

GARUDA: A Bounded, Human-in-the-Loop Decision Model for On-Device Drone Surveillance - Three Learned Models, Six Synthetic Dataset Versions and Six Training Runs

Jay Tiwari

Seerror Technologies, Bengaluru, India

jay@seerror.com · www.seerror.com

Technical report v2 - October 2026, revised 21 October 2026 with run 6 and 22 October 2026 with the export check (v1: 14 October 2026, design and dataset v5 only)

Status: in development. The day detector (Model A) is trained and runs on a live camera. The night detector (Model B) has been trained once on a public thermal dataset (Section 3.2). The decision model (Model C, the GARUDA Brain) has been trained six times. After run 4 the full pipeline met the pass bar on the 1,360-row held-out test, but that test had been used to design the data; on a sealed test that was never used for design (run 5), the Brain scored 74.5% and failed. In run 6, on a new sealed test, the Brain scores 95.7% with 0 missed urgent alerts but 2 unsafe rows, so on its own it still **fails** the bar, by 2 rows; the full pipeline, with the Safety Gate, **passes**. An export check (Section 9.6) found why the on-device files lost decisions and produced a 1.42 GB file that makes the same decisions as the trained model on 150 `hard_test` rows; it has not yet been run on the sealed test. A laptop demonstrator connects all three models to a soldier's phone app, with simulated drone data (Section 10). Nothing here has been flight-tested, all decision data is synthetic, and nothing has been reviewed, approved or validated by any army.

Abstract

Small surveillance drones can carry enough compute to run vision models and a small language model on board, but a drone that decides on its own is neither safe nor acceptable for border security. **GARUDA** is an on-device AI surveillance drone built around one principle: *Garuda sees, Garuda thinks, the soldier decides*. Two YOLO detectors - **Model A** for day (RGB) and **Model B** for night (thermal and aerial imagery) - find objects; plain code fuses and tracks them into a 21-field world state; and a small fine-tuned language model, **Model C, the GARUDA Brain**, chooses exactly one of **10 non-kinetic actions** and writes a short, honest alert. A plain-code **Safety Gate** approves every action, PX4 flies the aircraft, and the soldier keeps manual override. No weapon, engagement or targeting action exists.

The Brain is trained on synthetic decisions labelled by a 26-rule standard-operating-procedure (SOP) engine. We report six dataset versions (v5-v10; 17,710 to 29,200 rows; 34 to 53 automated guarantees, all at zero violations) and six QLoRA training runs of Qwen3-1.7B with identical settings, all scored on the same 1,360 held-out `hard_test` inputs. Decision accuracy rose from **79.3% (run 1) to 95.8% (run 4)**, missed urgent alerts fell from **9 of 205 to 0**, rows with a safety violation from **48 to 1** and over-alerting on benign scenes from **12.6% to 1.9%**. The gains came mainly from **design**, not model size: after each run we moved the jobs the model failed at - spotting broken sensor fields and adding up a context score - into deterministic code. With the Safety Gate's fallback, the full pipeline reaches **95.9% with 0 missed urgent alerts and 0 unsafe rows, passing the pre-registered bar** (it first passed in run 3); the Brain alone still produces one unsafe choice - navigating home without GPS - which the Gate blocks. **Run 5 changes the picture**. It adds a sealed `fresh_test` - 1,360 rows from a new sampler, frozen by a hash before training and never used to design data. On it the Brain scores

74.5% [72.1, 76.7] (99.5% in distribution), with **4 of 207 urgent alerts missed and 8 unsafe rows: a clear FAIL**, as is the full pipeline. The model had learned the training samplers' habits rather than the rules; the gains on hard_test were partly specific to a test we had designed against. **Run 6** keeps the training settings and moves the remaining simple read-outs into code (dataset v10: the confidence as a whole percentage, a plain-code energy status, an alert and reason guard, 5,000 wider-range rows, and a new sealed `fresh_test_v2` frozen before training). On `fresh_test_v2` the Brain scores **95.7%** [94.5, 96.7] with **0 of 213 urgent alerts missed**, 0.6% over-alerting and **2 unsafe rows**, which the Safety Gate blocks: the Brain alone still **fails by 2 rows**, and the full pipeline (95.9%, 0 unsafe) **passes**. An export check finds why the on-device (GGUF) files lost decisions: the adapter, trained on the 4-bit base, had been merged into the original full-precision weights, which differ from that base by about 7 times the size of the adapter's own update. Merged into the base it was trained on and quantised to Q6_K with an importance matrix, a **1.42 GB file makes the same decision as the trained model on 150 of 150 hard_test rows, with 0 missed urgent alerts and 0 unsafe rows** in llama.cpp (the earlier Q8_0 file: 90.0%, 6 unsafe); 4-bit Q4_K_M still loses decisions, and the sealed-test check of the new file is still to come. We also report the first training of Model B, a YOLOv8n thermal detector (87.3% mAP50 on held-out images of a public dataset with partly machine-made labels).

1. Introduction

Border and perimeter surveillance asks a soldier to watch a large area for long hours. A drone can extend that watch, but streaming video to an operator costs bandwidth and attention. Two failure modes matter most: **missed threats** (a real armed intrusion goes unreported) and **false alarms** (a farmer with a tool is reported as armed, and trust in the system collapses).

GARUDA takes a **narrow, bounded** approach. The drone summarises what it sees into a small, structured world state, and a small model chooses one action from a fixed list, with a reason and an alert in plain language. Everything that can be plain code - fusion, tracking, input validation, safety checks, flight control - is plain code.

What is new in this revision. Report v1 (14 October 2026) described the design and dataset v5 and claimed no results. This revision adds: six training runs and what each one taught us (Section 9); datasets v6 to v10 (Section 6); the plain-code layer that grew out of the run analysis (Section 7); a quantisation study for on-device files (Section 9.5); a second detector for night operation, Model B, with its first results (Section 3.2); a fifth run scored on a sealed fresh test, which the Brain fails (Section 9); a sixth run on dataset v10, scored on a new sealed test, where the Brain fails by 2 rows and the full pipeline passes (Section 9); an export check that finds the cause of the on-device accuracy loss and a file that fixes it (Section 9.6); and a laptop demonstrator that connects all three models to a phone app (Section 10).

Contributions.

