Autonomous Mission Planning with an Onboard Vision-Language Model: What a Small Model Can and Cannot Do for a CubeSat, Measured

Patrick McKeen[*] and Zac Manchester[**]
[*]Massachusetts Institute of Technology
Cambridge, MA 02139, USA; pmckeen@mit.edu
[**]Massachusetts Institute of Technology
Cambridge, MA 02139, USA; zacm@mit.edu

Abstract

We built and evaluated a planner in which Gemma 3 4B makes the high-level decisions for a simulated Earth-imaging CubeSat, through reversible tools and deterministic checks that decide when the model is consulted, what it is shown, and whether its answer is accepted. The model reads operator intent well and, with its output grammar enforced, stays usable over multi-hour missions. But a rule tree applying what the same checks read in the directive delivered five of six targets where the model delivered two, and the perception limit moved with the vision backbone and spectral bands, not model size.

1. Introduction

Most spacecraft are still tasked from the ground. A plan is built, uplinked, and executed on a timescale of hours to days, and anything that happens in between waits for the next contact. On-board autonomy has so far come in two forms, schedulers and risk-aware optimizers, and learned policies trained for one task. Both work when the objectives, constraints, and action set can be written down in advance, and both are stuck when they cannot, which is when reacting on its own would help most: a wildfire that was not on the target list, an operator note deprioritizing a region tonight, a capture that came back cloudy.

Small vision-language models (VLMs) offer a different trade. A model of a few billion parameters can read a mission statement and a live state summary, weigh a priority-1 target with a grazing pass against a priority-2 target directly overhead, and emit a decision, on a Jetson-class processor. What it cannot do is be verified the way flight software is verified. This paper is about one way to get the flexibility while keeping the model's authority bounded, and what measurement shows.

We built AI_TASKER, a planner in which Gemma 3 4B [1] does the high-level thinking for a simulated 3U CubeSat and the Generalized_ADCS stack [2] does the flying. The model decides *what* to do (which target, whether to charge, whether to re-image, what to send down), and classical control decides *how*; the model never writes a torque. It sees the spacecraft through a typed set of tools, all read-only or reversible planning changes, and a small library of deterministic checks sits between the model and the vehicle.

The architecture is a straightforward application of propose-and-verify [3] and classical function allocation [4] to a new setting. What is new is the evidence: for each design choice we can say what we measured and what went wrong when we did it the other way.

Where this sits. Remote Agent [5] and EO-1's Autonomous Sciencecraft [6] bounded every autonomous action by construction and lacked open-world flexibility; Φ-Sat-1 [7] flew on-board cloud screening. NAVI-Orbital [8] flew a small VLM for scene description and reported unbounded hallucination as an open problem; ASTREA [9] ran an LLM agent on the ISS and found its latency mismatched to the thermal cycle it supervised. Planners built on large models [10] degrade sharply at small scale, which is why our checks sit outside the model.

The case for a language model on board came out narrower than the one we set out to make. The plainest alternative, a rule tree that applies what the gate's checks already read out of the operator's text, matched the hybrid on all but one scenario, and what the model still added can reasonably be done on the ground. Narrower is not none. The case that motivated the work, an event nobody wrote a rule or a target for, where waiting for the next contact would lose the

Copyright © 2026 by Patrick McKeen and Zac Manchester. Published by the i-SAIRAS association with permission.

observation, is the one these experiments do not test. The measurements here say what it would take. It needs perception trained for the events the mission is meant to catch, and a model whose authority is bounded as in Sec. 3. Three other things surprised us. The model's own confidence is useless for deciding when to abstain. The failure that took down a long mission was a slow drift into malformed output. And on the content-driven downlink the vision, not the planner, is the limit. A 4 B model cannot see fine damage in an overhead tile, a bigger model does no better, and a differently trained vision backbone does.

2. The Mission and the Planner

Spacecraft and orbit. The simulated vehicle is a 3U CubeSat with magnetorquers and reaction wheels in a circular 400 km orbit at 51.6° inclination. The flight software is our open-source Generalized ADCS framework [2]: six-degree-of-freedom attitude dynamics, a family of Kalman estimators, and pointing controllers derived for arbitrary wheel-and-magnetorquer combinations [11], here a PD law over both actuator sets. The vehicle starts tumbling; a rule-based commissioning autopilot (not the model) runs B-dot detumble, hands over to the pointing controller after about two minutes, and settles on a nadir hold; from then on the planner owns the pointing goals.

