attachment-motion feedback, local obstacle bypass, or velocity reanchoring yields modest aggregate gains, whereas removing near-goal braking or carry vertical damping causes larger declines. These post-hoc test-set observations suggest scope for targeted simplification of the frozen policy; they do not establish an independently validated optimization procedure.

Table 5: Recorded costs for fresh training and references from Tables 2–3; missing values are unknown, not zero. The full procedure starts from handwritten code. K and M denote 10^3 and 10^6; token counts are rounded.

Method	Calls	Requests (K)	Attempts	Input tokens (M)	Output tokens (K)
Fixed-code parameter search	0	10.75	8875	0	0
Scalar feedback	16	10.75	10375	0.26	55.88
Per-case, no trace	16	10.75	9500	1.025	88.86
No external numerical search	16	4.75	4750	2.716	59.66
Best-of-N	16	10.75	8875	0.258	59.33
Full procedure	16	10.75	10750	2.76	86.24

Table 6: Mechanism interaction in the frozen full policy over 250 paired test cases. Success is in percent; Δ is relative to the full policy in percentage points. No retraining or retuning.

Carry damping	Attachment feedback	Success	Δ
On	On	38.8	0.0
Off	On	31.2	-7.6
On	Off	41.2	$+2.4$
Off	Off	34.4	-4.4

Table 7: Combined-disturbance success (%) of frozen policies on eight paired scenes per subset. Single/pair means equally average the three one-/two-event subsets; Triple enables all events. Event groups: T1, COM shift/frame drop/goal shift; T2, gust/impulse/mass drop; T3, position bias/push/weak motor; T4, COM shift/frame drop/gust. Scenes and seeds are shared across subsets and policies; "w/" denotes the trained no-external-search variant in Table 3.

Group	Policy	None	Single mean	Pair mean	Triple
T1	Handwritten	25.0	33.3	29.2	37.5
	Full method	**75.0**	**54.2**	**50.0**	**62.5**
	w/ external parameter search	50.0	50.0	**50.0**	**62.5**
T2	Handwritten	25.0	37.5	41.7	50.0
	Full method	**62.5**	54.2	62.5	62.5
	w/ external parameter search	50.0	**66.7**	**70.8**	**87.5**
T3	Handwritten	25.0	25.0	25.0	25.0
	Full method	37.5	**41.7**	33.3	**50.0**
	w/ external parameter search	**50.0**	**41.7**	**37.5**	**50.0**
T4	Handwritten	25.0	29.2	25.0	37.5
	Full method	**62.5**	**66.7**	50.0	37.5
	w/ external parameter search	50.0	62.5	**62.5**	**62.5**

Mechanism interaction. Table 6 evaluates the four enabled/disabled combinations of carry vertical damping and attachment-motion feedback on the same test cases, with other code and parameters fixed. Damping improves success with either feedback setting, while removing attachment feedback improves success with damping either enabled or disabled. The effects are nearly additive on aggregate success, but the mechanisms share attachment-velocity information, so these interventions do not isolate independent physical contributions.

Combined-disturbance robustness. To examine whether the main-benchmark advantage extends to disturbance combinations, Table 7 compares frozen handwritten, full-method, and no-external-search policies on matched disturbance subsets, with shared seeds and verdicts and no retraining or retuning. The full method matches or exceeds handwritten control, but its advantage over the no-external-search variant on the main benchmark (Table 3) does not consistently carry over to these combinations. Thus, better aggregate task performance need not imply uniformly improved robustness. Each subset contains only eight paired scenes, so these results characterize the tested