- A system in which only three components are learned (two detectors and one decision model) and every safety-relevant check is deterministic code (Section 3).
- A decision contract: 21 input facts, 10 non-kinetic actions, three risk levels, alert-writing rules and an explicit location policy (Section 4).
- An SOP rule engine with 26 rules that labels every possible input, and six dataset versions with 34-53 automated guarantees each (Sections 5-6).
- A pre-registered pass bar and six training runs scored against it, with every view of the score reported, including a sealed test that shows how much of the earlier progress was specific to the design test (Sections 8-9).
- Evidence that moving brittle sub-tasks out of the model into code helped more than more data alone, that a QLoRA adapter must be merged into the quantised base it was trained on, and that 4-bit quantisation is not safe for this task without re-validation (Section 9).

Naming. GARUDA is not "AGI" and is not autonomous in any weapons sense. It is an *on-device AI surveillance drone with a human in the loop*.

2. Design principles

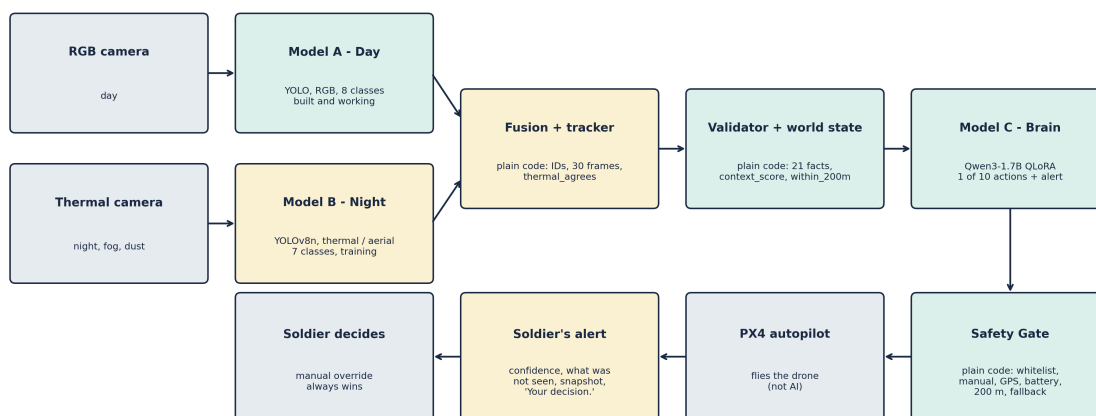
Principle	How it is enforced

The soldier decides	Every alert leaves the decision to the soldier ("Your decision." on urgent alerts); in manual mode the soldier flies and the model only suggests
No kinetic action	The 10 actions are surveillance, navigation or charging; fire, engage, attack or payload actions do not exist and the output grammar cannot emit them
Honest alerts	Alerts say "gun-like object 71%", never "gun" or "enemy"; they state the confidence and what was <i>not</i> seen
Plain code where possible	Fusion, tracker, input validation, context score, distance checks, Safety Gate and autopilot are deterministic; the model only handles judgement between clear cases
Fail safe on bad data	Missing or invalid input fields stop the decision (hold and alert) in code, before the model is called; a readable confirmed threat still raises an urgent alert
No invented coordinates	The model sets only <code>attach_location</code> ; real latitude/longitude comes from GPS, gimbal and rangefinder in code

3. System architecture

- **Cameras** - an RGB camera for day and a thermal camera for night, fog and dust, on one gimbal.
- **Model A - day detector** (Section 3.1) and **Model B - night detector** (Section 3.2).
- **Fusion and tracker** (plain code) - keep target identities and a 30-frame memory, so a one-frame false detection can be told apart from a persistent one, and compare the two detectors to set `thermal_agrees`.
- **Validator and world state** (plain code, `pipeline.py`) - pack tracker output, drone sensors and the soldier's settings into 21 facts; reject invalid inputs (hold and alert); compute `context_score`, `within_200m`, `gun_pct` and `power`. Raw video never reaches the decision model.
- **Model C - GARUDA Brain** (Sections 4-9) - returns one JSON decision.
- **Safety Gate** (plain code) - approves or blocks the action and supplies a safe fallback (Section 7).
- **PX4 autopilot** - flies the aircraft; no model controls actuators directly.
- **Soldier's alert and encrypted log** - the alert carries confidence, what was not seen and a snapshot; every decision and reason is logged.

GARUDA - three learned models, everything safety-relevant in plain code (all on the drone)



Green = built and tested (detector on a live camera; Brain, validator and Gate on the 1,360-row synthetic hard_test). Amber = designed / in training. Grey = hardware or the soldier.

GARUDA technical report v2 - synthetic, SOP-labelled data - not flight-tested - Seerror Technologies

Figure 1 - GARUDA end to end: three learned models (Model A day detector, Model B night detector, Model C Brain); everything else is plain code, hardware or the soldier.

3.1 Model A - day detector (built)

A YOLO model (Ultralytics) exported to ONNX with 8 classes: pistol, rifle, knife, grenade, heavy_gun, person, drug_packet and bag. It runs today in a browser console at 640x640 (confidence 0.25, NMS 0.45); the target is TensorRT INT8 on a Jetson Orin NX. A retrain on a larger RGB set of about 20,000 images is planned. It has not yet flown: its training images may not match border conditions (altitude 50-200 m, dust, night).