Targets, stations, and objectives. The demo mission carries six ground targets, Alpha through Foxtrot, with priorities from 1 (top) to 3 (nice to have), and two ground stations, Pittsburgh and Lisbon, at about 1 Mbps. The standing objectives read: *"Image every ground target on the list at least once. Prefer high-elevation passes (better imagery) and conserve battery between passes. Follow up on captures whose on-board analysis flagged an anomaly."* On top of that, an operator can uplink a free-text directive at any time, which appears to the planner as a `[TASKING]` block. The directives in our evaluation are the kind an operator writes, such as the cloud forecast in Figure 1 or *"URGENT: volcanic eruption reported in the Indonesian archipelago; investigate immediately."*

What a planning turn looks like. Every 30 simulated seconds a deterministic renderer assembles a user turn from the spacecraft state and the target list (Figure 1, lightly condensed). The model answers with one JSON object naming a tool, its arguments, and a short rationale. The system prompt (about 4 KB, frozen for all experiments) explains the fields, tells the model to commit to a target while it is still *imminent* because the slew takes about 150 s, and gives a priority order: charge if the battery is low, re-image a cloudy capture if one is waiting, otherwise pick the best overhead or imminent target by priority and peak elevation.

```
[STATE]
{ "t_s": 2300, "active_goal": "Nadir", ... }

[TARGETS] (sorted most-actionable first)
[ { "name": "Charlie", "priority": 1, "pass": "OVERHEAD",
    "elev_now_deg": 50, "peak_elev_deg": 70,
    "n_capture_attempts": 0, "last_image_usable": null },
  { "name": "Foxtrot", "priority": 2, "pass": "OVERHEAD",
    "peak_elev_deg": 55, ... } ]

[TASKING] (operator directive)
Cloud forecast over Charlie tonight: 80% obscured.
Skip Charlie if a usable alternative exists.

--> {"tool": "push_target_imaging_goal",
     "args": {"target_name": "Foxtrot"},
     "thought": "Charlie forecast cloudy; Foxtrot ok"}
```

Figure 1: One planning turn, condensed. The rule tree would pick Charlie (priority 1, higher peak); the operator's note makes Foxtrot right.

The tools. The catalog has 25 tools: sixteen read-only (state, clock, orbit, pass predictions, the downlink queue, and the like) and nine that change the plan, every one of them reversible: point the vehicle at nadir, the Sun, or a coordinate; start imaging a named target; add a target by latitude and longitude; mark a cloudy capture for re-imaging; queue a capture for downlink at a chosen compression and priority; switch controller or estimator; log a rationale. Under grammar-constrained decoding (Sec. 4.3) the model can emit eight of them. No tool fires a thruster or commands a wheel, and every plan change can be replaced on the next tick; as Sec. 4.3 shows, that restriction is the deepest safety guard in the system.

Imaging, analysis, and downlink. When a target is overhead and the camera is pointed, the simulator synthesizes a capture: a scene drawn from a library of real overhead imagery [12], sometimes with a real disaster scene inserted at the target, sometimes obscured by cloud. There is no weather model; cloud is injected at random, which is why seven of the nine captures in the run below came back cloudy. An on-board classifier then summarizes the scene and grades

six damage “atoms” (flood, burn, active fire, collapse, and so on) at four confidence levels. Four compression options are costed with real byte counts (for a typical capture about 660 KB lossless, 62 KB at JPEG quality 75, 3 KB as a thumbnail); the planner decides whether to queue the capture, at which quality and priority, and the queue drains during station passes.

A mission timeline. Table 1 shows the production planner’s six-hour run on the seed we return to in Sec. 4.3: two targets captured clear and queued, three cloudy captures never retried, Charlie cloudy twice, and Delta re-imaged sixteen seconds after a cloudy capture, within the 5° pointing tolerance a capture requires. Nothing was transmitted for 4.7 hours because the spacecraft never rose above either station’s 10° elevation mask in its first 2.6 orbits; at 4.77 h the first Lisbon pass drained the 74 KB queue in two seconds.

Table 1: The production planner’s six-hour run (seed 7), through the first station pass.

t (s)	Event	Planner’s decision
348	capture: Alpha, cloudy	
1850	capture: Bravo, ocean, usable	
1862		queue Bravo: q75, 35 KB
3350	capture: Charlie, cloudy; again at 3394	re-image Charlie
4848	capture: Delta, cloudy	re-image Delta
4864	capture: Delta, ocean, usable	
4892		queue Delta: q75, 39 KB
6346	capture: Echo, cloudy; Foxtrot (7852) cloudy	
17 172	Lisbon pass: Bravo, Delta transmitted	

How we evaluate it. Besides long end-to-end runs, we use a battery of 27 frozen planning situations, each a fixed state, target list, objectives text, recent history, and station schedule. Eight are *rule-solvable*: a simple priority rule gets them right. Nineteen are *judgment-required*: more than one answer is defensible and the operator’s text decides, as in Figure 1. For those we fixed the set of acceptable answers before looking at model output at scale, and included each known failure pattern three times so no single scenario carries a claim. Every scenario runs at five seeds; at our temperature the model’s answer never flipped between seeds, so the unit of evidence is the scenario. An independent model, shown none of our rubric, agreed with 85% of our pass/fail calls. Three further sets, written after the system was frozen, serve only as held-out tests. Latencies were measured on an Apple M5 Pro laptop through Ollama, whose default 4 B build is a 4-bit quantization; nothing has run on flight-class hardware. Code and raw outputs will be released at `github.com/patrickmckeen/ai-tasker`.