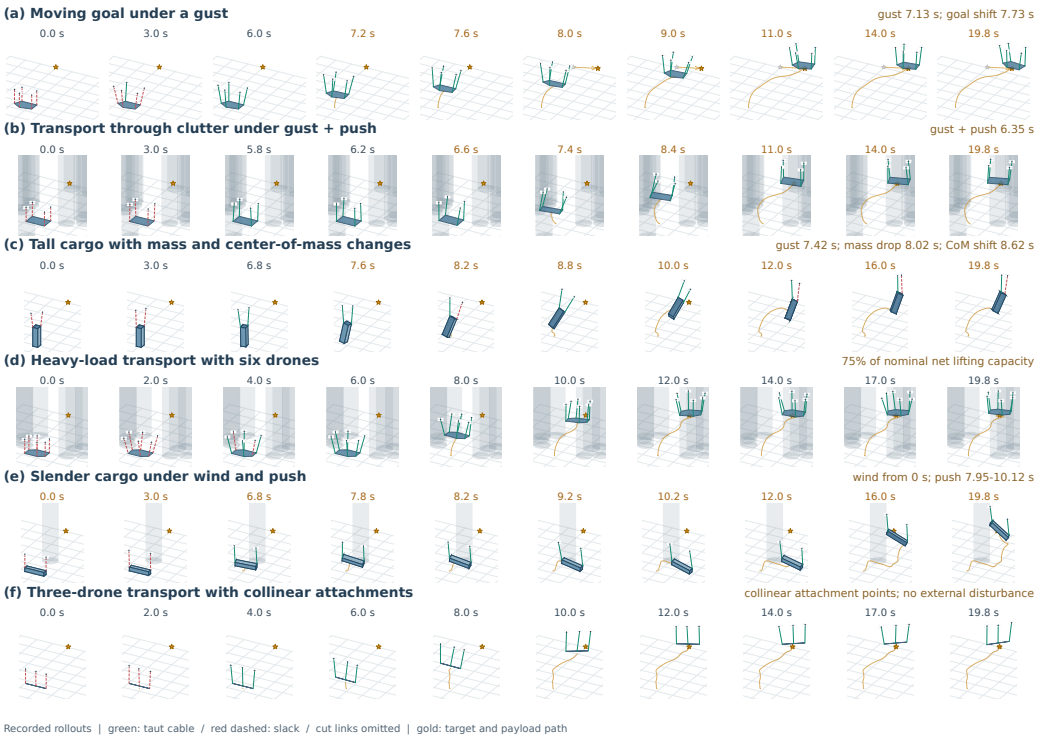

Figure 3: **Recorded transport across task conditions.** Time increases left to right, with denser sampling near disturbances: (a) goal shift during a gust; (b) clutter with gust and push; (c) tall cargo with gust, mass drop, and center-of-mass shift; (d) heavy-load transport; (e) slender cargo under wind and push; (f) transport with collinear attachments. Green/red dashed cables indicate taut/slack states; gold marks targets and payload paths. Views are fixed by row; vehicles and obstacles are schematic. Panels (c, e) use recorded box geometry and orientation; the others show recorded attachment geometry, including collinear points in (f).

combinations rather than establishing general superiority or transfer to combinations guaranteed to be unseen during training.

Qualitative Rollouts. To illustrate controller behavior across task conditions, we inspect recorded transport trajectories with goal changes, clutter, changing payload properties, heavy loads, slender cargo, and collinear attachment points (Figure 3). The selected rollouts complete their recorded transport tasks and illustrate team motion under these conditions. Six distinct additional cases and provenance appear in Appendix C.

5 Limitation and Conclusion

Unequal computational budgets limit causal and superiority claims. Repeated development evaluations and shared scenario families constrain generalization evidence. Mechanism interventions characterize one frozen policy, not outcomes after retraining. Results rely on simulator-derived observations and fixed control/protection layers; deployment under tracking errors, sensing and communication delays, and dynamics mismatch remains untested. Static screening certifies neither dynamic feasibility nor safety, and the program-evolution adaptation remains incomplete.

We optimize shared transport programs through language-model code revision and local parameter search. The full procedure improves overall success over the reported references but trails PPO on commanded setdown. Ablations support diagnostic feedback and parameter search on the main benchmark, with mixed mechanism effects and inconsistent search benefits under combined disturbances. These simulation results establish neither equal-compute superiority nor real-system effectiveness.

AI USE STATEMENT

Generative AI assisted with polishing the manuscript's language and presentation. It supported literature retrieval and the discovery of related work. It helped draft sections of the paper. It also assisted in generating the synthetic benchmark task configurations described in this paper.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the task split, policy interface, optimization procedure, and evaluation boundaries. Appendix A identifies the experimental records used in this draft and distinguishes completed evaluations from incomplete runs.

REFERENCES

Godwyll Aikins, Mawaba Pascal Dao, Koboyo Josias Moukpe, Thomas C. Eskridge, and Kim-Doang Nguyen. LEVIOSA: Natural Language-Based Uncrewed Aerial Vehicle Trajectory Generation. *Electronics*, 13(22):4508, 2024. doi: 10.3390/electronics13224508. URL https://doi.org/10.3390/electronics13224508.

Kaylee Burns, Ajinkya Jain, Keegan Go, Fei Xia, Michael Stark, Stefan Schaal, and Karol Hausman. GenCHiP: Generating robot policy code for high-precision and contact-rich manipulation tasks, 2024. URL https://arxiv.org/abs/2404.06645.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the Next AutoDiff: Generative Optimization with Rich Feedback, Execution Traces, and LLMs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2287. URL https://papers.nips.cc/paper_files/paper/2024/hash/83ba7056bce2c3c3c27e17397cf3e1f0-Abstract-Conference.html.

Chenggang Cui, Jiaming Liu, Peifeng Hui, Pengfeng Lin, and Chuanlin Zhang. GenControl: Generative AI-Driven Autonomous Design of Control Algorithms, 2025. URL https://arxiv.org/abs/2506.12554v3.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your Coding Agent is Secretly a Roboticist, 2026. URL https://arxiv.org/abs/2606.16458.

Qinglong Hu, Xialiang Tong, Mingxuan Yuan, Fei Liu, Zhichao Lu, and Qingfu Zhang. Multimodal LLM-assisted Evolutionary Search for Programmatic Control Policies, 2025. URL https://arxiv.org/abs/2508.05433.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models, 2023. URL https://arxiv.org/abs/2307.05973.

