AEROSOP: BRIDGING TASK INTENT AND AAV EXECUTION THROUGH EXECUTABLE PROCEDURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) enable flexible mission planning for autonomous aerial vehicles (AAVs), but reliance on remote reasoning makes mission progress vulnerable to inference delays and communication disruptions. Meanwhile, action failures and changing observations can invalidate planned steps, making reliable continuation depend on execution feedback and updates consistent with the vehicle's current state. We introduce AeroSOP, a framework that decouples base-station reasoning from onboard execution through verifiable, incrementally updatable standard operating procedures (SOPs). The base station composes reusable skills into validated procedures, while an onboard runtime checks step contracts, verifies completion evidence, and maintains explicit execution state without task-level LLM inference. State-compatible patches introduce subsequent tasks at verified task-completion boundaries, preserving completed results and activating atomically after local compatibility and energy checks. Bounded waiting and return-feasibility monitoring regulate continuation under delayed updates and disturbances. Evaluation against eleven baselines across simulated urban missions shows that AeroSOP achieves 79.6% overall success and 97.00% inspection success, with no false completions or invalid proposals observed on inspection and search–confirmation tasks. Its patches match state-preserving full refresh at 91.0% useful continuation while reducing 95th-percentile switching latency from 6.8 to 4.7 s. Relative to remote skill-by-skill dispatch, AeroSOP reduces control traffic per mission by 75.5% and response latency at 24 AAVs by 79.7%.

1 INTRODUCTION

Large language model (LLM) agents offer a promising interface for specifying autonomous aerial vehicle (AAV) missions through natural language, allowing operators to express objectives and constraints without manually defining every execution step (Chen et al., 2024a; Vemprala et al., 2024). This capability is particularly relevant to continuous missions, where subsequent tasks may arrive during flight or depend on earlier outcomes (Buckman et al., 2019). For example, an instruction to inspect an area and investigate detected anomalies requires an initial survey followed by observations whose targets cannot be fully specified before takeoff. Executing such a request requires the mission intent to be grounded in available capabilities (ichter et al., 2023a), valid parameters, execution dependencies, and observable completion criteria (Liang et al., 2023; Singh et al., 2022; Liu et al., 2023). The AAV must also determine whether each operation remains applicable under its current state and how to respond when execution deviates from expectations (Tagliabue et al., 2024; Joublin et al., 2024). These requirements expose a semantic and procedural gap between expressing a mission objective and establishing an executable process that fulfills it (Huang et al., 2022).

Existing research provides important foundations for connecting language reasoning with robotic capabilities (Shi et al., 2025). SayCan grounds skill selection in task relevance and physical affordances (ichter et al., 2023b), while Code as Policies generates programs that compose perception and control interfaces (Liang et al., 2023). Building on these capabilities, continuous AAV missions require an execution interface that jointly addresses procedural validity, local autonomy, and adaptation to subsequent tasks (Ravichandran et al., 2025). Selecting callable skills alone does not establish the evidence required for mission completion or the permitted responses to partial failure (Duan et al., 2025; Zeng et al., 2026). Meanwhile, frequent task-level language reasoning introduces onboard computational demands, whereas reliance on successive base-station decisions makes mis-

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.

sion progress sensitive to communication delays and interruptions (Ling et al., 2026). Executing a complete procedure locally can separate task-level reasoning from immediate execution, but the procedure must remain adaptable as new requirements emerge. Subsequent tasks may change execution steps, dependencies, and completion conditions, requiring updates to account for retained results (Anwar et al., 2025), the latest vehicle state, and the energy needed for further execution and return. These requirements lead to the following central question:

> *How can high-level mission intent be translated into locally executable and verifiable procedures that can be updated to support continuous AAV missions?*

To answer this question, we introduce AeroSOP, a framework that uses standard operating procedures (SOPs) as an executable interface between base-station task reasoning and onboard physical execution. Each SOP organizes reusable local skills into a mission-oriented process with explicit capability bindings, typed parameters, execution conditions, completion evidence, and bounded exception handling. The base-station agent interprets task requirements and constructs SOPs with supporting validation tools, while an onboard runtime checks their contracts, maintains execution state, and invokes local skills. Perception, local planning, feedback control, and immediate safety responses remain onboard, allowing prescribed branches to be handled locally. To support continuous missions, AeroSOP introduces a next-task SOP patch that incrementally extends the installed procedure while preserving relevant execution state and completed results. A patch is activated at a declared task-completion boundary after local checks of its version, required capabilities, state compatibility, and remaining energy, including the estimated return requirement. Bounded waiting and independent safety monitoring govern continuation when updates are delayed, invalid, or no longer feasible. The main contributions of this work are summarized as follows:

- **An executable SOP interface for mission grounding.** We introduce a contract-based SOP representation that translates mission intent into locally executable procedures. By organizing reusable skills with explicit capability bindings, typed parameters, execution dependencies, completion criteria, and bounded exception handling, this representation supports both step-level execution checks and mission-level verification.
- **A reasoning–execution decoupled architecture.** We design a base-station–AAV architecture that places task-level language reasoning and SOP construction at the base station, while retaining perception, local planning, feedback control, and safety responses onboard. An onboard runtime maintains execution state and executes accepted procedures, allowing the AAV to handle prescribed branches without requiring a new base-station decision at every step.
- **A state-compatible patch mechanism for continuous missions.** We develop a next-task SOP patch mechanism that supports incremental procedure updates while preserving completed results and relevant execution state. Updates are activated at task-completion boundaries after local validation of version, capability, state, and energy compatibility. Bounded waiting and continuous return-feasibility monitoring govern task continuation when updates are delayed, invalid, or no longer feasible.

2 Related Work

Language-guided task planning and grounding. Language-guided planning grounds instructions through admissible action mapping (Huang et al., 2022), skill affordances (ichter et al., 2023b), and plan updates informed by environmental and execution feedback (Huang et al., 2023b; Song et al., 2023). SayPlan and VoxPoser extend grounding through hierarchical scene graphs with simulator feedback and language-conditioned 3D value maps, supporting task planning and closed-loop motion generation, respectively (Rana et al., 2023; Huang et al., 2023a). Building on these mechanisms for grounded planning and adaptation, our work packages local skills into executable mission procedures with explicit parameters, execution conditions, and completion evidence for onboard checking.

Executable programs and procedural validation. Code as Policies composes perception and control APIs into executable programs (Liang et al., 2023), while ProgPrompt incorporates precondition assertions and recovery (Singh et al., 2022). AutoTAMP translates instructions into signal tem-

poral logic for task and motion planning, with syntactic and semantic correction (Chen et al., 2024c). Behavior trees support modular, reactive execution and formal analysis (Colledanchise & Ögren, 2018), with BTGenBot exploring their generation and validation using lightweight LLMs (Izzo et al., 2024). HyCodePolicy uses execution traces and visual feedback to diagnose failures and repair programs (Liu et al., 2025). Building on these validation and recovery mechanisms, our SOP interface combines step contracts, mission-level completion checks, bounded failure responses, and procedure-update rules. Behavior trees or state machines can implement its control flow; our focus is execution and update semantics across mission boundaries.