3. Architecture and Where the Model’s Authority Ends

Figure 2 shows the whole system. A perception layer (the image classifier, the attitude and orbit estimators, ephemeris) reports to the planner and is kept outside the language model on purpose. The deterministic layer holds the renderer that builds each turn, a rule tree that always produces a default action from the state alone, and the check library. Five checks look at the rule’s default action in the light of the operator’s text: does it violate a stated threshold, pick a target the operator said to skip, image under a no-imaging or hold instruction, or point at the Sun in eclipse? If none fires, the rule’s action is executed and the model is never called. If one fires, the model is consulted, and the same library decides whether to prepend a short bank of worked examples for that kind of situation. The reply is decoded under a JSON grammar that fixes the tool name and its argument fields; a final check vetoes one bad action (queueing a capture the classifier called unusable) and substitutes a nadir hold; whatever survives is dispatched through a whitelist. One caveat applies to every production run below. The routing step is evaluated on the scenario battery, while the production loop calls the model on every tick, with the same library choosing its examples and screening its replies; call counts differ accordingly.

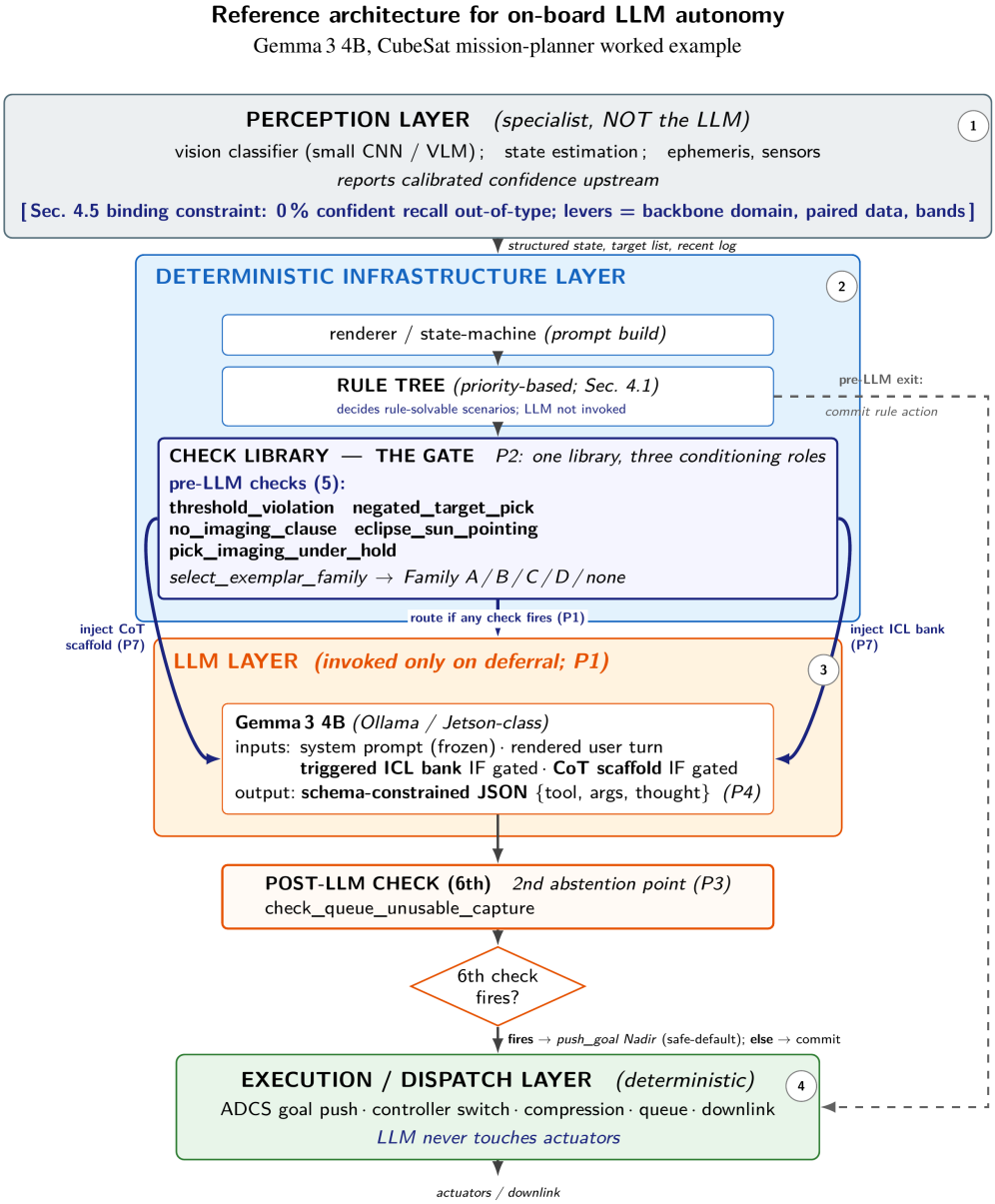


Figure 2: The planner. Most of it is deterministic software; the model is the orange block, consulted only when a check fires. One check library decides whether to ask it, what to show it, and whether to accept its answer.

Deciding where the boundary goes. The question that mattered most was which decisions need the model at all. Following Parasuraman, Sheridan, and Wickens [4], acquiring information, analyzing it, and executing an action are done deterministically; only the choice of action goes to the model, and only when a rule cannot read what it depends on ("skip Charlie if the forecast is cloudy" qualifies; "image the highest-priority target overhead" does not). About 37% of our planner's candidate decisions qualify, and Sec. 4.1 shows that most even of those can be settled deterministically once a check has read the directive. A directive is written by a person at a console, with time, a map, and a chance to check the parse. The hard part on board is acting on a structured instruction under conditions the ground did not foresee. So for every judgment the model gets right below, we ask whether a check could have read the same thing. The failures that remain sort by what fixes them: a larger model, a gated example, or a check.