Wenkang Ji, Huaben Chen, Mingyang Chen, Guobin Zhu, Lufeng Xu, Roderich Groß, Rui Zhou, Ming Cao, and Shiyu Zhao. GenSwarm: Scalable Multi-Robot Code-Policy Generation and Deployment via Language Models, 2025. URL https://arxiv.org/abs/2503.23875.

Aoran Jiao, Tanmay P. Patel, Sanjmi Khurana, Anna-Mariya Korol, Lukas Brunke, Vivek K. Adajania, Utku Culha, Siqi Zhou, and Angela P. Schoellig. Swarm-GPT: Combining Large Language Models with Safe Motion Planning for Robot Choreography Design, 2023. URL https://arxiv.org/abs/2312.01059.

Shyam Sundar Kannan, Vishnunandan L. N. Venkatesh, and Byung-Cheol Min. SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models, 2023. URL https://arxiv.org/abs/2309.10062.

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, 2024. URL https://arxiv.org/abs/2406.09246v3.

Vaishak Kumar. Act-Observe-Rewrite: Multimodal Coding Agents as In-Context Policy Learners for Robot Manipulation, 2026. URL https://arxiv.org/abs/2603.04466.

Peihan Li and Lifeng Zhou. LLM-Flock: Decentralized Multi-Robot Flocking via Large Language Models and Influence-Based Consensus, 2025. URL https://arxiv.org/abs/2505.06513.

Zhaoxin Li, Rongrong Qian, Yuan Qi, Chaofan Wang, and Hao Su. Intent-Driven Cooperative Control of UAV Swarms: An LLM-Based Approach. *Applied Sciences*, 16(7):3297, 2026. doi: 10.3390/app16073297. URL https://doi.org/10.3390/app16073297.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. URL https://arxiv.org/abs/2209.07753.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction, 2023. URL https://arxiv.org/abs/2306.15724.

Mingrui Lv, Hangzhi Liu, Zhi Luo, Hongjie Zhang, and Jie Ou. PolicyEvolve: Evolving programmatic policies by LLMs for multi-player games via population-based training, 2025. URL https://arxiv.org/abs/2509.06053.

Artem Lykov, Sausar Karaf, Mikhail Martynov, Valerii Serpiva, Aleksey Fedoseev, Mikhail Konenkov, and Dzmitry Tsetserukou. FlockGPT: Guiding UAV Flocking with Linguistic Orchestration, 2024. URL https://arxiv.org/abs/2405.05872.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2023. URL https://arxiv.org/abs/2310.12931.

Martin Schuck, Dinushka Orrin Dahanaggamaarachchi, Ben Sprenger, Vedant Vyas, Siqi Zhou, and Angela P. Schoellig. SwarmGPT: Combining Large Language Models with Safe Motion Planning for Drone Swarm Choreography, 2024. URL https://arxiv.org/abs/2412.08428.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Denis Shcherba, Eckart Cobo-Briesewitz, Cornelius V. Braun, and Marc Toussaint. Meta-Optimization and Program Search using Language Models for Task and Motion Planning. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 5339–5361. PMLR, 2025. URL https://proceedings.mlr.press/v305/shcherba25a.html.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models, 2022. URL https://arxiv.org/abs/2209.11302.

Volker Strobel, Marco Dorigo, and Mario Fritz. LLM2Swarm: Robot Swarms that Responsively Reason, Plan, and Collaborate through LLMs, 2024. URL https://arxiv.org/abs/2410.11387.

Alkis Sygkounas, Victor Aregbede, Amy Loutfi, and Andreas Persson. MEMENTO: Memory-Guided Memetic Code-as-Policy Evolution, 2026. URL https://arxiv.org/abs/2607.22832.

Wenhao Wang, Yanyan Li, Long Jiao, and Jiawei Yuan. Large Language Model-Driven Closed-Loop UAV Operation with Semantic Observations, 2025. URL https://arxiv.org/abs/2507.01930.

Zhilin Wang, Han Song, Runzhe Zhan, Jusen Du, Jiacheng Chen, Tianle Li, Qingyu Yin, Yulun Wu, Zhennan Shen, Tong Zhu, Yanshu Li, Guanjie Chen, Derek F. Wong, Yafu Li, Yu Cheng, and Yang Yang. EvoPolicyGym: Evaluating Autonomous Policy Evolution in Interactive Environments, 2026. URL https://arxiv.org/abs/2607.02440.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning, 2023. URL https://arxiv.org/abs/2309.11489.

Chenduo Ying, Linkang Du, Peng Cheng, and Yuanchao Shu. RoboInspector: Unveiling the unreliability of policy code for LLM-enabled robotic manipulation, 2025. URL https://arxiv.org/abs/2508.21378.

Deyu Zhang, Xicheng Zhang, Jiahao Li, Tingting Long, Xunhua Dai, Yongjian Fu, Jinrui Zhang, Ju Ren, and Yaoxue Zhang. LLM-Driven Self-Refinement for Embodied Drone Task Planning, 2025. URL https://arxiv.org/abs/2508.15501.