Language-enabled AAV systems and continuous missions. ChatGPT for Robotics explores function libraries and interactive feedback for tasks including aerial navigation (Vemprala et al., 2024). TypeFly combines a remote LLM with an edge-hosted MiniSpec runtime for compact programs, streaming interpretation, and exception-triggered replanning (Chen et al., 2024a), while SwarmGPT couples language-based choreography with motion safety filtering (Schuck et al., 2025). Persistent-surveillance methods address energy-constrained routing and mobile recharging (Seyedi et al., 2019; Lin et al., 2022); HGAM coordinates mission and charging AAVs through heterogeneous graph observations and multi-agent reinforcement learning (Hu et al., 2025). Our interface connects base-station SOP construction and updates with onboard state management, contract checking, and skill execution under local safety supervision. Next-task patches preserve relevant completed results and activate at task-completion boundaries after version, capability, state, and energy checks. Energy admission and return monitoring constrain continuation, while bounded waiting handles unavailable or invalid updates.

3 METHOD

Figure 1 presents AeroSOP, which connects base-station reasoning with onboard execution through executable SOPs. The agent instantiates templates tested before deployment, and the onboard runtime executes local skills under explicit contracts and verifies completion. A locally validated patch then introduces the next task at a completion boundary, while control and immediate safety responses remain onboard.

3.1 SYSTEM MODEL AND PROBLEM FORMULATION

We consider an AAV equipped with local perception, planning, control, and safety monitoring, together with versioned skills, cached SOP definitions, and an initial validated procedure. Tasks may arrive during flight or depend on earlier results. Each task j specifies a goal g_j, constraints $\mathcal{C}_j$, a completion predicate ϕ_j, and a value w_j. The base station selects tasks and constructs procedures using the latest reported state $\hat{\mathbf{z}}$, while the AAV checks them against its current state $\mathbf{z}(t)$, including energy, capabilities, execution progress, and retained results. Because reports and updates can be delayed, these decisions use only currently available information. Over horizon T, we seek a causal policy that maximizes the value of validly completed tasks:

$$\max_{\pi \in \Pi_{\text{causal}}} \mathbb{E}_{\pi} \left[\sum_{j \in \mathcal{J}(T)} w_j y_j(T) \right], \tag{1}$$

where $\mathcal{J}(T)$ contains released tasks and $y_j(T)$ indicates completion by T. The policy must satisfy capability, contract, task, energy, and update-compatibility constraints. These requirements guide both how an SOP is constructed and when its execution can begin or be declared complete.

3.2 EXECUTABLE SOP REPRESENTATION AND CONSTRUCTION

AeroSOP grounds a task in the skills already installed on the AAV. Each SOP $\mathcal{P}_j$ combines an acyclic process graph, versioned skill bindings, typed parameters, execution contracts, and a task-level completion predicate. A node contract $\kappa_i = (p_i, q_i, \tau_i, \rho_i, f_i)$ specifies its precondition, outcome check, per-attempt timeout, retry budget, and approved failure response. Dependencies connect required operations and their outputs, while bounded retry subprocedures remain local to nodes. The runtime uses these contracts to govern dispatch and verification. Templates are tested offline under nominal,

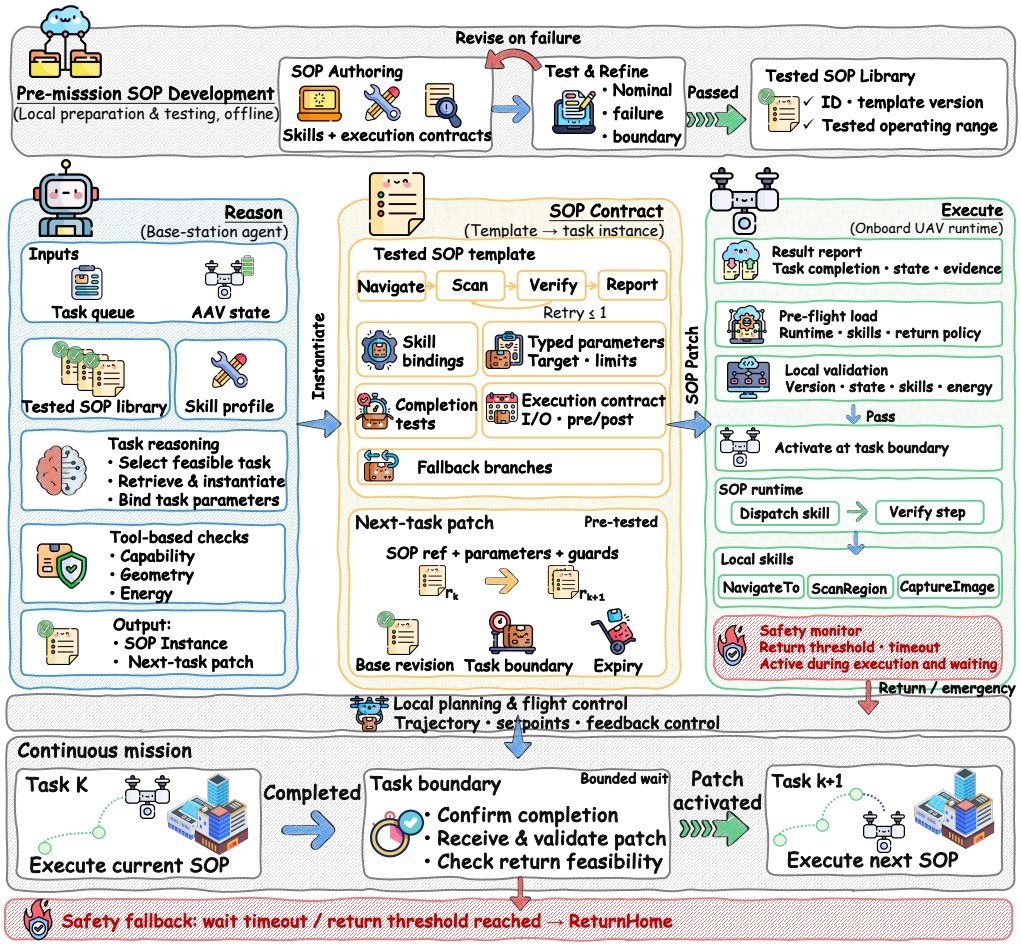


Figure 1: Overview of AeroSOP: executable SOPs connect base-station reasoning with onboard execution, while validated patches support continuous AAV missions.

failure, and boundary conditions. For an incoming task, the agent retrieves applicable templates, binds inputs, and composes their dependencies into a candidate. The resulting instance is checked for interface compatibility, parameter validity, graph structure, contract completeness, current-state feasibility, and energy. Writing these binary checks as c_a, indexed by $\mathcal{A}_{\text{check}}$, acceptance requires

$$V(\mathcal{P}, \mathbf{z}) = \prod_{a \in \mathcal{A}_{\text{check}}} c_a(\mathcal{P}, \mathbf{z}) = 1. \tag{2}$$

Failed checks guide revision within a fixed budget; unresolved violations cause rejection. Validation therefore addresses the composed instance, using the representation, grounding, and construction rules detailed in Appendix A.1.