4. Results

4.1 How much the model is worth, and when to ask it

Table 2 compares five ways of running the planner on the 27-scenario battery. In *LLM-on* the model decides everything. In *rule-only* the rule tree decides everything. In *forced-defer* the model is called every time but must accept the rule's answer whenever the rule has one. In *content-aware*, the configuration in Figure 2, the model is called only when a check fires.

Table 2: Five ways to run the planner (Gemma 3 4B, 27 scenarios × 5 seeds). Rule+ applies the checks' readings of the directive to the rule tree, no model.

Subset	LLM-on	Rule	Rule+	Forced	Content-aware
Rule-solvable (8)	7/8	8/8	8/8	8/8	8/8
Judgment-req. (19)	14/19	5/19	15/19	7/19	11/19
Overall (27)	78%	48%	85%	56%	70%
Mean latency	2.59 s	0	0	2.59 s	0.97 s
LLM calls	100%	0%	0%	100%	37%

The model helps only on the judgment-required scenarios, where it gets nine more right than the rules do, and is slightly worse on the simple ones, where a rule is perfect. Forced deferral inherits the rule's wrong answers in the cases that matter. Calling the model only when a check fires recovers 21 *pp* of that lost ground on the judgment-required scenarios (14 *pp* overall), gives up 8 *pp* overall against always calling it, and does so with 63% fewer calls and a third of the latency. Routing and answering can be scored separately. Of the 14 scenarios in which the rule's default was wrong, the gate sent the model in and the model fixed 6; in 3 more the model would have been right but no check fired; in 5 neither was right. In none of the 13 where the rule was right did consulting the model make it wrong. The misses are directives the checks do not yet recognize.

The plainest alternative. The checks already read the directive: a threshold, a target to skip, a no-imaging or hold clause, Sun in eclipse. Once that is read, a planner can act on it without a model, so we wrote one and froze it before running it. When a check fires, the offending option is removed and the rule tree re-runs in its own priority order, holding nadir where it used to abstain. It has no gazetteer. It scores 23 of 27 (Rule+ in Table 2), and 9 of 10, 9 of 20, and 4 of 5 on the held-out sets of Sec. 4.2. Its misses are of two kinds: a place name that needs coordinates, and a wording the patterns do not match (a hyphenated target name defeats the skip pattern). That is what the model buys on this battery, and both are jobs for a ground tool, with a gazetteer and a person reading the parse before uplink.

Table 3: Five simulated hours on seed 7, identical cloud draws: the production model under both grammars, and Rule+ with follow-up bookkeeping.

	LLM min.	LLM per-tool	Rule+
Accepted by the API	94%	100%	100%
Mission targets imaged usable	2 of 6	2 of 6	5 of 6
Captures queued / transmitted	2 / 2	3 / 3	5 / 5
Bad-argument rejections	37	0	0
Post-model vetoes	7	1	0
Model calls	593	593	0

In the loop. The battery scores one decision at a time, so we also ran Rule+ through the first five hours of the mission of Table 1, same seed and same cloud draw at every capture, against the production model under both grammar tiers (Table 3). Rule+ imaged and downlinked five of the six targets; the model managed two. Near-perfect acceptance and poor output coexist because a planner that re-images the target it already has, and never retries the ones it lost, is doing nothing the dispatcher can object to. The per-tool grammar removed every bad-argument rejection and left the gap, so what remains is not a decoding problem. The model retried Charlie through three cloudy captures until it got a clear one, then spent four hours pointing at Charlie again, never re-imaged Alpha, Delta, or Echo after cloudy first captures, and got no capture of Foxtrot at all. Rule+ had a failure of its own. With no record that it had already added the smoke-

plume follow-up target, it proposed it on every tick for 3.8 hours, until one line treated an anomaly as handled once a target named for it exists.

When the model gets worse. Prompt length matters more than prompt content for a 4 B model. Padding the system prompt with irrelevant but plausible text shows three regimes. Up to about 8 KB the model does well on its own (78 to 81% when always called). From 9 to 12 KB the router does as well or better (72% vs 68% at the 9.3 KB prompt the production system used to run). At 16 KB, the edge of the context window, the always-called model collapses to 11% while the router holds the rule baseline of 48%, because the rules do not care how long the prompt is. The same shape holds across five models from 1 to 12 B: always-on accuracy runs from 26% (Gemma 1 B) to 89% (Gemma 12 B), and the router never falls below the rules. At 1 B it scores 59% against the model's own 26%, so even a model too weak to run alone adds something behind the gate. At 8.9 s per call on the laptop, the 12 B model is a preview; it is not a flight option.