Weixu Zhu, Marco Dorigo, and Mary Katherine Heinrich. Online automatic code generation for robot swarms: LLMs and self-organizing hierarchy, 2025. URL https://arxiv.org/abs/2510.04774. Workshop abstract, MACRAI at IROS 2025.

APPENDIX

A IMPLEMENTATION AND EXPERIMENTAL SETTINGS

The reported comparisons use completed final evaluations frozen at 23:44 EDT on 24 September 2026. The following details document the handwritten-start optimization, policy interface, and recorded computation. Runs lacking final evaluations are excluded.

Recorded code-optimization configuration. All LLM-based program-search methods in Tables 2–3 use `gpt-6-astra`, `medium` reasoning effort, and the `codex_cli` backend. The generated policy has no runtime model calls. Each scheduled training screen contains 125 instances, simulations last 20 s, and evaluation uses four simulation workers without GPU acceleration. A valid proposal exposes at most three active parameters and is evaluated with its default setting plus three clipped Gaussian samples when any parameter is active. Monetary cost is not populated in the records and is not treated as zero.

PPO selection and interaction counts. The PPO result in Table 2 uses the checkpoint at update 500, selected using the training monitor without validation or test feedback. Its recorded training count is 24,880,000 agent steps. Planned training and monitoring episode budgets appear in Table 8. Agent steps are not equivalent to complete team episodes or program-revision rounds.

Policy interface and implementation details. Initial memory is $m_{i,0} = \mathrm{Init}_P(\xi_i)$, where ξ_i is the permitted initialization context for drone i. The shared program implements `decide(view, mem, theta)` with private per-drone memory and an initialization context. Observations include vehicle state, payload-relative motion, goal and attachment positions, cable direction and tension, payload tilt, nearby drones and obstacles, room distances, and status flags. Initialization provides team size, payload mass and geometry, cable properties, and control limits. Inputs are derived from simulator state, apart from explicitly injected observation faults; decentralized execution does not imply realistic onboard sensing. Programs cannot access scene identifiers, future events, files, or simulator internals. The fixed command filter modifies velocities near other drones and room boundaries; it does not plan around obstacles or provide a formal safety guarantee.

Score definition and equivalence. The implementation computes category-weighted success. Let B_b be the instances of category b in evaluation set B. The score is

$$S_B(Q) = \sum_{b=1}^{3} \frac{w_b}{|B_b|} \sum_{c \in B_b} y_c(Q), \qquad w = (0.872, 0.064, 0.064),$$

where the categories are standard transport, emergency setdown, and stress transport. Scheduled counts $(109, 8, 8)$ give $w_b/|B_b| = 1/125$ for each category, yielding the pooled success rate in Eq. 2. The same proportions hold in the full training, validation, and test splits, so this equivalence holds there as well. These weights do not balance categories; for a differently proportioned evaluation set, the weighted and pooled scores need not coincide.

Proposal checking and paired selection details. Malformed proposals receive interface or import error feedback for up to three generation attempts. Only valid candidates enter the screen comparison; without a valid candidate there is no policy replacement. These generation attempts are distinct from the three parameter samples; model-service and evaluation-infrastructure failures are recorded separately. Parameter search evaluates the default first, followed by three clipped Gaussian samples when active parameters exist; it uses neither gradients nor covariance adaptation. Rotating panels make within-round comparisons paired, but scores from different rounds are not a fixed-panel learning curve. The final program is frozen before validation and test evaluation. Source modules, parameter settings, reports, and parent relationships are retained for accepted and rejected proposals.

For the secondary metric, let $d_c(Q)$ indicate successful delivery under the fixed transport verdict, excluding emergency setdown. In contrast, $y_c(Q)$ counts success under each instance's mission verdict, including emergency setdown. Let $T_c(Q)$ be the episode-level worst-drone peak commanded-

thrust fraction, measured with the recorded 0.5 s moving window. Define common successful deliveries and, when this set is nonempty, mean paired thrust reduction by

$$\mathcal{C}_B(Q', Q) = \{c \in B : d_c(Q') = d_c(Q) = 1\},$$
$$\Delta_T(Q', Q; B) = \frac{1}{|\mathcal{C}_B(Q', Q)|} \sum_{c \in \mathcal{C}_B(Q', Q)} [T_c(Q) - T_c(Q')].$$

The comparison BETTER(Q', Q) on B, also written $Q' \succ_B Q$, holds exactly when

$$S_B(Q') > S_B(Q)$$
$$\text{or } \big[S_B(Q') = S_B(Q),\ |\mathcal{C}_B(Q', Q)| > 0,\ \Delta_T(Q', Q; B) \geq 0.002\big].$$

The threshold is an absolute reduction in thrust fraction (0.2 percentage points). A success tie with no common successful delivery cannot establish improvement; all comparisons failing this rule retain the current comparison incumbent.