3.3 Reasoning–Execution Decoupling

Once a procedure is committed, its contracts govern onboard progress without requiring a language-model decision at every step. The runtime dispatches a node when its selected-branch dependencies, inputs, and precondition p_i are satisfied, then evaluates q_i before releasing successors. Failures invoke bounded recovery with renewed precondition, time, and energy checks; exhausted recovery terminates the task and triggers a preloaded fallback. Each attempt contributes evidence to an execution record $\mathcal{R}_j(t)$ associated with task and invocation identifiers.

These local checks establish the evidence needed to assess the task as a whole. Let $s_j^{\text{term}}(t)$ denote its terminal status, and let Sat evaluate the declared task constraints. Completion is verified through

$$\chi_j(t) = \mathbf{1}[s_j^{\text{term}}(t) = \text{success}]\, \phi_j(\mathcal{R}_j(t)) \cdot \text{Sat}(\mathcal{C}_j; \mathcal{R}_j(t)), \tag{3}$$



Figure 2: Representative sparse, medium, and dense urban mission layouts with 14, 24, and 40 buildings, respectively. Each includes three base stations, task targets, a temporary no-fly zone, and a link-shadow region.

where ϕ_j and Sat are binary predicates, giving $y_j(T) = \mathbf{1}[\exists t \le T : \chi_j(t) = 1]$. An inspection task requiring delivery therefore remains incomplete while its observations are cached but undelivered. A verified completion report prompts the base station to rank eligible tasks by decreasing value, breaking ties by earlier deadlines, and construct candidates until one passes validation or the decision budget expires. Throughout this process, local control and safety monitoring remain active. Appendix A.2 specifies dispatch and recovery in detail.

3.4 State-Compatible Continuous Mission Chaining

A completion report can release further work while the AAV remains in flight. To connect this work to the current execution state, the runtime records a unique boundary event and opens a bounded waiting window. The next-task patch carries the new procedure and its input bindings, together with the target AAV, required base revision, boundary identifier, and expiry. The AAV reconstructs the candidate $\mathcal{P}^{\Delta}$ and reassesses its feasibility locally. Activation is restricted to an open waiting window with no active skill or latched safety response:

$$A_{\Delta}(t) = C_{\Delta}(t)\, V(\mathcal{P}^{\Delta}, \mathbf{z}(t)) \cdot \mathbf{1}[W(t) = 1,\, \mathcal{U}(t) = \varnothing,\, \sigma(t) = 0] = 1, \tag{4}$$

where C_{Δ} checks message compatibility and identifier uniqueness, W indicates an open waiting window, $\mathcal{U}$ active skills, and σ a latched safety response. A serialized commit rechecks this guard and atomically installs the procedure, advances its revision, and records activation while retaining completed results. Rejected patches preserve the installed procedure, and duplicates receive their recorded status. Appendix A.3 formalizes the patch and update protocol. Continuation must also preserve the ability to return. Waiting can reduce available energy after a candidate was constructed, so admission is repeated using current energy and conservative estimates for remaining waiting, travel, task execution, and return. Including a safety reserve, the candidate must satisfy

$$E_{\mathrm{rem}}(t) \ge \overline{E}_{\mathrm{wait}}(t) + \overline{E}_{\mathrm{transfer},j}(t) + \overline{E}_{\mathrm{task},j}(t) + \overline{E}_{\mathrm{return},j}(t) + E_{\mathrm{reserve}}. \tag{5}$$

The task estimate covers permitted branches and retries. During execution and waiting, reaching the return threshold or exhausting the waiting budget triggers a latched fallback. Link loss allows an accepted SOP to continue while its contracts and safety conditions hold; a further task requires a valid patch. Appendix A.4 states the energy assumptions, and Algorithm 1 in Appendix A.6 gives the event handlers.

4 Experiments

In this section, we evaluate AeroSOP to answer four research questions: **(RQ1)** How does AeroSOP compare with representative baselines in end-to-end task completion and mission productivity under common execution constraints? **(RQ2)** How do pre-tested SOPs, execution contracts, and online verification contribute to reliable task execution? **(RQ3)** How do state-compatible SOP patches support continuous mission execution and return feasibility under changing task demands and operational disturbances? **(RQ4)** How efficiently does AeroSOP scale multi-AAV execution while keeping task-level reasoning offboard?

Table 1: Task success and execution reliability. Values below success rates are interval bounds; values below error rates are event counts n/N. I+SC denotes inspection and search–confirmation. Bold marks the best point estimates in each column, including ties.

Method	Success rate (%) ↑				I+SC error rate (%) ↓	
	Overall	Inspection	Search–Conf.	Compound	False compl.	Invalid prop.
Inner Monologue	78.92 [77.33, 80.50]	87.50 [84.25, 90.75]	92.25 [90.00, 94.50]	57.00 [53.50, 60.25]	5.29 38/719	13.50 108/800
Code as Policies	41.92 [39.08, 44.75]	66.00 [62.25, 69.75]	28.75 [23.00, 34.50]	31.00 [27.00, 35.00]	19.16 82/428	19.88 159/800
ProgPrompt	23.08 [20.75, 25.50]	30.75 [27.00, 35.00]	21.25 [17.50, 25.25]	17.25 [13.25, 21.75]	20.23 52/257	17.00 136/800
BETR-XP-LLM	62.25 [60.92, 63.58]	34.25 [31.75, 36.75]	98.50 [96.25, 100.00]	54.00 [52.25, 56.00]	**0.00** 0/531	32.62 261/800
Agent4Drone	78.33 [76.67, 80.00]	78.50 [75.00, 81.75]	93.50 [89.75, 96.75]	**63.00** [60.25, 65.75]	4.89 27/552	29.00 232/800
LLM-OBTEA	59.75 [58.58, 61.00]	33.25 [30.50, 36.25]	98.50 [96.25, 100.00]	47.50 [46.25, 49.00]	**0.00** 0/527	33.38 267/800
AdaPlanner	69.83 [66.92, 72.50]	81.00 [77.00, 84.75]	90.50 [85.75, 94.50]	38.00 [35.00, 41.00]	**0.00** 0/686	3.25 26/800
LLM-DP	62.00 [61.08, 62.92]	33.25 [30.50, 36.00]	**100.00** [100.00, 100.00]	52.75 [51.25, 54.75]	**0.00** 0/533	33.38 267/800
SayCan	60.08 [58.83, 61.42]	46.75 [43.99, 49.75]	98.50 [97.25, 99.50]	35.00 [32.75, 37.75]	12.82 85/663	1.75 14/800
ReAct	51.83 [50.25, 53.50]	59.00 [54.50, 63.01]	51.25 [47.00, 55.75]	45.25 [41.25, 49.25]	17.17 80/466	23.75 190/800
AUSPEX	61.00 [60.17, 61.92]	33.25 [30.50, 36.00]	99.50 [98.75, 100.00]	50.25 [50.00, 50.75]	**0.00** 0/531	33.38 267/800
AeroSOP	**79.67** [76.00, 83.08]	**97.00** [94.00, 99.25]	94.50 [90.50, 97.75]	47.50 [41.25, 54.00]	**0.00** 0/766	**0.00** 0/800

4.1 Experimental Setup

Task scenarios. We evaluate AeroSOP in sparse, medium-density, and dense urban layouts with 14, 24, and 40 buildings, respectively, as illustrated in Figure 2, with each layout containing three base stations. Static workloads provide controls, while dynamic missions combine continuous inspection and search–confirmation. Inspection involves scanning designated regions and collecting evidence. Search–confirmation requires target detection, close-up confirmation linked to the parent search task, and report delivery, with data buffered locally during communication outages. Each task specifies a 3D location, deadline, priority, required skills, and estimated energy cost. Online task arrivals and cancellations, link changes, and skill or energy disturbances test continuous mission chaining. Shared EventTapes and action-keyed randomness make external events and stochastic outcomes reproducible, while online policies access only information available at each decision point.