4.2 Showing the model examples, and when not to

Some of the model's mistakes come from unfamiliarity. It had never seen a thermal-cooldown instruction, so it kept imaging, and about 1 KB of worked examples for that family of situation fixes it. Always-on examples lifted content-aware accuracy from 70% to 74% but broke four scenarios that had been working, because the model imitated the examples where they did not apply (the pattern-dominance effect of [13]). Examples shown only when the gate fires reached 78% with nothing broken, at which point the 4 B model matched the 12 B model's content-aware accuracy at a third of the latency.

To check this was not tuned to the battery, we wrote new scenarios after freezing the system. On the first set of ten, the family that had been failing went from 1 of 25 runs to 25 of 25, and a fifth check written for two hold scenarios there lifts the router to 89%, 24 of 27. On a second set of twenty spanning four families, the gate picked the right examples every time with no false triggers and the failing family went from 40% to 100%. We had reported the other three families at ceiling as well. They were not, because a scoring predicate for the two region families returned a value the scorer read as a pass. Re-scored from the raw outputs, two families are at 100% and the two region families at 10 and 12 of 15 runs; the surviving failures are coordinates and target identity, the mechanisms of Sec. 4.4; one is a strict reading of our own naming rule. Over the twenty the model is right on 17 at every seed. A third set of five, written for the post-model veto, scores 4 of 5; the fifth is a priority-order call on which the model and the rule tree agree with each other and not with us.

4.3 Keeping the model's output usable, and what the safeguards still miss

What went wrong in a long run. A one-hour logged run of the production loop showed a 5% rate of unusable replies; a four-hour run showed what that number was hiding. After about 57 minutes the model fell into a rut. It kept emitting the same action under the wrong JSON key (`action` instead of `tool`) on every one of the remaining 360 ticks. The intent was right, the output was unusable, no further imaging goals were pushed, and the mission stalled. Rejected replies never enter the recent-actions block, so the model faced nearly the same input on every tick and kept producing the same shape. Why the key flipped the first time we do not know.

Enforcing the output format. The fix is to make that output impossible rather than ask the model not to produce it. We decode under a two-tier JSON grammar [14], the overall shape (`tool`, `args`, `thought`, nothing else) and, per tool, the allowed argument names and ranges. On the twenty held-out scenarios, across three decoding modes, three seeds, and the four planner modes of Table 2 (720 runs), no decoding mode changed a single outcome and latency was within 7%; the guarantee costs nothing, and the rationale field stays free text. The ranges do nothing in our serving stack, which enforces bounds on integers but not on numbers, so `lat_deg: 200` can still be emitted; the dispatcher's range check rejects it.

Three replicates, three tiers. We then took the planner as it existed before this analysis, switched on the three defaults it recommends (grammar-constrained decoding, gated examples, the post-model veto), and re-ran the four-hour mission three times each way with the same orbit, targets, and cloud draws (Table 4). The seed sets only the initial tumble and the estimator noise, so the three runs are replicates of one mission, and their spread measures the planner's own run-to-run variation. The legacy planner's collapse is intermittent. Across six legacy runs, three under the earlier cloud seeding and these three, it fell into the wrong-key rut in three, at 57 minutes, 57 minutes, and 64 minutes, and never recovered once in it; the other three ran four hours without it. Grammar decoding makes that state unreachable. Parsed, accepted, and useful are three different outcomes. Under the minimal grammar every reply parsed, but 7 to 8% carried extra argument keys and were rejected by the API, and in one earlier run the model echoed the spacecraft state into the argument object until the reply hit the 400-token cap, so that 86% parsed was 12% accepted. The per-tool tier, which fixes each tool's arguments to its named fields, removed both failure modes (100% accepted); it should be the

default. The example gate and the veto never fired, as expected for directives outside the gated families, so grammar decoding carried the difference. Whether an accepted action was useful is the question Table 3 answers.

Table 4: Four simulated hours, three replicate runs (seeds 7, 8, 9; same orbit, targets, and cloud draws; about 472 ticks each): the legacy planner, the design rules with the minimal grammar, and with the per-tool grammar.

	legacy	**minimal**	**per-tool**
Replies parsed	100/100/100%	100/100/100%	100/100/100%
Accepted by API	92/92/92%	92/93/92%	100/100/100%
Rut onset	none	none	none
Usable captures	2/1/1	2/1/1	2/2/2

Worst case. A flight lead will ask what the worst thing the model can emit is. We replaced only the model call in the production planner and drove the real parser, verifier, dispatcher, and simulator with fifteen faults, from a timeout to a prompt injection to imaging under a critical battery. They sort into three outcomes. Failed and malformed replies, the cut-off state echo included, are no-ops that keep the previous goal, and planning resumes on the first good reply; that also means a timeout during a critical battery keeps an imaging goal active, because nothing hands the vehicle to the rule tree. Invalid arguments, unknown targets, out-of-range coordinates, and invented tools such as `fire_thrusters` are rejected by the dispatcher. Well-formed actions that are operationally wrong execute: a `clear_goals` from a prompt injection, which the grammar would not have let the model produce but the dispatcher accepts, left the vehicle without a pointing goal until the next tick, and imaging under a critical battery and under a no-imaging directive were both accepted, since the directive check sits in front of the model and no battery check exists behind it. Because every tool acts on the plan, the next tick can correct each one, though the energy and the window spent meanwhile are not refunded. The intended authority order on a flight vehicle is FDIR and safe mode, then the checks, then the whitelist, then the model; the simulation has no FDIR, so the first layer is untested here.
The model cannot tell when it is unsure. The alternative to a deterministic veto is letting the model abstain when unsure, as in KnowNo [15]. On 81 calibration calls (69 good actions, 12 bad), the bad actions were *more* confident than the good ones by sequence log-probability (median −0.295 vs −0.358), and verbal elicitation gave the same inversion (9.32 vs 8.74 log-odds). The overconfidence is in the judgment itself, so a deterministic check has to play the abstention role.