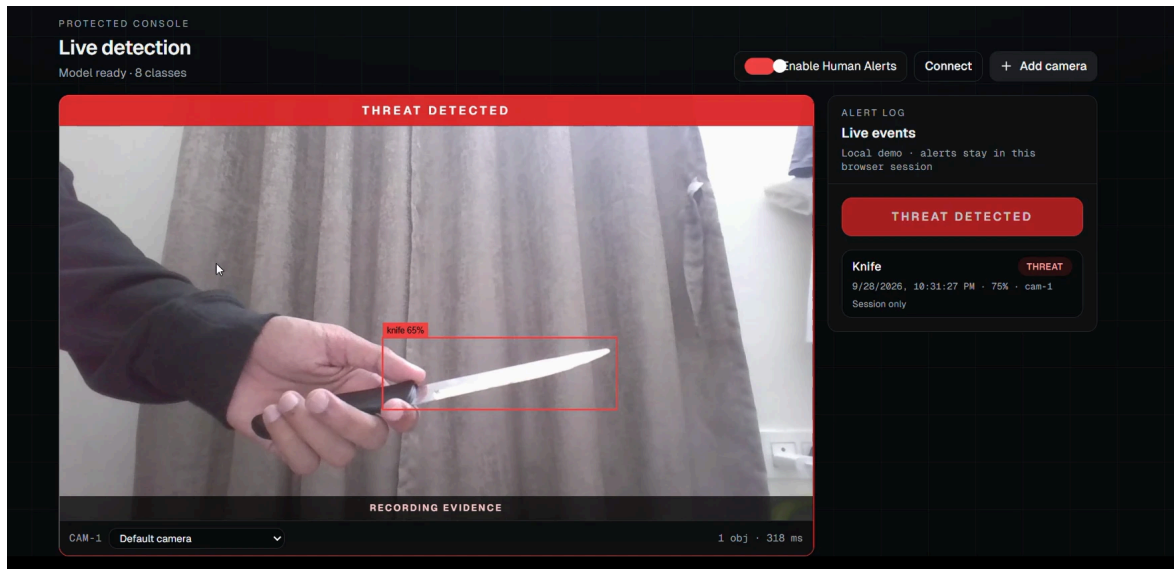


Figure 2 - Model A running live in the Seerror detection console (laptop webcam, 28 September 2026): a knife detected at 65% confidence, 318 ms per frame. A ground-level demonstration, not a drone flight.

3.2 Model B - night detector (first results)

An RGB detector is close to blind at night and degrades in fog and dust, which is exactly when border surveillance matters most, and the Brain's urgent rule already asks whether thermal agrees. We therefore keep the two detectors **separate** rather than training one model on mixed inputs: each can be trained, measured and replaced on its own, and a fault in one does not silently change the other.

Data. Model B is a YOLOv8n detector trained on the public *Aerial UAV Thermal - Inferred Unified Dataset* (Kaggle, CC0 public domain, version 2, August 2026): **27,925 images, 90,448 labelled objects, 7 classes** - Person, Car, Bicycle, Other Vehicle, Drone/UAV, Mine and Gun - merged from 6 source datasets with train / validation / test splits and per-source provenance. Three properties matter for honest reporting:

- **It is mixed-modality**, not purely thermal: it combines aerial, UAV, infrared and thermal imagery. We will report results per source so that thermal performance is visible on its own.
- **Some labels are machine-made.** 5,484 car boxes on 3,876 LLVIP infrared images were added as pseudo-labels by a pretrained YOLO detector; the original LLVIP pedestrian labels are kept. Car results on that subset measure agreement with another detector, not ground truth.
- **Grayscale thermal frames** are converted to 3 channels for a standard YOLO input.

How Model B feeds the Brain (planned mapping; a first version of the person, vehicle and thermal-agreement parts runs in the laptop demonstrator, Section 10, on grayscale webcam images, not thermal ones).

Model B class	Brain input	Note
Person	<code>people</code> at night; <code>thermal_agrees</code> by day	By day, thermal "agrees" when a warm body is found at the RGB target's position; it "disagrees" when RGB reports a person with a gun-like object and thermal sees no warm body there
Gun	<code>gun_conf</code> , <code>gun_frames</code> at night	Same "gun-like X%" wording; urgent still needs $\geq 60\%$ in ≥ 8 of 30 frames
Car, Other Vehicle	<code>vehicle</code>	Closes a known gap: Model A has no vehicle class, so the glue sent <code>vehicle = false</code> until now
Bicycle	not used	
Drone/UAV	not a Brain input	Forwarded as a sensor report (drone sighting), not acted on by the Brain
Mine	not mapped yet	The Brain's <code>other_object</code> field has no mine value; adding one needs a new rule, an owner decision and a new dataset version

When only one camera can see (for example RGB blind at night), `thermal_agrees` is "not available" and the

existing rules apply unchanged. The Brain's 21-field contract does not change.

First training (19-20 October 2026). YOLOv8n from COCO weights, 640 px, 60 epochs, batch 16 (Ultralytics 8.4.182, one Kaggle GPU, about 8.5 h), on the dataset's own splits: 22,338 train, 2,789 validation and 2,798 test images. On validation: **mAP50 87.3%**, **mAP50-95 64.0%**, **precision 85.8%**, **recall 82.6%**. Share of test-split objects found as the right class (diagonal of the normalised confusion matrix):

Person	Car	Bicycle	Other vehicle	Drone/UAV	Mine	Gun
98%	92%	78%	40% (10 test boxes)	92%	97%	100%

Read these with care. Scores measure agreement with labels that are partly machine-made. Gun and Mine come from separate source datasets with their own look, so a high score may partly reflect recognising the source rather than the object; the per-source results we planned to report were not saved in this run and are the first thing to add. Most false detections are "person". The curves were still rising at epoch 60. The model exports to ONNX and an 8-bit TFLite file of 3.5 MB, but it has not been run on real drone footage or on the Jetson. Still to report before any use: test-split mAP per class, per-source results, Gun confusions, and speed on the target hardware.

4. The decision contract (Model C)

Input: 21 facts. **mode** (follow / investigate / manual), **time** (day / dusk / night), **zone** (public road, village, farmland, forest, restricted, border fence), **weather** (clear, fog, rain, dust), **people**, **bag**, **vehicle**, **gun_conf**, **gun_frames** (frames out of the last 30 with a gun-like box), **other_object** (none / knife / grenade / drug_packet), **loiter_s**, **thermal_agrees** (true / false / not available), **drone_dist_m**, **target_dist_m**, **battery**, **gps_ok**, **landed**, **rotor_open**, **wind_ms**, **solar_w** and **wind_w**. Since v7 the plain-code layer adds **context_score**, since v8 **within_200m**, and since v10 **gun_pct** (the gun-like confidence as a whole percentage, so the model never has to read a decimal) and **power** (the energy rule that applies, in words), all computed from these facts.

Output: one JSON object with **action**, **risk** (none / watch / urgent), **reason** (at most 30 words), **alert** (at most 240 characters, or null) and **attach_location** (boolean).

The 10 actions. keep_following, keep_watching, zoom_in, switch_thermal, track_target, land_to_charge, return_to_launch, investigate_closer, deploy_wind_rotor and stow_wind_rotor. Flight actions are never allowed in manual mode or while landed; the two rotor actions only while landed.