Baselines. We compare AeroSOP with eleven representative baselines, including LLM-based task planning methods Inner Monologue (Huang et al., 2023b), SayCan (ichter et al., 2023a), ReAct (Yao et al., 2023), and AdaPlanner (Sun et al., 2023); program generation methods Code as Policies (Liang et al., 2023) and ProgPrompt (Singh et al., 2022); symbolic and behavior-tree planning methods LLM-DP (Dagan et al., 2024), BETR-XP-LLM (Styrud et al., 2025), and LLM-OBTEA (Chen et al., 2024b); and AAV-oriented frameworks Agent4Drone (Zhang et al., 2026) and AUSPEX (Döschl et al., 2025). For AUSPEX, we add a language interface and denote the resulting baseline as AUSPEX+LLM-interface. All baselines are adapted to the common AAV environment with a shared skill interface and safety layer, and all LLM inference is performed at the base station.

Experimental Platform. All experiments were conducted under a unified hardware and software environment to ensure fair comparison and reproducibility of the reported latency, energy, and execution results. Specifically, the evaluation was performed on a server running Ubuntu 22.04.5 LTS with Python 3.12.12, PyTorch 2.8.0+cu128, and CUDA 12.8. The platform was equipped with 8 NVIDIA H100 80GB HBM3 GPUs, dual Intel Xeon Platinum 8562Y+ CPUs, and 1.0 TiB system memory.

4.2 Overall Performance (RQ1)

Observation 1: AeroSOP achieves the highest overall task-completion rate while maintaining strong execution reliability. Under the common evaluation setting, AeroSOP achieves the high-

est observed overall SR at 79.67%, compared with 78.92% for Inner Monologue and 78.33% for Agent4Drone. Its clearest advantage appears in inspection, where it reaches 97.00% SR, exceeding the next-best method by 9.50 percentage points. AeroSOP is also the only evaluated method with both zero observed false completions (0/766) and zero invalid proposals (0/800) on I+SC tasks, indicating that its task-completion gains are accompanied by reliable execution decisions. The gains are not uniform across task types: AeroSOP reaches 94.50% on search–confirmation and 47.50% on compound missions, below the strongest task-specific baselines. Overall, AeroSOP provides the strongest observed balance between end-to-end task completion and execution reliability, with compound missions remaining the main performance gap.

Observation 2: AeroSOP delivers the highest mission productivity without sacrificing execution reliability. Under the same execution setting, AeroSOP achieves the highest externally accepted root-task throughput at 47.80 tasks/h, compared with 47.35 tasks/h for Inner Monologue and 47.00 tasks/h for Agent4Drone. Because throughput counts externally accepted root tasks, it measures productive mission completion rather than raw action generation or execution frequency. Together with the I+SC results in Table 1, AeroSOP therefore combines the highest observed accepted-task output with zero observed false completions and invalid proposals. These results show that AeroSOP's execution safeguards do not trade mission productivity for reliability; the framework preserves reliable execution while sustaining the highest observed task throughput.

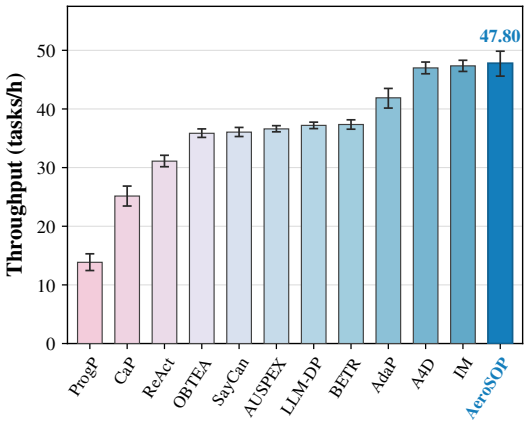


Figure 3: Externally accepted root-task throughput across three scenarios, with 95% bootstrap confidence intervals.

4.3 SOP Reliability Analysis (RQ2)

Table 2: Ablation study of AeroSOP on task success, completion reliability, and fault recovery. ID and Composition-OOD denote in-distribution and unseen task compositions, respectively. All values are percentages; ↑ and ↓ indicate better directions.

Variant	ID SR (%) ↑	Composition-OOD SR (%) ↑	False completion (%) ↓	Recoverable-fault SR (%) ↑
Full AeroSOP	**92.8**	**85.6**	**0.8**	**81.0**
Untested SOP Library	86.4	75.8	3.4	74.6
w/o Step Contract Checks	87.8	79.0	2.2	74.0
w/o Mission Verification	92.0	85.0	7.6	80.6
w/o Local Exception Handling	90.4	82.8	1.0	62.8

Observation 3: Pre-tested SOPs provide the reliability foundation for both in-distribution execution and unseen task compositions. Table 2 shows that replacing the pre-tested SOP library with untested procedures reduces ID SR from 92.8% to 86.4% and Composition-OOD SR from 85.6% to 75.8%, corresponding to drops of 6.4 and 9.8 percentage points, respectively. The larger degradation on unseen compositions indicates that pre-testing is particularly important when previously validated procedures are reused in new task combinations. Untested

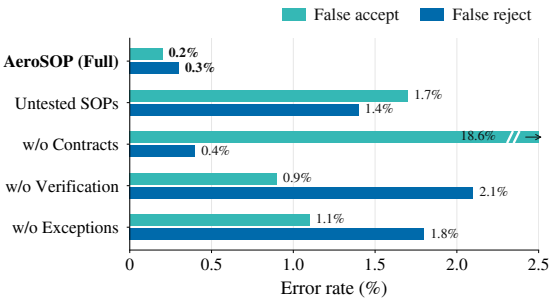


Figure 4: Step-level admission errors of AeroSOP and its ablations.

SOPs also increase false completion from 0.8% to 3.4% and reduce recoverable-fault SR from 81.0% to 74.6%, showing that their effect extends beyond nominal task success. These results demonstrate that offline SOP validation provides a reliable procedural basis before runtime safeguards are applied.

Observation 4: Execution contracts and online verification provide complementary safeguards for reliable execution. Full AeroSOP achieves only 0.2% false acceptance and 0.3% false rejection. Removing step contract checks raises false acceptance to 18.6%, showing their key role in preventing invalid execution. Removing mission verification increases false completion from 0.8% to 7.6%, despite little change in task success, highlighting the need for explicit completion evidence. Local exception handling complements these checks under execution faults, with its removal reducing recoverable-fault SR from 81.0% to 62.8%. Together, these results show that AeroSOP combines admission checking, completion verification, and local recovery to maintain reliable execution.