4.4 Where the model's judgment gives out: places, counts, and repeats

Two episodes from development set the pattern. Asked to image five points off the California coast, the planner added five targets within a few kilometres of one another. It satisfied the count and missed the intent. Asked to image conflict zones, it added Syria five times under slightly perturbed coordinates and fresh names, and told not to repeat itself it produced Syria, eastern Ukraine, western Russia, and then Syria twice more. The logged battery shows the same thing. On the scenario that asks for a third conflict-zone site with Syria already imaged, every model at every seed (20 of 20 runs across four models) re-added Syria under a name such as `Conflict_Syria_W_E`. Our accept set rejected only the exact prior name, so those runs scored as passes; read strictly, that scenario fails for every mode that consults the model, and LLM-on would be 13 of 19 and content-aware 10 of 19. The model knows what a conflict zone is; what it lacks is bookkeeping. The fix is in the tools, not the prompt. The add-target tool refuses a name that already exists, the rendered turn lists every target added under the current directive, and the multi-site family of gated examples shows what five samples spread along a coastline look like. A minimum-spacing check is the obvious next addition. Choosing which region matters is the model's job; counting and spacing are the rule tree's.
Asked to image a named place, the model identified the intended region on all 30 places at three seeds, but then had to supply coordinates, and there it is unreliable. Twenty of the thirty coordinates were within 50 km, and three were more than 500 km off, identical at every seed (Cape Horn placed 1300 km toward Antarctica, Murmansk 850 km too far south, Fiji flipped across the antimeridian). Famous cities were perfect; landmarks 3 of 10. The lesson is the first boundary question again. Let the model decide *which* place matters, and look the coordinate up in a gazetteer. Nor did the model handle reworded geographic directives better than a keyword matcher (16 vs 18 of 24).

4.5 Does perception give the planner what it needs?

The planner's downlink decisions rest on one assumption: that the on-board classifier can say, with usable confidence, what is in a capture. The content-driven compression policy spends bandwidth on a region when the classifier is

confident, and for that to pay off the confident flags need to be right about 80% of the time. The numbers in this section are therefore recall at 80% precision; AUROC can look good while the top of the ranking is polluted. The question is narrow: does any method clear that bar, and if not, what moves it?

Protocol. The imagery is deliberately the hard case: 1024-pixel Maxar tiles at half-meter resolution from the xBD disaster dataset [16], before and after six disaster types, labeled for the six damage atoms by event type and pre- or post-event status; 192 training tiles from four types, 34 held-out tiles from the two types never seen in training. In-distribution claims use scene-disjoint splits that hold out whole disasters, because a random split leaks adjacent tiles of one event and once inflated a result from 37% to 98.5% before we caught it. The open-world test uses 32 never-seen disaster tiles against 180 ordinary scenes [12]; the two sets come from different sources, but a control with the same scene's pre-event tile as the negative (24 pairs, three models, two prompts) gave the same chance-level result. The burn-scar tests use 97 six-band Harmonized Landsat-Sentinel chips at 30 m. HIGH and MEDIUM are the model's own discrete confidence levels, with no threshold to choose; for a continuous score the 80% operating point is read from the held-out precision-recall curve and is therefore an upper bound on what a threshold fixed in advance would achieve. Gemma 3 resizes inputs to 896 pixels square (about 0.6 m per pixel here); a pan-and-scan variant that feeds high-resolution crops raised recall but not precision. Table 5 collects the results.

Table 5: What a small model can and cannot see in overhead imagery, and what helps. Recall is at the 80% precision the downlink policy needs unless noted.

Task	Method	Result	What it says
Grade damage in a new disaster of a type seen in training	small ImageNet CNN, fine-tuned; frozen natural-image DINOv2 + linear head	0% recall (AUPRC 0.39; 0.31)	labels do not fix a backbone trained on the wrong images
	frozen DINOv3-sat (satellite-pretrained) + linear head	36.7% (AUPRC 0.55)	what the backbone was trained on is what matters
Grade damage in an unseen disaster type	any of the above	0%	you have to have labeled that kind of event
Same, 4 B VLM fine-tuned on single post-event images	QLoRA, two runs of the same recipe	0% and 58.6%@40%	unstable: the two runs failed in opposite directions
Same, 4 B VLM fine-tuned on before/after pairs	QLoRA, two runs	100% recall@41–43%; 44.8%@80% reading the logits only	stable; needs a stored reference, so planned monitoring of known sites
Is this overhead scene unusual? (32 real disaster tiles vs 180 ordinary scenes)	zero-shot 4 B / 7 B / 12 B VLM	AUROC 0.41 / 0.49 / 0.43	chance at every size and prompt, even in the raw logits (0.51)
	distance from a bank of normal scenes, frozen DINOv3-sat	0.68; after>before in 36/41 scenes	a scene-change flag, strong for wildfire (0.99), useless for earthquake (0.25)
Find burn scars in 30 m six-band imagery	normalized burn ratio, no training	0.92 macro AUROC	the right bands beat any RGB model
	Clay v1.5 six-band backbone + 60 labeled chips	0.99; 85.2%@80% (RGB probe: 27%)	the one result that met the precision bar