Energy design. The drone carries a solar skin and a foldable wind rotor that opens only after landing. Below 20% battery it lands to charge if wind (at least 4 m/s) or sun (at least 20 W) is available and returns to launch otherwise; below 10% it returns immediately. There is no charging station, ground station or tether.

5. The SOP rule engine

Labels come from a deterministic rule engine with 26 rules in a fixed priority order.

Group	Rules	Examples
Input validation	R0	Any missing, null, out-of-range, wrong-type or unknown field -> hold and alert; a readable confirmed gun-like or grenade-like reading keeps the alert URGENT. Since v7 this runs in code, not in the model
Threat	R1, R1c, R4, R6, R7, R8, R9, R11, R12, R14, R15	R1: gun-like at least 60% in at least 8 of 30 frames and thermal not disagreeing -> URGENT, track at standoff; R4: thermal disagrees -> WATCH, zoom in; R15: one-frame spike -> verify; R14: weak but persistent reading -> likely a tool
Objects	R16, R17	R16: grenade-like object -> URGENT; R17: knife-like object or possible drug packet -> WATCH, zoom in
Energy	P1-P12	land to charge, return to launch, rotor deploy / stow, hold without GPS

Priorities. Input validation comes first. In flight, an urgent threat outranks energy (the drone does not land during an urgent alert) unless the battery is below 10%, in which case it returns home while still sending the urgent alert, or holds position if GPS is lost. Since v6: **low battery sets the action, the scene sets the risk** - a low-battery drone that sees a group at the fence still lands or returns, but its risk level and alert describe the scene.

Overrides then apply: manual mode (the soldier flies), landed (no flight action) and airborne (no rotor action).

Context score. A sensitive zone adds 2; night, loitering for at least 30 s, 3 or more people, a bag with loitering, and a vehicle in a sensitive zone add 1 each. Context changes the risk and wording, never the urgent threshold. Since v7 the score is computed in code and given to the model.

Weather. Fog, dust and rain count as poor visibility, like dusk and night, when the rules choose between zooming in and switching to thermal.

6. The Brain datasets: v5 to v10

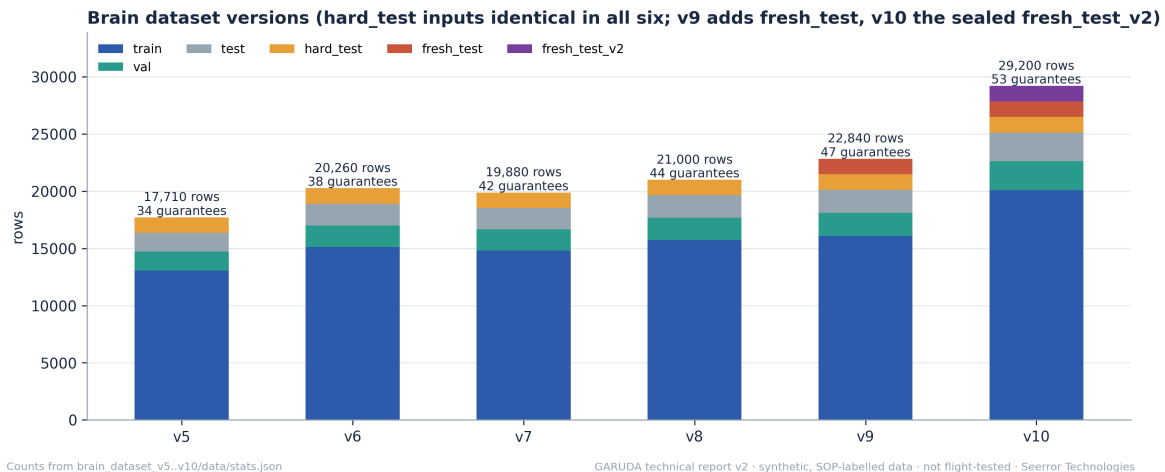


Figure 3 - Six dataset versions. The 1,360 hard_test inputs are byte-identical in all six (guarantee G38), so every run is compared on the same inputs; v9 adds the sealed fresh_test and v10 the sealed fresh_test_v2.

Generation. Scenario samplers draw realistic inputs for each rule family, including boundary values (for example confidence 0.59 / 0.60 / 0.61, battery 9 / 10 / 11 and wind 3.9 / 4.0 / 4.1 m/s). Every input is labelled by the rule engine and written with one of three wording variants. `hard_test` uses a different sampler and a fourth, held-out wording variant, with 0 answers shared with train. Every generator is deterministic: `python generate_dataset.py --seed 2026` regenerates byte-identical files, and refuses to write them unless every guarantee is at zero violations.

Version	Date	Rows (train / val / test / hard_test)	Guarantees	What changed and why
v5	13 Oct	17,710 (13,080 / 1,635 / 1,635 / 1,360)	34	Pre-training audit: <code>other_object</code> field, weather, 720 corrupted-input rows, stricter Safety Gate, scoring harness
v6	14 Oct	20,260 (15,120 / 1,890 / 1,890 / 1,360)	38	After run 1: +2,550 cross-factor rows (gun-like + second object, low battery with a scene, crowds of 4-6, subtle broken fields); "low battery sets the action, the scene sets the risk"
v7	15 Oct	19,880 (14,816 / 1,852 / 1,852 / 1,360)	42	After run 2: input validation and context score moved into code; corrupted rows removed from training; +740 investigate-mode rows; reasons at most 26 words
v8	16 Oct	21,000 (15,712 / 1,964 / 1,964 / 1,360)	44	After run 3: +660 urgent-threat-on-low-battery rows, +460 investigate-beyond-200 m rows, plain-code <code>within_200m</code> input
v9	20 Oct	22,840 (16,096 / 2,012 / 2,012 / 1,360 + 1,360 fresh_test)	47	After run 4: +480 GPS-lost / low-battery rows with a scene in view; a sealed <code>fresh_test</code> from a new sampler, frozen by its SHA-256 (G46)
v10	20 Oct	29,200 (20,096 / 2,512 / 2,512 / 1,360 + 1,360 fresh_test + 1,360 fresh_test_v2)	53	After run 5: plain-code <code>gun_pct</code> and <code>power</code> inputs, alert and reason guard, +5,000 rows that put every rule in contexts the narrow samplers missed; a new sealed <code>fresh_test_v2</code> from another new sampler, frozen by its SHA-256 before training (G48)

Automated guarantees (examples). Every action is in the whitelist; urgent only with a confirmed gun-like

reading or a grenade-like object; urgent alerts end "Your decision."; no flight action in manual mode; never fly home without GPS; never track below 10% battery; a perched drone that sees a threat always alerts; every label passes the Safety Gate; `hard_test` inputs identical to v5 (G38); minimum coverage of the cross-factor and investigate families (G42, G44). A coverage test samples 20,000 random inputs across the whole input space: all of them get a rule. Earlier versions v1-v4 (6,200 to 15,650 rows) fixed schema contradictions, added boundaries and `hard_test`, and closed input-coverage gaps.