4.4 Continuous Execution and Robustness (RQ3)

Table 3: Comparison of SOP update schemes. UC denotes useful continuation; S30 denotes starting within 30 seconds. P50 and P95 are the 50th and 95th percentiles of switching latency. Dup., Inv., Rej., and Loss denote duplicate actions, invalid activation, rejection of valid updates, and state loss, respectively. Arrows indicate the preferred direction, and bold values indicate the best displayed values, including ties.

Update scheme	UC (%) ↑	S30 (%) ↑	P50 (s) ↓	P95 (s) ↓	Dup. (%) ↓	Inv. (%) ↓	Rej. (%) ↓	Loss (%) ↓
AeroSOP	**91.0**	96.2	**1.1**	4.7	**0.2**	**0.2**	0.4	**0.2**
State-preserving Full Refresh	**91.0**	90.2	1.9	6.8	0.4	0.6	0.4	0.4
Stateless Patch	79.6	91.2	2.0	9.7	14.2	0.8	0.4	16.4
w/o Version Check	86.0	85.6	1.2	6.8	1.0	13.6	**0.2**	5.2
w/o Idempotency	87.8	88.8	1.2	**4.5**	12.8	0.4	0.4	1.0
Receive-Time Check Only	85.4	**97.0**	1.2	4.6	0.6	9.8	**0.2**	1.8

Observation 5: State-compatible patches match full-refresh continuation with lower switching latency. Table 3 shows that AeroSOP matches state-preserving full refresh at 91.0% useful continuation, reducing P50/P95 switching latency from 1.9/6.8 s to 1.1/4.7 s. Its observed rates of duplicate actions, invalid activation, and state loss are each 0.2%. Stateless patching lowers useful continuation to 79.6%, with duplicate actions and state loss reaching 14.2% and 16.4%. Removing version checks or using receive-time-only checks raises invalid activation to 13.6% and 9.8%, respectively; removing idempotency raises duplicate actions to 12.8%. These results support state preservation, activation-time validation, and idempotency as complementary safeguards for reliable continuation.

Observation 6: Return-aware continuation improves both return feasibility and task utility. Figure 5 shows that return failure decreases from 2.6% under a fixed battery threshold to 0.9% with return-aware admission, 0.7% with online monitoring, and 0.2% with the full AeroSOP policy. Relative to FBT, AeroSOP reduces observed return failures by 92.3%. Return-aware admission provides the largest initial reduction, lowering return failure from 2.6% to 0.9%. Online monitoring further reduces it to 0.7%, while the full AeroSOP policy suppresses the remaining failures to 0.2%. Meanwhile, utility increases monotonically from 83.6 to 84.8, 87.0, and 88.4 across the four policies. These results show that return-aware admission establishes the primary return-feasibility constraint, while online monitoring and the complete continuation policy provide additional protection as execution conditions evolve. AeroSOP therefore improves return feasibility without sacrificing useful task continuation.

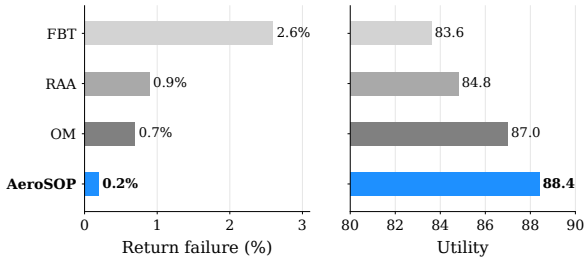


Figure 5: Return failure and utility under different continuation policies.

4.5 System Efficiency and Scalability (RQ4)

Table 4: Online overhead and onboard resource requirements. BS denotes base station, and RSS denotes resident set size. CPU time is measured per AAV-minute. Bold values indicate the lowest values in each column, including ties.

Method	Online overhead per mission ↓			Onboard resources ↓	
	BS calls (calls)	BS tokens (tokens)	Control traffi (KiB)	Executor CPU (s/AAV-min)	Peak RSS (MiB)
Remote Skill-by-Skill Dispatch	8.4	12,432	36.4	0.78	50.6
Batched Skill Dispatch	3.1	6,448	19.3	0.57	43.5
State-preserving Full Refresh	2.3	4,456	18.6	0.82	51.2
AeroSOP	**1.2**	**3,756**	**8.9**	**0.46**	**39.8**

Observation 7: AeroSOP reduces coordination overhead while keeping onboard execution lightweight without task-level LLM reasoning. Table 4 shows that AeroSOP requires only 1.2 BS calls, 3,756 tokens, and 8.9 KiB of control traffic per mission, reducing these costs by 85.7%, 69.8%, and 75.5%, respectively, relative to remote skill-by-skill dispatch. Compared with state-preserving full refresh, incremental SOP updates further reduce control traffic by 52.2%. Despite shifting task-level reasoning to the BS, AeroSOP does not introduce additional onboard execution cost: its SOP executor records the lowest CPU time and peak RSS, at 0.46 s/AAV-min and 39.8 MiB. These results show that AeroSOP reduces remote coordination while retaining a lightweight onboard execution path.

Observation 8: AeroSOP exhibits the smallest latency growth as the fleet scales and achieves the lowest response latency at 24 AAVs. Although skill-by-skill dispatch is faster at small fleet sizes, its latency increases rapidly with load. From 3 to 24 AAVs, AeroSOP's response latency increases by only 0.84 s, from 2.38 to 3.22 s, compared with increases of 14.70 s for skill-by-skill dispatch and 4.46 s for batched dispatch. At 24 AAVs, AeroSOP achieves the lowest latency at 3.22 s, reducing response time by 79.7% relative to skill-by-skill dispatch, 46.7% relative to batched dispatch, and 6.7% relative to state-preserving full refresh. Together with its lower BS interactions and control traffic, this result shows that SOP-level execution limits coordination growth as the fleet expands. Overall, AeroSOP supports multi-AAV execution with a lightweight onboard executor and sparse BS coordination, while maintaining bounded response-latency growth as the fleet scales to 24 AAVs.

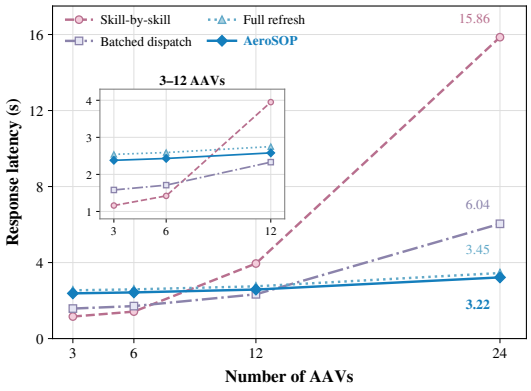


Figure 6: Response latency versus fleet size, with a zoomed inset for 3-12 AAVs.