Limits of sequential screen selection. The pair-dependent common-delivery set can make $\succ_B$ nontransitive. For example, consider three transport instances in the same category and three policies with thrust metrics $A = (0.60, 0.40, \text{fail})$, $B = (\text{fail}, 0.30, 0.40)$, and $C = (0.70, \text{fail}, 0.30)$. Embed these instances in a panel where all three policies fail the remaining instances. All have the same score, yet $B \succ A$, $C \succ B$, and $A \succ C$, each on a different common-success instance. With incumbent A, default B, and subsequent sample C, sequential selection retains C and the final comparison rejects it, although B would have passed against A. This counterexample disproves a general ordering guarantee; it does not demonstrate an actual event in these runs.

A possible alternative is to retain the set of all evaluated candidates that pass directly against the incumbent, $\mathcal{E}_k = \{Q \in \mathcal{Q}_k : Q \succ_{B_k} Q_k\}$, where $\mathcal{Q}_k$ is the evaluated candidate pool. If $\mathcal{E}_k$ is nonempty, selection must return a member using a prespecified tie-breaking rule; otherwise the incumbent is retained. This prevents the illustrated loss of an available acceptable candidate. A candidate-order-independent ranking additionally requires a common evaluation criterion and deterministic tie handling. This alternative was not used to obtain the reported results; assessing its effect requires a separate implementation and evaluation.

A.1 Additional Experimental Configuration

The tables use the frozen, audited evaluation reports identified in the accompanying source manifest. Program search combines pretrained-language-model proposals with local Gaussian parameter search; the no-external-search variant evaluates defaults only. Table 8 collects numerical settings and evaluation sizes. Recorded optimization costs and their accounting conventions appear in Table 5 and the corresponding main-text discussion.

Reference-policy definitions. Handwritten control is the common initial program. Best-of-N uses $N = 16$ independent program proposals from that fixed handwritten start, without passing execution outcomes or prior revisions to subsequent code generation. Each valid proposal retains local parameter evaluation: its defaults and, when parameters are active, up to three clipped Gaussian samples. Thus, N counts program proposals, not all parameter candidates. Candidate selection uses the declared success-first comparison and paired-thrust tie rule on a fixed 125-instance training panel throughout Best-of-N; the iterative program-search variants rotate training panels. Best-of-N therefore differs from the full procedure in both revision feedback and panel scheduling. One-shot uses the predesignated first valid Best-of-N program at its default parameters, selected before observing its evaluation outcomes. PPO is the separately trained velocity-action reference whose checkpoint selection is described in Appendix A.

The MLES-inspired text adaptation was stopped after repeated candidate timeouts. It uses textual execution evidence rather than reproducing the original multimodal method. Its failure records are retained, without imputing task-success results.

Table 8: Experimental settings for the handwritten-start comparisons.

Setting / quantity	Value
Program-search proposal budget	16
Training-screen size	125
Test cases per frozen policy	250
Fresh PPO training agent steps	24,880,000
Fresh PPO planned training / monitor episodes	12,500 / 2,500

A.1.1 Fixed-policy intervention definitions

Tables 4 and 6 intervene on the frozen full program; Full denotes the unchanged training output, not a policy selected after observing these ablations. The executed policy diffs define the following operations, with parameter defaults and all unlisted code unchanged.

Near-goal braking. Zero only the horizontal payload-velocity braking vector in the intact-cable CARRY branch; preserve its activation calculation and the other feedback terms.

Carry vertical damping. Zero the complete applied vertical-damping term in that branch, including both attachment-velocity damping and its vertical-reference correction; retain vertical tracking, levelling, slack handling, and setdown control.

Attachment-motion feedback. Compute the attachment-motion estimate normally, then zero the vector supplied to its feedback consumers. Estimator memory updates and center-of-mass velocity feedback remain active. This changes shared inputs to horizontal swing control, carry damping, and pre-contact setdown control.

Local obstacle bypass. Zero only the tangential addition to preferred velocity; retain obstacle sensing, clearance activation, constraints, projection, braking, and the fixed command filter. This intervention does not remove all obstacle avoidance.

Velocity reanchoring. Zero the tracking-error correction in the transport slew routine; retain tracking-error computation, speed and slew bounds, and the separate setdown branch.

The double removal in Table 6 composes the carry-damping and attachment-feedback edits exactly. Their shared signal use means these are implementation-level interventions, not independent physical contributions. All interventions use paired cases and fixed verdicts without retraining or retuning.

B Benchmark construction and dimension-wise allocation

What counts as a case. A case is one frozen simulator configuration with an explicit scene, event schedule, task objective, and random seed, rather than a demonstration or a policy rollout. A scenario family fixes team size, payload shape, attachment arrangement, layout, event-type combination, requested load setting, and timing mode. Each family contributes five distinct training settings, one validation setting, and two test settings: $125 \times (5+1+2) = 1{,}000$ cases. The standard, commanded-setdown, and stress categories contain 109, 8, and 8 families. The 180 calibration candidates are separate and excluded below.

How to read the census. Table 9 reports counts recomputed from the frozen case index and checked against the split definition files. Blocks describe the same population from different dimensions and must not be added together. Each of the first six blocks partitions all 1,000 cases. Event-type rows count cases containing that event, including composite disturbances, so memberships overlap. Timing rows cover only the 448 multi-event cases. Family counts are not additional cases. Every row retains the 5:1:2 split ratio.