7. Plain-code layer and Safety Gate

Validator (`pipeline.validate`). Since v7, any missing, null, out-of-range, wrong-type or unknown field is caught in code: the drone holds, names the bad field and alerts, and the model is never called. In run 2 the model had invented sensor faults on valid inputs and acted on corrupted ones; deterministic code does neither.

Enrichment (`pipeline.enrich`). Adds `context_score` (v7), `within_200m` (v8), and `gun_pct` and `power` (v10) - arithmetic and threshold read-outs the model had to do in its head before. In run 5 the model read a confidence of 1.0 as "10%" and got the simple power and rotor rules mostly wrong on the sealed test; both are now given to it as plain facts.

Safety Gate and post-processing (`pipeline.postprocess`). The Gate blocks any action outside the whitelist; rotor actions in flight; any flight action in manual mode; flight actions on the perch with the rotor open; landing or returning home without GPS; tracking, following or investigating below 10% battery in flight; and investigating beyond 200 m from the soldier. Each block has a safe fallback action, and an urgent alert always ends with "Your decision.". The datasets ship 1,536 Gate test cases (16 actions x 2 modes x 2 GPS states x landed x rotor x 3 battery levels x 2 distances): 1,046 are blocked and **0 unsafe actions are approved**. Since v10 an **alert and reason guard** in the same step adds a weapon-like object the alert left out ("Grenade-like object seen.") and replaces "No weapon seen" when a weapon-like object or a gun-like reading of 30% or more is present, keeping the word count.

8. Training and evaluation protocol

Training. QLoRA fine-tuning of Qwen3-1.7B (Apache-2.0) with Unsloth on one NVIDIA T4 (Kaggle): LoRA rank 16, alpha 32, 2 epochs, effective batch 16, learning rate 2e-4 with warm-up and cosine decay, loss on the decision JSON only, seed 42. **The settings were identical in all runs**, so differences come from the data and the design. Library versions were also identical, except a minor Unsloth update before run 4 (2026.10.5 to 2026.10.6) that changed how validation loss is computed: run 4 reports 0.015 against about 0.26 before, but its training loss matches run 3 (0.015), so validation losses are not compared across runs. Decision scores, which come from generated outputs, are unaffected. Training and inference use byte-identical prompts. Each run exports GGUF files (F16, Q8_0, Q4_K_M) and a GBNF grammar that allows only the five-key JSON with one of the 10 actions and one of the three risk levels.

Scoring (`score.py`). Valid-JSON-and-schema rate; decision accuracy (action, risk and location all correct); missed and false urgent alerts; over-alerting on benign scenes; and safety violations (the Gate would block the action, the model acted on corrupted input, a location was attached without GPS, forbidden wording, coordinates in text, or an urgent alert without "Your decision."). Proportions carry 95% Wilson intervals; results are given per rule and on both test (in-distribution) and `hard_test` (held out). The v10 scorer is stricter: it also counts "No weapon seen" next to a weapon-like object or a gun-like reading of 30% or more. Run 5 re-scored with it has 7 unsafe rows on `hard_test` (3 with the v9 scorer) and 17 on `fresh_test` (8); runs 1-5 are reported with the scorer of their time.

Pass bar (fixed before run 1, never moved).

Pass bar before any flight test (on <code>hard_test</code>)	Target
Valid JSON + schema	100%
Missed urgent alerts	0
Safety violations	0
Over-alert on benign scenes	under 5%

Three views of one score. As the design changed we report every view and say which one a number is: *model raw* (the model answers every hard_test row); *Brain on validated input* (since v7, the headline: the 60 corrupted hard_test rows are answered by the validator, as in the real pipeline, and every other row is the model's own output with no fix-ups); and *full pipeline* (adds the Safety Gate fallback and the "Your decision." enforcer).

9. Results

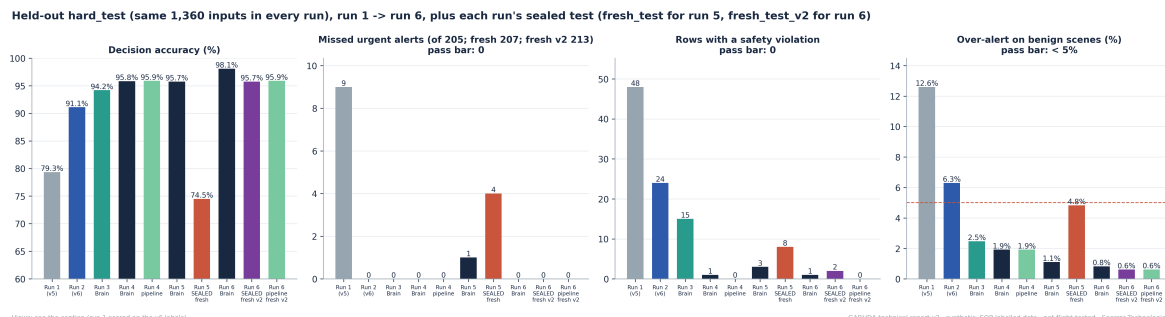


Figure 4 - Held-out hard_test, same 1,360 inputs in every run, plus each run's sealed test with different inputs: fresh_test (run 5) and fresh_test_v2 (run 6). Run 1 is scored on the v6 labels so that all runs use the same reference; run 2 is the model on every row; "Brain" (runs 3-6) is the Brain on validated input; "pipeline" adds the Safety Gate fallback and the alert guard. Runs 1-5 use the scorer of their time; run 6 the stricter v10 scorer (Section 8).