5 Conclusion

We presented AeroSOP, a framework that connects base-station task reasoning with onboard AAV execution through executable SOPs. The framework combines tested templates, step contracts, mission-level verification, and bounded local recovery with state-compatible patches for continuous missions. Task-level language reasoning remains at the base station, while perception, local planning, control, and immediate safety responses remain onboard. Simulation results show strong inspection reliability and competitive overall task completion. The patch mechanism achieves 91.0% useful continuation, matching state-preserving full refresh with lower switching latency. The architecture also reduces control traffic and limits response-latency growth over the tested fleet sizes. These findings support executable procedures with explicit update semantics as an interface for continuous AAV operation. Future work will improve compound-task execution, refine continuation decisions under uncertainty, and validate the framework on physical AAV platforms. See Appendix A.5 for additional limitations and discussion.

AI USE STATEMENT

Generative AI tools were used solely for language polishing, including improving grammar, clarity, fluency, and conciseness of author-written text. They were not used to generate research ideas, methods, experiments, data, analyses, results, or citations. All AI-assisted edits were reviewed and verified by the authors, who take full responsibility for the final content of this work.

REFERENCES

Abrar Anwar, John Welsh, Joydeep Biswas, Soha Pouya, and Yan Chang. Remembr: Building and reasoning over long-horizon spatio-temporal memory for robot navigation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2838–2845, 2025. doi: 10.1109/ICRA55743.2025.11127706.

Noam Buckman, Han-Lim Choi, and Jonathan P How. Partial replanning for decentralized dynamic task allocation. In *AIAA Scitech 2019 Forum*, pp. 0915, 2019.

Guojun Chen, Xiaojing Yu, Neiwen Ling, and Lin Zhong. Typefly: Flying drones with large language model, 2024a. URL https://arxiv.org/abs/2312.14950.

Xinglin Chen, Yishuai Cai, Yunxin Mao, Minglong Li, Wenjing Yang, Weixia Xu, and Ji Wang. Integrating intent understanding and optimal behavior planning for behavior tree generation from human instructions. In Kate Larson (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 6832–6840. International Joint Conferences on Artificial Intelligence Organization, 8 2024b. doi: 10.24963/ijcai.2024/755. URL https://doi.org/10.24963/ijcai.2024/755. Main Track.

Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6695–6702, 2024c. doi: 10.1109/ICRA57147.2024.10611163.

Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a LLM. In *Language Gamification - NeurIPS 2024 Workshop*, 2024. URL https://openreview.net/forum?id=ewx2RFiEYR.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 45493–45517, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/70a06501001e1820fd1eb9ee821302d2-Paper-Conference.pdf.

Björn Döschl, Kai Sommer, and Jane Jean Kiam. Auspex: An integrated open-source decision-making framework for uavs in rescue missions. *Frontiers in Robotics and AI*, Volume 12 - 2025, 2025. ISSN 2296-9144. doi: 10.3389/frobt.2025.1583479. URL https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2025.1583479.

Yuhan Hu, Yirong Sun, Yanjun Chen, Xinghao Chen, Xiaoyu Shen, and Wei Zhang. Breaking the pre-planning barrier: Adaptive real-time coordination of heterogeneous uavs, 2025. URL https://arxiv.org/abs/2501.14488.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/huang22a.html.

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. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 540–562. PMLR, 06–09 Nov 2023a. URL https://proceedings.mlr.press/v229/huang23b.html.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. Inner monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 14–18 Dec 2023b. URL https://proceedings.mlr.press/v205/huang23c.html.

brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 14–18 Dec 2023a. URL https://proceedings.mlr.press/v205/ichter23a.html.

brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 14–18 Dec 2023b. URL https://proceedings.mlr.press/v205/ichter23a.html.

Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci. Btgenbot: Behavior tree generation for robotic tasks with lightweight llms. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9684–9690, 2024. doi: 10.1109/IROS58592.2024.10802304.

Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and Michael Gienger. Copal: Corrective planning of robot actions with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8664–8670, 2024. doi: 10.1109/ICRA57147.2024.10610434.

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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591.

Xiaoshan Lin, Yasin Yazıcıoğlu, and Derya Aksaray. Robust planning for persistent surveillance with energy-constrained uavs and mobile charging stations. *IEEE Robotics and Automation Letters*, 7(2):4157–4164, 2022. doi: 10.1109/LRA.2022.3146938.

Neiwen Ling, Guojun Chen, Anurag Khandelwal, and Lin Zhong. Timelyllm: Time-sensitive llm serving system for physical-i/o limited agents. In *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services*, MobiSys '26, pp. 261–278, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400720277. doi: 10.1145/3745756.3809203. URL https://doi.org/10.1145/3745756.3809203.

Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Grounding complex natural language commands for temporal tasks in unseen environments. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1084–1110. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/liu23d.html.

Yibin Liu, Zhixuan Liang, Zanxin Chen, Tianxing Chen, Mengkang Hu, Wanxi Dong, Congsheng Xu, Zhaoming Han, Yusen Qin, and Yao Mu. Hycodepolicy: Hybrid language controllers for multimodal monitoring and decision in embodied agents, 2025. URL https://arxiv.org/abs/2508.02629.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 23–72. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/rana23a.html.

Zachary Ravichandran, Varun Murali, Mariliza Tzes, George J. Pappas, and Vijay Kumar. Spine: Online semantic planning for missions with incomplete natural language specifications in unstructured environments. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13714–13721, 2025. doi: 10.1109/ICRA55743.2025.11128238.

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. *IEEE Robotics and Automation Letters*, 10(11):12237–12244, 2025. doi: 10.1109/LRA.2025.3619745.

Sepehr Seyedi, Yasin Yazicioğlu, and Derya Aksaray. Persistent surveillance with energy-constrained uavs and mobile charging stations. *IFAC-PapersOnLine*, 52(20):193–198, 2019. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2019.12.157. URL https://www.sciencedirect.com/science/article/pii/S2405896319320087. 8th IFAC Workshop on Distributed Estimation and Control in Networked Systems NECSYS 2019.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: open-ended instruction following with hierarchical vision-language-action models. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

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.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2998–3009, October 2023.

Jonathan Styrud, Matteo Iovino, Mikael Norrlöf, Mårten Björkman, and Christian Smith. Automatic behavior tree expansion with llms for robotic manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1225–1232, 2025. doi: 10.1109/ICRA55743.2025.11127942.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 58202–58245. Curran Associates, Inc., 2023. doi: 10.52202/075280-2537. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/b5c8c1c117618267944b2617add0a766-Paper-Conference.pdf.

Andrea Tagliabue, Kota Kondo, Tong Zhao, Mason Peterson, Claudius T. Tewari, and Jonathan P. How. Real: Resilience and adaptation using large language models on autonomous aerial robots. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 1539–1546, 2024. doi: 10.1109/CDC56724.2024.10885890.

Sai H. Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *IEEE Access*, 12:55682–55696, 2024. doi: 10.1109/ACCESS.2024.3387941.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Xianchao Zeng, Xinyu Zhou, Youcheng Li, Jiayou Shi, Tianle Li, Liangming Chen, Lei Ren, and Yong-Lu Li. Diagnose, correct, and learn from manipulation failures via visual symbols. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 42386–42395, June 2026.