Table 9: **Dimension-wise census of the frozen benchmark.** A *scenario family* is a shared scenario template that fixes the task category, team size, payload shape, attachment arrangement, environment layout, event-type combination, requested load setting, and timing mode. Each family generates distinct cases across the training, validation, and test splits by varying concrete parameters and random seeds. The *Families* column counts templates satisfying each row, not extra cases or successful rollouts. The same family appears in different dimension blocks, so blocks must not be added together. Event-type memberships can overlap; timing rows include only multi-event families. Calibration candidates are excluded.

Setting / attribute	Families	Train	Val.	Test	Total
A. Task category					
Standard transport	109	545	109	218	872
Commanded emergency setdown	8	40	8	16	64
Stress transport	8	40	8	16	64
B. Team size					
2 drones	18	90	18	36	144
3 drones	18	90	18	36	144
4 drones	18	90	18	36	144
5 drones	18	90	18	36	144
6 drones	18	90	18	36	144
7 drones	18	90	18	36	144
8 drones	17	85	17	34	136
C. Payload shape					
Ordinary box	72	360	72	144	576
Flat payload	19	95	19	38	152
Elongated payload	19	95	19	38	152
Tall slender column	15	75	15	30	120
D. Attachment arrangement					
Perimeter	81	405	81	162	648
Line	16	80	16	32	128
Uneven	16	80	16	32	128
Clustered	12	60	12	24	96
E. Environment layout					
Open	38	190	38	76	304
Blocking obstacle	29	145	29	58	232
Pillars	29	145	29	58	232
Corridor	29	145	29	58	232
F. Number of events per case					
0 (no events)	25	125	25	50	200
1	44	220	44	88	352
2	28	140	28	56	224
3	20	100	20	40	160
4	8	40	8	16	64
G. Event type (overlapping membership)					
Steady wind `wind`	11	55	11	22	88
Gust `gust`	15	75	15	30	120
One-sided gust `gust_side`	11	55	11	22	88
Payload impulse `impulse`	12	60	12	24	96
Sustained payload push `push`	11	55	11	22	88
Payload snag `snag`	11	55	11	22	88
Single cable cut `dropout`	13	65	13	26	104
Two cable cuts `dropout2`	13	65	13	26	104
Drone failure, cable attached `dead_drone`	13	65	13	26	104
Reduced motor thrust `weak_motor`	14	70	14	28	112
Position observation bias `pos_bias`	12	60	12	24	96
Missed policy decisions `frame_drop`	12	60	12	24	96
Payload mass reduction `mass_drop`	11	55	11	22	88
Center-of-mass shift `com_shift`	11	55	11	22	88
Goal shift `goal_shift`	11	55	11	22	88
Moving obstacle `moving_obstacle`	11	55	11	22	88
H. Timing mode (multi-event cases only)					
Overlap mode	25	125	25	50	200
Sequential mode	31	155	31	62	248
All benchmark cases	125	625	125	250	1,000

Within-family variation and physical settings. All cases use a $14 \times 14 \times 6$ m room, 20 s episodes, and 1 m cables. Reference payload dimensions (length, width, height) are $(0.70, 0.55, 0.50)$ m for boxes, $(1.10, 0.80, 0.08)$ m for flat payloads, $(1.20, 0.25, 0.25)$ m for elongated payloads, and $(0.35, 0.35, 1.00)$ m for tall columns. Each dimension varies independently by up to 4%. Initial

horizontal position varies by up to 0.08 m per axis, with the nominal goal 4 m away horizontally at a height of 1 m. Center-of-mass offsets, event directions, magnitudes, durations, and affected drones are also instantiated and stored. Initial payload masses span 0.02912–0.38080 kg; they are not sampled uniformly over that interval. Standard-transport generation reduces mass in 96 cases to cap post-fault static total-thrust demand at 68%, below the 70% acceptance threshold.

For ordinary triggered events, a base onset is drawn from 6–8 s. Sequential mode adds 0.6 s per event index, whereas overlap mode shares the base onset. Steady wind starts at 0 s and moving obstacles at 1 s. Timing labels therefore do not imply identical onsets for every event or strictly non-overlapping effects. Standard transport is generated with a chronological static total-thrust ceiling of 68% and checked against a 70% acceptance threshold; stress transport lies above 70% and at or below 95%. Setdown cases use an event-dependent capacity setting and a different objective.

Coverage design and an example. This is a structured allocation, not a Cartesian product. The 109 standard families comprise 21 event-free, 32 single-event, 28 two-event, 20 three-event, and 8 four-event families. The 8 setdown families each contain one fault. Of the 8 stress families, 4 have no event and 4 contain a gust. Every split covers all 28 team-size–shape combinations and all 20 layout–event-count combinations. Setdown and stress families use perimeter attachments only; all setdown families use open layouts. Wind variants are mutually exclusive within a family, at most one cable/motor fault type is selected, and goal shifts are not paired with moving obstacles. Marginal coverage thus does not establish exhaustive coverage of interactions.