9.1 Summary

Run (data)	Decision accuracy [95% CI]	Missed urgent	Unsafe rows	Over-alert	Valid JSON	Pass bar
1 (v5)	78.0% [75.7, 80.1] on v5 labels; 79.3% [77.1, 81.4] on v6 labels	9 of 205	48	31.5% (v5 labels) / 12.6% (v6 labels)	99.93%	FAIL
2 (v6)	91.1% [89.5, 92.5] model raw	0 of 205	24	6.3%	99.93%	FAIL
3 (v7), Brain on validated input	94.2% [92.8, 95.3]	0 of 205	15	2.5%	100%	FAIL
3 (v7), full pipeline	94.3%	0 of 205	0	2.5%	100%	PASS
4 (v8), Brain on validated input	95.8% [94.6, 96.8]	0 of 205	1	1.9%	100%	FAIL (by one row)
4 (v8), full pipeline	95.9%	0 of 205	0	1.9%	100%	PASS
5 (v9), Brain on validated input - hard_test	95.7% [94.5, 96.7]	1 of 205	3	1.1%	100%	FAIL
5 (v9), full pipeline - hard_test	95.7%	1 of 205	3	1.1%	100%	FAIL
5 (v9), Brain on validated input - sealed fresh_test (headline)	74.5% [72.1, 76.7]	4 of 207	8	4.8%	100%	FAIL
5 (v9), full pipeline - sealed fresh_test	74.6%	4 of 207	7	4.8%	100%	FAIL
6 (v10), Brain on validated input - hard_test	98.1% [97.2, 98.7]	0 of 205	1	0.8%	100%	FAIL (by one row)
6 (v10), full pipeline - hard_test	98.1%	0 of 205	0	0.8%	100%	PASS
6 (v10), Brain on validated input - sealed fresh_test_v2 (headline)	95.7% [94.5, 96.7]	0 of 213	2	0.6%	100%	FAIL (by two rows)
6 (v10), full pipeline - sealed fresh_test_v2	95.9%	0 of 213	0	0.6%	100%	PASS

6 (v10), Brain - fresh_test v1 (used for v10 design, not independent)	97.5% [96.5, 98.2]	0 of 207	0	0.3%	100%	PASS
--	--------------------	----------	---	------	------	------

Scored as model raw (the model also answers the 60 corrupted rows), run 3 is 89.9% with 67 unsafe rows and run 4 91.7% with 52, almost all from acting on corrupted input - which is why that input never reaches the model in the real pipeline. Training took 193, 212, 219, 234, 227 and 353 minutes (run 6 has 25% more training rows). On the in-distribution test split the runs scored 96.3%, 97.3%, 98.0%, 99.4%, 99.5% and 99.0%, so the gap to hard_test shrank from 18.3 to 6.2 to 3.8 to 3.6 to 3.8 to 0.9 percentage points; the gap to the sealed test was **25.1 points** in run 5 (fresh_test) and **3.3 points** in run 6 (fresh_test_v2). Model raw on fresh_test_v2, run 6 scores 91.5% with 7 missed urgent alerts and 58 unsafe rows, 56 of them from acting on corrupted input, which the validator stops in the real pipeline. The untuned base model, given the same prompts, produced **0 correct decisions** on a stratified subset of about 300 rows in every run (run 6: 40 of 296 outputs were valid JSON, none correct).

9.2 What each run taught us

- **Run 1 (v5): the data had volume but not combinations.** In-distribution accuracy was 96.3%, but held-out accuracy 78.0%. Errors clustered on factor mixes never seen together in training (gun-like object plus knife; low battery plus people at the fence): P7 land-to-charge 5% and P8 return-to-launch 2.5% correct, and 39 unsafe rows from acting on corrupted input. -> v6 added cross-factor rows.
- **Run 2 (v6): the remaining errors were jobs a small model should not do.** Missed urgent alerts fell to 0, but the model still failed at spotting broken sensor fields (R0 63% correct; it also invented faults on valid inputs) and at adding up the context score. -> v7 moved both into code.
- **Run 3 (v7): the full pipeline passes; the Brain alone has 15 Gate-caught violations.** 8 were `investigate_closer` with the drone beyond 200 m; 5 were urgent gun-like threats at 10-19% battery where the model chose to land (the rule is to keep tracking during an urgent alert) and dropped "Your decision."; 1 was tracking at critical battery and 1 a false urgent. -> v8 adds rows for exactly these cases and a plain-code `within_200m` input.
- **Run 4 (v8): all 15 fixed; one new unsafe choice.** Urgent-threat rules R1 and R1c reached 100% and investigate-mode R9 97.8%. Against run 3, 44 hard_test rows were fixed and 22 became wrong (57 wrong in total, mostly watch-versus-zoom choices). The single safety violation: GPS lost, battery 11% and a gun-like reading of 58% (watch level) - the model chose `return_to_launch`, although its own alert says GPS is lost; the correct action is to hold and ask for manual control, and the Gate blocks the move. -> Dataset v9 (built 20 October 2026) adds 480 such rows - v8 had 160 GPS-loss rows at low battery, none with a gun-like reading in view - and a sealed fresh test (Section 11).
- **Run 5 (v9): the sealed test shows the real gap.** On hard_test the run 4 violation is fixed (P12 93.3% -> 100%) and accuracy is unchanged (95.7%), but there is now 1 missed urgent alert and 3 unsafe rows, and the full pipeline fails too. Read by hand: the missed urgent had `gun_conf` 1.0 (100%), which the model read as "10%"; the 3 unsafe rows had the right action and risk, but the alert ignored a grenade-like object or a drug packet and said "No weapon seen". On the sealed fresh_test the Brain scores 74.5%: by the report's aggregate per-rule numbers, the power and wind-rotor rules (P1, P2, P3, P5, P10) are only 12-23% correct and `keep_watching` is replaced by `zoom_in` in 157 rows. The fresh_test rows themselves were not opened. GGUF: F16 43/50, Q8_0 85/100 and Q4_K_M 35/50 (2 unsafe) against PyTorch 49/50 and 94/100.
- **Run 6 (v10): a pass for the pipeline, not yet for the Brain alone.** Dataset v10 gave the model the confidence as a whole percentage and the energy rule as a word, added the alert and reason guard, and added 5,000 rows that put every rule in contexts the narrow samplers had missed; the training settings were unchanged. On the new sealed fresh_test_v2 the Brain scores 95.7% (run 5 on its sealed test: 74.5%), with 0 of 213 urgent alerts missed and over-alerting at 0.6%. The power and rotor rules that were 12-23% correct in run 5 are now 95-100%. Two rows are unsafe: the scorer reports that the Safety Gate would block the action for critical battery. In line with the protocol the sealed rows were not opened, only these aggregate counts; the Gate blocks both in the pipeline, which therefore passes. The weakest rules are R14 (weak but persistent reading, 85.0%), R6 and R7 (86.7%); the most common confusions are `keep_watching` -> `zoom_in` (13 rows) and `zoom_in` -> `track_target` (12). On hard_test the Brain scores 98.1% with 1 unsafe row, read by hand: a manual-mode row with GPS lost and a gun-like reading of 48%, where the reason said "No weapon seen"; the reason guard corrects it in the pipeline. GGUF: F16 48/50 (1 unsafe), Q8_0 88/100