The grader does not clear it. The pretrained 4 B vision path is well calibrated on tiles like its training set but almost never confident. It gave zero HIGH-confidence gradings in 360, and on the unseen disaster types zero recall at HIGH and MEDIUM. With a perfect classifier the compression policy would deliver two to three times as many useful images per pass; with this one, nothing over a fixed codec.

What moves it. Three things did, one of them only for sites already being watched, and none of them is model size. First, what the vision backbone was trained on. Fine-tuning the 4 B VLM on the 192 tiles with quantized low-rank adapters [17] was unstable, and a 12 B generalist lost to a 7 B model trained for fine detail; the clearest signal came from frozen backbones with a linear head, where only the satellite-pretrained DINOv3-sat [18] crossed the bar on new disasters of a seen type, and an ImageNet CNN and a natural-image DINOv2 found nothing, fine-tuned or not. No single-image method generalized to a disaster type absent from training. Second, before-and-after pairs. Training the

4 B model on paired images, with the before-image teaching it to say "no damage," replicated across two runs and is the only configuration in which the 4 B model itself clears the bar on unseen disaster types, when its next-token probabilities are read instead of its text. Using it on board assumes a stored high-resolution reference for each site, plausible for a fixed watch list and not for arbitrary scenes; it is a result for planned monitoring of selected targets. Third, bands. On the 30 m chips a three-line normalized burn ratio on the SWIR bands finds burn scars where a 300 M-parameter RGB foundation model is at chance, and with six-band input and 60 labels the Clay v1.5 backbone [19] is the one configuration that met the precision bar. It is for a different task. A perception layer for a fire-monitoring mission can be built to the bar; the damage classifier is not fixed. The index must match the phenomenon. A variant with higher AUROC scored 0% at the strict bar, so the operating point is what matters.

What does not move it. The use we most wanted, a model that looks at a fresh tile and says "this looks unusual, send it down," does not work with any VLM we tried. On the 32 disaster tiles against 180 ordinary scenes, the 4, 7, and 12 B models all scored at chance and failed in opposite directions (the 7 B called real floods in Brazil and Italy "normal scene"; the 4 B flagged two-thirds of the ordinary scenes). The best substitute is a training-free score of how far a tile's features sit from a bank of ordinary scenes on the satellite backbone. After-images out-score their own before-images in 36 of 41 scenes, but it is a radiometric change detector, excellent for wildfire scars and useless for earthquake damage, and it cannot say where in the tile the change is. A change head trained only on pre-disaster building footprints localizes destroyed buildings (0.83 for tsunami) but not roof-level damage (0.52 for hurricane).

5. Limitations

Table 6 collects the design rules the results support; two things bound them.

It is all simulation. That is what made the ablations possible (a flight cannot replay a seed with the rules toggled), but radiation effects on inference, real thermal and power behavior, link budgets, and operator-loop dynamics are unmeasured. Every latency here was measured on a laptop, and the energy estimate in Table 6 applies an assumed Jetson power draw to those latencies; flight-class hardware is next. The logged runs are synchronous. The simulation pauses while the model thinks, so inference latency is not charged against mission time. Planner ticks already take about 6 s with the simulation sharing the CPU, which argues for a dedicated accelerator and for wiring the hand-off to the rule tree that Sec. 4.3 shows is missing. The scenario batteries are small (27, 20, 10, and 5 members) and the model is deterministic at our temperature, so results are counts without confidence intervals. One scoring bug (Sec. 4.2) survived four months and an independent audit of our pass/fail calls, which is an argument for releasing raw outputs. The perception results rest on tens of tiles and six events. The four-hour comparison is three seeded runs, and the closed-loop one a single matched mission.

One mission. The rules should carry over to fault narration and mission logging; they are demonstrated only on Earth-observation tasking.

Table 6: Seven design rules for a small on-board language model, each with the measurement behind it.

	Rule	What we measured
P1	Do as much as possible deterministically; ask the model only where a rule cannot read what the decision depends on; parse operator language on the ground.	The model helps only on judgment-required decisions; Rule+ scores 23 of 27 against the router's 24 and delivers 5 of 6 targets against 2 in the loop, with no model; about 9 Wh/day saved at a 30 s cadence (estimate).
P2	Let one deterministic gate decide whether to ask, what examples to show, and whether to accept.	One gate handled routing, examples, and scaffolds across four families.
P3	Bound failures with checks, not with the model's confidence.	Wrong answers rank as more confident at 4 B, by log-probability and by asking.
P4	Enforce the output format at decode time, arguments included.	An intermittent, unrecoverable collapse into malformed output in three of six legacy runs, unreachable under grammar decoding; the per-tool tier also removes the 7 to 8% of replies with stray argument keys.
P5	Design for the model getting worse, and for it getting better.	The router holds the rule floor as the model collapses at long prompts or shrinks to 1 B; four of five 4 B failures vanish at 12 B.