For example, family `v3_f000` uses two drones, an ordinary box, perimeter attachments, a corridor, and a sustained push combined with reduced motor thrust. It contributes five training, one validation, and two test cases with separately instantiated settings. Exact configurations and seeds do not overlap across splits, but family definitions do. The evaluation measures generalization to new settings within known families. Static input checks do not certify dynamic feasibility, and the census does not count successful policy outcomes.

C Additional Qualitative Rollouts

The six cases below complement the six main-text rollouts without repeating their task instances or recorded trajectories. Each figure contains ten snapshots of a single recorded trajectory, read left to right across the first row and then the second. Sampling is denser around disturbances. The camera is fixed within each case, and every state is taken from its recorded timestamp without rerunning or interpolating the dynamics. Green cables are taut, red dashed cables are slack, and severed mechanical links are omitted. Gold denotes the transport target and the payload path; the original transport target is omitted for the emergency-setdown task. Vehicle glyphs do not encode body attitude, and obstacles are translucent schematic silhouettes.

Selection and provenance. The twelve examples were selected for scenario diversity and visual interpretability from 28 existing project videos. They are not a random sample and do not estimate a success rate. The main-text moving-goal, clutter, and heavy-load cases use the EXP013 repaired controller; the tall- and slender-cargo cases use the EXP015 trained controller; and the collinear-attachment case uses EXP011 release 05. In this appendix, the four-event case uses EXP013; the cable-cut, blocking-pillar, flat-cargo, and clustered-attachment cases use EXP015; and the eight-drone case uses EXP011 release 05. The render manifest records task identifiers, seeds, source hashes, policy provenance, verdicts, and exact sample indices.

Geometry and verdicts. Where recorded rigid-body rotations are available, the cargo uses its recorded dimensions and orientation. Other cases show only the recorded cable-attachment surface, not a reconstructed solid payload. Passing a historical reach-and-hold verdict does not necessarily mean that the payload remains inside the goal tolerance at the final sample. The emergency-setdown example below also exposes a limitation of its recorded touchdown-based verdict.

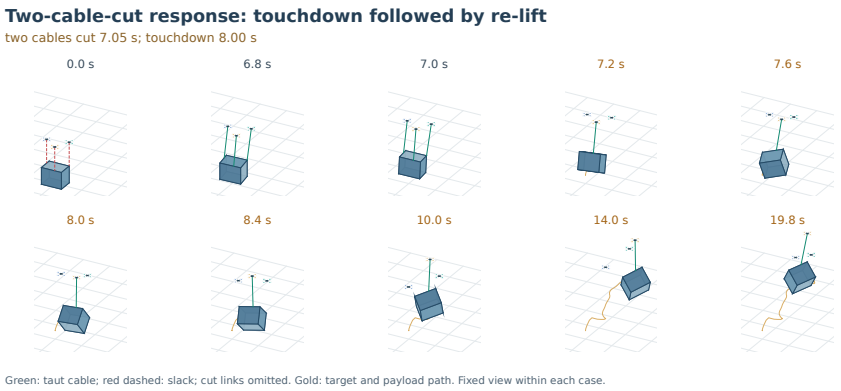

Figure 4: **Two-cable-cut response (EXP015; three drones).** The payload touches down after the cuts but later lifts again. It passes the historical touchdown-based verdict without demonstrating persistent safe rest. Snapshots include contact and subsequent motion; box geometry is recorded.

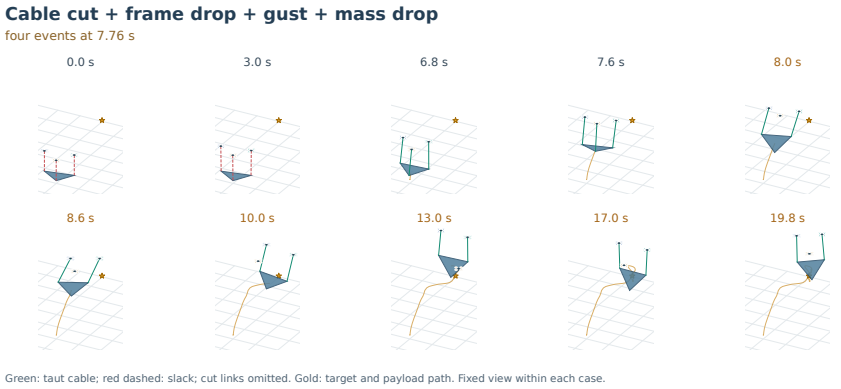

Figure 5: **Four simultaneous disturbances (EXP013; three drones).** Cable cut, frame drop, gust, and mass drop trigger transient recovery. The historical transport verdict passes, but the final payload position lies outside the goal tolerance. Only the recorded attachment surface is shown.