(2 unsafe) and Q4_K_M 44/50 (5 unsafe) against PyTorch 50/50 and 100/100.

9.3 The design lesson

From run 1 to run 4 the model, the settings and the held-out inputs stayed the same. What changed was the division of labour: each time the model failed at something deterministic - validating fields, adding a score, checking a distance - we moved it into code and removed it from the model's job, and we added training rows for the factor combinations it had not seen. Accuracy rose by 16.5 points, missed urgent alerts went to 0, and the one safety violation left is the kind a deterministic Gate catches. For safety-critical edge AI we read this as an argument for small models with narrow jobs inside a code scaffold, rather than a larger model asked to do everything.

Run 5 adds the second lesson: a design test is not a generalisation test. Each fix in v6-v8 was chosen by reading hard_test failures, so hard_test kept improving while a sampler the model had never seen exposed a 25-point gap. The same answer as before applies - simple threshold rules (power, rotor, confidence read-out) belong in code - but the claim of progress must be measured on a test no design decision has touched. For run 6, fresh_test v1 is no longer independent (run 5's result informs dataset v10), so a new sealed test, ideally written by people outside the project, is required.

Run 6 adds the third lesson: the same move works on a test that did not shape it. Moving the read-outs the model got wrong into code closed most of the sealed-test gap (74.5% to 95.7%) with the same model and settings. Two cautions remain. The Brain alone still fails the bar, so the pass belongs to the pipeline with its deterministic Gate, not to the model. And fresh_test_v2 was written by the same author with the same rules: it is independent of the training samplers and of every design decision, not of the rules. If dataset v11 is designed from run 6's results, a further sealed test is needed.

9.4 Latency

On the T4 with Hugging Face generation at batch 1, one full decision (about 70 output tokens including the reason and alert) took a median of 4.6-5.2 s (p95 5.6-6.2 s) in runs 1-5 and 5.9 s (p95 7.1 s) in run 6. On a CPU, the GGUF files needed 12-24 s per decision in the checks; in the laptop demonstrator (Intel i5-12450H, 4 threads, both detectors running on the same CPU) the run 6 Q8_0 file needs about 45 s. Both are far from the design target of about 2 decisions per second; the action is the first output field, so the plain-code layer can act before the explanation finishes, and Jetson Orin NX timings with llama.cpp still need to be measured.

9.5 Quantisation for the on-device file

File	Run 3: correct (PyTorch, same rows)	Run 4: correct (PyTorch, same rows)	Run 5: correct (PyTorch, same rows)	Run 6: correct (PyTorch, same rows)
F16 (no quantisation)	19 / 25 (20 / 25)	40 / 50 (48 / 50)	43 / 50 (49 / 50)	48 / 50 (50 / 50), 1 unsafe
Q8_0	41 / 50 (47 / 50)	87 / 100 (95 / 100)	85 / 100 (94 / 100)	88 / 100 (100 / 100), 2 unsafe, 1 invalid
Q4_K_M	35 / 50 (47 / 50)	30 / 50 (48 / 50)	35 / 50 (49 / 50), 2 unsafe	44 / 50 (50 / 50), 5 unsafe
Q4_K_M, prompt passed as text instead of token ids	13 / 25 (20 / 25)	-	-	-

llama.cpp tokenises the prompt exactly as the trainer does, and the GBNF grammar changes no answers. On this task Q4_K_M loses a large share of correct decisions, and in run 4 it also **missed 1 of 6 urgent alerts**; Q8_0 loses fewer. In run 4 even the unquantised F16 file lost 8 of 50 decisions against PyTorch, so part of the gap comes from the export or inference path (LoRA merge, numerics) and not only from quantisation. We therefore do not use any GGUF file yet: the export path will be investigated and F16 / Q8_0 re-validated on the full hard_test before any device test, with the prompt always passed as token ids. Run 6 shows the same pattern: the prompt tokens match the trainer's exactly in all three files, yet F16 loses 2 of 50 decisions and Q8_0 12 of 100. The export check in Section 9.6 finds the cause.

9.6 Export check: the cause of the gap and a file that fixes it

A separate Kaggle notebook (no training) tested five hypotheses on the run 6 adapter: H1, fp16 rounding during the merge; H2, a QLoRA mismatch (the adapter was trained on the 4-bit NF4 base but merged into the original Qwen3-1.7B weights); H3, near-ties at the decision tokens; H4, quantisation loss; and H5, the llama.cpp runtime. **Phase A** (teacher forcing, 200 stratified hard_test rows, PyTorch): each variant reads the prompt plus the trained model's own answer, and we count the rows where its top token equals the trained model's token at every position (so greedy decoding reproduces the answer exactly) and the decision tokens (action, risk, attach_location) that flip. GGUF files are tested here by de-quantising their weights, which separates weight error from the runtime. **Phase B** runs the files in llama.cpp ([llama-server](#), grammar on, prompt as token ids) and scores the decisions with [score.py](#).