Sheng Zhang, Qinglin Li, Yuechao Zang, Xueqin Huang, Yijia Fu, and Cheng Zhu. Multiuav-plat: An llm-oriented platform, benchmark and framework for multi-uav collaborative task planning. *arXiv preprint arXiv:2606.31073*, 2026.

A Additional Method Details

This appendix expands the construction, execution, and update mechanisms in Section 3. The complete pseudocode appears at the end in Algorithm 1.

A.1 SOP Contracts, Offline Preparation, and Construction

The SOP representation binds the task's procedural structure to deployed capabilities and explicit completion evidence:

$$\mathcal{P}_j = (\mathcal{G}_j, \mathcal{B}_j, \Theta_j, \mathcal{K}_j, \phi_j), \qquad \mathcal{G}_j = (\mathcal{V}_j, \mathcal{E}_j). \tag{6}$$

Here, $\mathcal{G}_j$ connects executable nodes through execution dependencies, data dependencies, and conditional transitions; $\mathcal{B}_j$ contains skill bindings, Θ_j parameters and input references, and $\mathcal{K}_j$ node contracts. The predicate ϕ_j specifies the task-level completion evidence. Skill bindings identify both a deployed skill and its interface version. Parameters resolve targets, coordinate frames, altitude references, units, and operating limits using task inputs, telemetry, and retained results. A future input is an explicit reference to its producer; its type and dependency are checked during construction, and its realized value is checked before dispatch. Missing external inputs or incompatible interfaces block the corresponding operation. Navigation skills receive destinations and motion constraints, while local planners and controllers determine trajectories and control inputs.

Each contract $\kappa_i = (p_i, q_i, \tau_i, \rho_i, f_i)$ associates the node with declared checker and failure interfaces. Predicates p_i and q_i evaluate applicability and outcome evidence, τ_i bounds an attempt, ρ_i bounds retries, and f_i specifies an approved failure response. Bounded retry subprocedures are encapsulated within nodes. The Scan–Verify retry in Figure 1, for example, is a local bounded subprocedure within the acyclic task-level graph. Its retry counter prevents indefinite repetition; failure transitions between graph nodes must also respect the validated termination structure.

Offline preparation produces a tested template library $\mathcal{L}$. Templates are exercised under nominal, failure, and boundary cases and revised when checks fail. Each admitted template records its identifier, template version, required skill interfaces, and tested operating range. Pre-flight provisioning synchronizes the required definitions, runtime, skills, return policy, and initial validated procedure with the AAV. Passing these tests provides evidence within the tested conditions; it does not establish validity for every parameterization or online composition.

For task j, the base station retrieves applicable templates $\mathcal{L}_j \subseteq \mathcal{L}$ and constructs a candidate:

$$\widetilde{\mathcal{P}}_j = F_{\mathrm{LLM}}(g_j, \mathcal{C}_j, \phi_j, \mathcal{L}_j, \mathcal{S}, \hat{\mathbf{z}}, \mathcal{R}), \tag{7}$$

where $\mathcal{S}$ is the deployed skill profile and $\mathcal{R}$ contains available results from preceding tasks. The checks in equation 2 assess interface and cached-template compatibility, typed parameters and references, dependency consistency, branch reachability, termination structure, contract completeness, and retry and timeout bounds. State-dependent tools evaluate capability, spatial, deadline, and energy feasibility. Energy admission follows equation 5. Diagnostics identify violations for revision within a fixed construction budget; unresolved violations cause rejection. Validation is relative to the declared interfaces and models and is repeated locally when changing state can invalidate a previously accepted candidate.

Eligible tasks must have been released, remain pending, and have their prerequisite results available. After preliminary capability and deadline checks, the base station orders them by decreasing value and then increasing deadline. Construction stops when a valid candidate is found or the decision budget expires. This is the online heuristic used to pursue equation 1; optimal scheduling is not assumed.

A.2 Runtime Verification and Bounded Recovery

For an active, committed SOP, the runtime maintains node states and an execution record $\mathcal{R}_j(t)$. Let $\mathcal{V}_j^{\mathrm{act}}(t)$ contain nodes admitted by resolved branch conditions, and let $\mathrm{Pred}_j^{\mathrm{act}}(i,t)$ denote the

dependencies required for node i on those branches. The enabled set is

$$\mathcal{D}_j(t) = \big\{ i \in \mathcal{V}_j^{\mathrm{act}}(t) : s_i(t) = \text{pending}, \quad p_i(\mathbf{z}(t), \mathcal{R}_j(t)) = 1, \\ \mathrm{InputsReady}_i(t) = 1, \\ s_h(t) = \text{success} \quad \forall h \in \mathrm{Pred}_j^{\mathrm{act}}(i,t) \big\}. \tag{8}$$

Unresolved branch conditions do not enable their dependent nodes. Dispatch is restricted to $\mathcal{D}_j(t)$ and remains subject to the local safety monitor. Each invocation receives task and attempt identifiers, and reused results must be referenced explicitly and satisfy the receiving node's input conditions.

For attempt a of node i, let $e_i^{(a)}$ denote returned evidence and $o_i^{(a)}$ its return status. At return time t, success requires

$$\mathrm{Success}_i^{(a)}(t) = \mathbf{1}[o_i^{(a)} = \mathrm{ok}]\mathbf{1}[q_i(\mathbf{z}(t), e_i^{(a)}) = 1]. \tag{9}$$

A timed-out attempt cannot have an accepted normal return status. The execution record retains identifiers, results, and verification outcomes, including failures, together with the observations and timing information required by task-level checkers. A successful skill return without the required evidence does not release successors. Equation equation 3 additionally requires a success terminal, task-level evidence, and satisfaction of the declared constraints. An inspection SOP may combine `NavigateTo`, `ScanRegion`, conditional `CaptureImage`, and `UploadResults`; its predicate can require coverage, observation quality, and confirmed delivery. Undelivered required artifacts leave it incomplete.

Timeouts, failed checks, and unavailable capabilities invoke f_i. Before retrying, the runtime rechecks preconditions, remaining time and energy, and whether repetition is allowed directly or requires state reconciliation. At most $1 + \rho_i$ attempts are permitted. A node becomes pending again only after these checks pass. Exhausted recovery terminates the affected task without producing a success boundary, preserves its record, and triggers a preloaded fallback with a diagnostic report. Perception, local planning, control, and safety monitoring continue independently of task-level language reasoning.

A.3 Patch Validation, Atomic Activation, and Protocol Properties

Let $\mathscr{P}^r$ denote the installed procedure bundle at revision r. A verified completion boundary has a unique event identifier b and a bounded waiting expiry. Its report includes state, retained results, revision, and expiry. The next-task patch binds the proposed procedure to this report:

$$\Delta = (\iota, u_\Delta, r_b, b_\Delta, \ell, \Omega), \tag{10}$$

where ι is a unique patch identifier, u_Δ the target AAV, r_b the required base revision, and b_Δ the intended boundary. The expiry ℓ is copied from the boundary report in the AAV-local clock, so acceptance does not require clock synchronization. The payload Ω contains SOP additions or cached references, required skill versions, input bindings, and activation guards. Bundle revisions r_k and r_{k+1} in Figure 1 identify successive handovers and remain distinct from template versions.