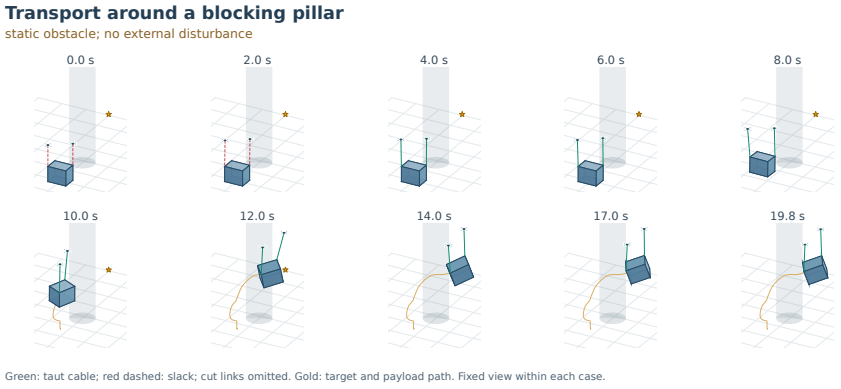

Figure 6: **Blocking-pillar transport (EXP015; two drones).** The controller completes transport around a static pillar without external disturbance. Cargo geometry and orientation are recorded; the pillar is translucent for visibility.

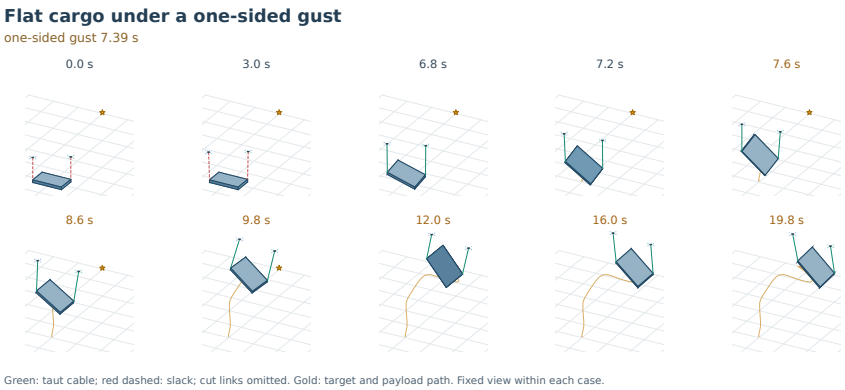

Figure 7: **Flat cargo under a one-sided gust (EXP015; two drones).** The controller completes the recorded transport task with a flatter payload and asymmetric wind exposure. Cargo dimensions and orientation are recorded.

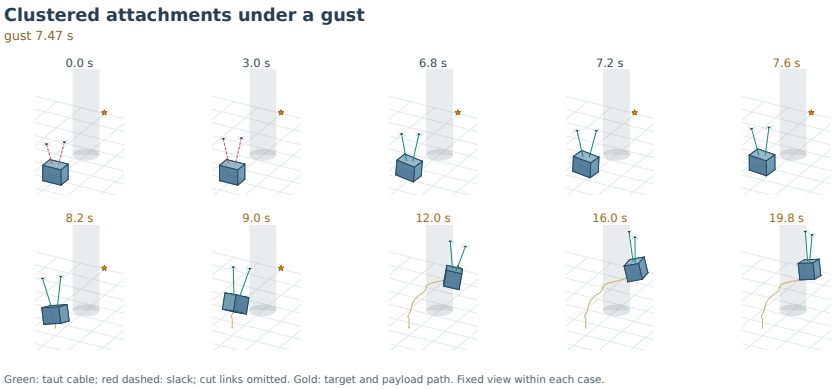

Figure 8: **Clustered attachments under a gust (EXP015; two drones).** The controller completes transport with clustered attachment points and a blocking pillar. This example varies attachment geometry; cargo geometry and orientation are recorded.

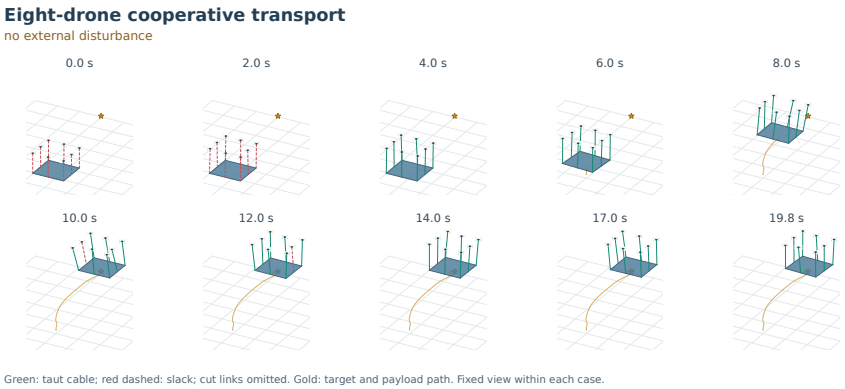

Figure 9: **Eight-drone transport (EXP011 release 05).** This undisturbed case extends the illustrated team-size range and passes both the historical transport verdict and terminal-stability audit. The recorded attachment surface represents the payload.