	Rule	What we measured
P6	Keep perception out of the language model; use a swappable specialist.	Zero confident recall out of type; zero-shot triage at chance at every size; only a satellite-pretrained backbone and the right bands crossed the bar.
P7	In-context examples work, but only when gated.	Always-on examples broke four working scenarios; gated ones fixed the failing family held-out, nothing broken.

6. Conclusion

A 4 B language model can plan for a small Earth-imaging spacecraft, given a narrow job, a reversible set of tools, and a deterministic gate that decides when to ask it, what to show it, and whether to believe it. On this mission, though, the model did not add much. A rule tree that applies what the gate's checks read out of the directive came within one scenario of the hybrid on the battery and beat it in the loop, five targets delivered against two, with no model at all. What the model still contributed belongs on the ground, where an operator can check the parsed directive before it is uplinked as structured commands. The vision side is weaker still. The model cannot grade damage in a single overhead tile, and what improved its eyesight was a satellite-pretrained backbone and the right bands, never a larger model. Better perception would also make a deterministic planner more useful, and a more capable model would change the numbers without settling whether the model needs to be on board. What is left for an on-board model is the case that motivated the work, an event nobody wrote a rule or a target for, where the right action is to notice that something looks wrong and send it down. Table 6 is how we would put a model up for that, and the measurements here are what to expect of it.

Acknowledgment

An AI assistant (Claude, Anthropic) wrote much of the code, ran the analyses, and drafted and edited some of this text. It has read the conclusion and notes that it is not a small model.

References

[1] Gemma Team, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.
[2] P. McKeen, N. Scheuer, and K. Cahoy, "A modular open-source ADCS framework for small satellite development and testing," in *Proc. SmallSat Europe*, Amsterdam, 2026.
[3] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, and A. Murthy, "Position: LLMs can't plan, but can help planning in LLM-Modulo frameworks," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2024.
[4] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *IEEE Trans. Systems, Man, and Cybernetics – Part A*, vol. 30, no. 3, pp. 286–297, 2000.
[5] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams, "Remote agent: To boldly go where no AI system has gone before," *Artificial Intelligence*, vol. 103, no. 1–2, pp. 5–47, 1998.
[6] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castaño, A. Davies, D. Mandl, S. Frye, B. Trout, S. Shulman, and D. Boyer, "Using autonomy flight software to improve science return on Earth Observing One," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 4, pp. 196–216, 2005.
[7] G. Giuffrida, L. Fanucci, G. Meoni, M. Batic, L. Buckley, A. Dunne, C. van Dijk, M. Esposito, J. Hefele, N. Vercruyssen, G. Furano, M. Pastena, and J. Aschbacher, "The Φ-Sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation," *IEEE Trans. Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022.
[8] J. M. Delfa Victoria, T. C. John, and A. W. Herson, "NAVI-Orbital: First in-orbit demonstration of a zero-shot vision-language model for autonomous earth observation," *arXiv preprint arXiv:2606.18271*, 2026.
[9] A. D. Mousist, "ASTREA: Introducing agentic intelligence for orbital thermal autonomy," *arXiv preprint arXiv:2509.13380*, 2025, accepted at ESA AI STAR 2025.
[10] M. Ahn, A. Brohan, N. Brown *et al.*, "Do as I can, not as I say: Grounding language in robotic affordances," in *Proc. Conf. on Robot Learning (CoRL)*, 2022.
[11] P. McKeen, N. Scheuer, and K. Cahoy, "Generalized attitude control for small spacecraft," in *Proc. 40th Annual Small Satellite Conference*, Salt Lake City, UT, 2026, SSC26-P2-54.

[12] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *Proceedings of the IEEE*, vol. 105, no. 10, pp. 1865–1883, 2017.
[13] W. Rudman, M. Golovanevsky, D. Arad, Y. Belinkov, R. Singh, C. Eickhoff, and K. Mahowald, "Mechanisms of prompt-induced hallucination in vision-language models," in *Proc. 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026, arXiv:2601.05201.
[14] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-constrained decoding for structured NLP tasks without finetuning," in *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
[15] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, Z. Xu, D. Sadigh, A. Zeng, and A. Majumdar, "Robots that ask for help: Uncertainty alignment for large language model planners," in *Proc. Conf. on Robot Learning (CoRL)*, 2023.
[16] R. Gupta, R. Hosfelt, S. Sajeev, N. Patel, B. Goodman, J. Doshi, E. Heim, H. Choset, and M. Gaston, "xBD: A dataset for assessing building damage from satellite imagery," in *Proc. CVPR Workshops*, 2019.
[17] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Advances in Neural Information Processing Systems*, 2023.
[18] O. Siméoni *et al.*, "DINOv3," *arXiv preprint arXiv:2508.10104*, 2025.
[19] Clay Foundation, "Clay foundation model v1.5," github.com/Clay-foundation/model, 2025.