The retained ledger $\mathcal{H}$ records accepted patch identifiers and activation status. A known identifier receives its recorded status without restarting the task, even if the installed revision has subsequently advanced. For an unseen patch, the runtime reconstructs $\mathscr{P}^\Delta = \mathrm{Apply}(\mathscr{P}^{r(t)}, \Omega)$ in staging and validates its next-task procedure $\mathcal{P}^\Delta$. Missing cached definitions, incompatible skill versions, or unavailable required external inputs cause rejection; internally produced inputs retain the explicit dependency semantics described in Appendix A.1.

The compatibility term in equation 4 is

$$C_\Delta(t) = \mathbf{1}[u_\Delta = u,\ r_b = r(t),\ b_\Delta = b(t)] \cdot \mathbf{1}[t < \ell,\ \iota \notin \mathrm{dom}(\mathcal{H}(t))]. \tag{11}$$

Here, comma-separated conditions within an indicator are conjunctive. The remaining terms in equation 4 require current feasibility, an open waiting window, no active skill invocation, and no latched safety response. A candidate feasible under reported state may therefore fail admission after further waiting or a change in capability status.

Commit is serialized. The runtime rechecks all activation conditions, including identifier uniqueness. If all hold, it performs

$$(\mathscr{P}, r, \mathcal{H}) \longleftarrow \Big(\mathscr{P}^\Delta,\ r+1, \mathcal{H} \cup \{\iota \mapsto (r+1, \text{activated})\} \Big) \tag{12}$$

atomically, closes the waiting window, and selects the next SOP. Completed results and unchanged definitions are retained. Only committed procedures can dispatch skills. Rejection preserves the installed bundle; an unknown patch with an incompatible revision or boundary is rejected. A lost acknowledgment is handled by retransmitting the same identifier or querying its status.

Two consistency properties follow under the stated runtime assumptions. First, staging cannot change the installed bundle before a successful commit, so failed validation preserves the accepted procedure. Second, serialized uniqueness checks and a ledger retained throughout the mission allow at most one activation per accepted patch identifier. Concurrent duplicates cannot both commit: after the first insertion, the next uniqueness check fails. These properties concern procedure updates; physical safety also depends on the assumptions below.

A.4 Energy Accounting and Safety Assumptions

Equation equation 5 conservatively accounts for remaining waiting, travel to the task region, task execution, and return within the modeled disturbance range. Task energy includes permitted branches and retries, while return energy covers their possible terminal states. Already consumed waiting energy is reflected in $E_{\mathrm{rem}}(t)$ and must not be counted again. Admission uses reported state during construction and the latest local state before activation.

During execution and waiting, the safety monitor evaluates the current return margin:

$$M(t) = E_{\mathrm{rem}}(t) - \overline{E}_{\mathrm{return}}(\mathbf{z}(t)). \tag{13}$$

Return is triggered when $M(t) \leq E_{\mathrm{reserve}}$. Let $\overline{\Delta M}_{\mathrm{mon+stop}}$ bound the reduction of this margin over one monitoring interval and the subsequent skill interruption. To retain a residual margin $E_{\mathrm{min}} > 0$ when return begins, the reserve satisfies

$$E_{\mathrm{reserve}} \geq \overline{\Delta M}_{\mathrm{mon+stop}} + E_{\mathrm{min}}. \tag{14}$$

Waiting also ends when its time or energy budget expires. A triggered return or emergency response is latched against later task updates. During link loss, a committed SOP may continue while its contracts and safety conditions hold, but further chaining requires a valid received patch. Return feasibility depends on conservative energy bounds, bounded interruption costs, adequate observations, and a reachable return route. Contract checking alone does not provide an unconditional physical-safety guarantee.

A.5 Limitations and Discussion

Our current evaluation focuses on simulated urban missions with fleets of up to 24 AAVs. The experiments cover multiple environment densities, dynamic task arrivals, communication changes, and execution disturbances, providing a broad test of the proposed execution framework. Validation on physical AAV platforms, heterogeneous vehicles, and substantially larger fleets remains an important direction for future work. The reported onboard resource measurements characterize the SOP executor under the evaluated platform, while deployment on embedded flight hardware will provide a more complete assessment of runtime and resource behavior.

AeroSOP assumes that the required skills, SOP templates, and verification predicates are available to the runtime. Offline testing improves the reliability of reusable procedures, while online checks adapt their execution to the current vehicle state. Expanding the template library and verification coverage could further improve support for more diverse task compositions. Compound missions remain the most challenging category in the current evaluation, suggesting opportunities for stronger composition and recovery mechanisms.

State-compatible patches are activated at verified task-completion boundaries to preserve completed results and maintain update consistency. This design emphasizes reliable inter-task adaptation rather than arbitrary modification of an ongoing task. During temporary communication loss, an accepted SOP can continue locally while its execution and safety conditions remain valid, whereas introducing a subsequent task requires a valid base-station update. Finer-grained state-compatible updates are a promising extension for missions requiring more frequent online adaptation.

Return-aware admission and online monitoring regulate continuation using the current energy state and estimated return requirement. Their effectiveness depends on the accuracy of energy estimates,

state observations, and return-path information. Conservative continuation decisions can also reduce task utility in some cases. Improved uncertainty-aware estimation and task selection may further increase mission utility while retaining reliable return behavior.

Overall, the current limitations mainly concern evaluation scale, procedural coverage, and adaptation granularity. Extending AeroSOP to physical platforms, heterogeneous and larger fleets, richer task compositions, and uncertainty-aware continuation represents a natural next step.

A.6 Event-Driven Algorithm

Algorithm 1 summarizes the base-station and onboard event handlers, which operate independently. Safety monitoring remains active during both procedure execution and waiting.

Algorithm 1 Event-driven SOP construction and continuous mission chaining

procedure BaseStation($b, \hat{\mathbf{z}}, \mathcal{R}, \ell$)
  for task j in the ranked eligible task list **do**
    if the decision budget expires **then**
      return
    end if
    Construct and revise $\mathcal{P}_j$ within the remaining budget
    if $V(\mathcal{P}_j, \hat{\mathbf{z}}) = 1$ and the budget remains **then**
      Send Δ bound to b, the reported revision, and ℓ
      return
    end if
  end for
end procedure
procedure AAVEvent($e, \mathbf{z}$)
  Service the safety monitor and any latched fallback with priority
  if e is patch arrival and $\iota \in \mathrm{dom}(\mathcal{H})$ **then**
    Reply with the recorded status; **return**
  end if
  if safety is latched or the active waiting budget expires **then**
    Latch and execute the appropriate local fallback
    Reject a newly received patch, if any; **return**
  else if e is completion verified by equation 3 **then**
    Record b and open a bounded waiting window
    Report state, results, revision, and expiry
  else if e is a previously unseen patch arrival **then**
    Reconstruct and validate the candidate in staging
    if $A_{\Delta}(t) = 1$ at serialized commit **then**
      Install, advance revision, and record activation atomically
      Close the waiting window and select the next SOP
    else
      Reject Δ and retain the installed procedure
    end if
  else if a procedure is active **then**
    Dispatch enabled nodes; verify outcomes with bounded recovery
  end if
end procedure