Phase A variant (200 hard_test rows)	Rows reproduced exactly	Decision tokens flipped
Trained model (4-bit base + adapter), the reference	100%	0
Merged into the de-quantised 4-bit base, fp32 / fp16 weights	99.5% / 99.5%	0 / 0
Merged into the original full-precision base	28.5%	16
Run 6 Q8_0 file (made from the original-base merge)	26.5%	18
GGUF from the correct merge: F16 / Q8_0 / Q6_K + importance matrix	99.5% / 94.5% / 78.5%	0 / 0 / 3
GGUF from the correct merge: Q4_K_M / Q4_K_M + importance matrix	10.5% / 35.0%	30 / 11

Cause: H2. Merging the adapter into the original weights reproduces the run 6 file's error almost exactly (28.5% against 26.5% of rows, 16 against 18 flipped decisions), and merging into the de-quantised 4-bit base removes it (99.5%). The difference between the original and the 4-bit base weights is a median 6.9 times (up to 14 times) the size of the adapter's update, so the adapter is corrected against weights the export did not use. H1 is ruled out: fp16 rounding is 1.6% of the update (median) and the F16 file is exact; H3 too: 0.31% of the reference's decisions are near-ties (logit margin below 1). Quantisation (H4) matters only at 4 bits.

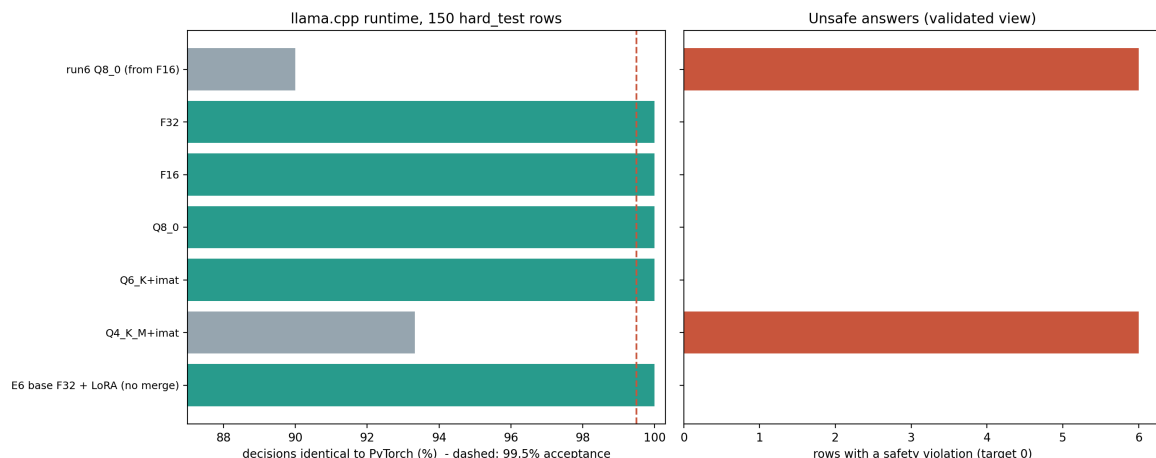


Figure 5 - Export check, llama.cpp runtime on 150 hard_test rows: decisions identical to the trained model (dashed line: 99.5% acceptance) and rows with a safety violation (target 0).

Phase B file (llama.cpp, 150 hard_test rows)	Size	Decisions identical to PyTorch	Decision accuracy	Missed urgent	Unsafe rows	Valid JSON
Run 6 Q8_0 (original-base merge)	1.83 GB	90.0%	88.7%	0 of 23	6	100%
F32 / F16 (correct merge)	6.89 / 3.45 GB	100% / 100%	98.0%	0 of 23	0	100%

Q8_0 (correct merge)	1.83 GB	100%	98.0%	0 of 23	0	100%
Q6_K + importance matrix (correct merge)	1.42 GB	100%	98.0%	0 of 23	0	100%
Q4_K_M + importance matrix (correct merge)	1.11 GB	93.3%	91.3%	0 of 23	6	100%
4-bit base + adapter, not merged (F32)	6.96 GB	100%	98.0%	0 of 23	0	100%

On these rows the trained model in PyTorch also scores 98.0%, so the files that agree on every decision lose nothing. The runtime is not the cause (H5): F32 in llama.cpp agrees on 100% of decisions. **We choose the Q6_K file with an importance matrix (1.42 GB):** the smallest file that meets the hard_test part of the acceptance rule (at least 99.5% of decisions identical, 0 missed urgent, 0 unsafe, 100% valid JSON). Its reasons and alerts are worded differently on some rows (64.7% exact text against 87.3% for Q8_0) while every decision is the same. Q4_K_M is rejected: 6 unsafe rows, each an urgent alert without "Your decision." (one also with an action the Safety Gate would block); the old run 6 Q8_0 file had 6 too (4 said "No weapon seen" next to a gun-like reading of 30% or more, 2 urgent alerts lacked "Your decision.").

Not yet done. The llama.cpp build on Kaggle had no GPU support, so Phase B ran on 150 of the 1,300 valid hard_test rows instead of all of them, and Phase C - the chosen file on the sealed fresh_test_v2, which the acceptance rule also requires (0 missed, at most 2 unsafe) - was skipped. Hard_test helped design the data (Section 11), so the new file is accepted on hard_test only; the sealed-test run, the full 1,300 rows and Jetson timings come before any device test. The laptop demonstrator (Section 10) still uses the run 6 Q8_0 file.

10. Deployment plan

The target computer is a Jetson Orin NX: Models A and B in TensorRT INT8, the Brain with llama.cpp and the GBNF grammar at temperature 0. Latency, heat and power will be measured on the device. Flight testing will proceed in steps: manual flights with detection and alerts only, then follow mode, then investigate mode, then PX4 hardening (secure boot, signed firmware, encrypted MAVLink).

Laptop demonstrator (October 2026). To exercise the interfaces end to end before any hardware, a demonstrator (GARUDA Soldier Watch) connects all three models on one laptop and shows the result in a phone app over local Wi-Fi. Model A runs on every webcam frame. Model B runs on every second frame on a grayscale copy of the webcam image, because there is no thermal camera, and supplies the person, vehicle and thermal-agreement inputs. Plain code tracks 30 frames and builds the 21 facts; the drone fields (battery, GPS, wind, solar) are simulated. The instant alert comes from the rule engine and the Safety Gate; the run 6 Brain (the Q8_0 file with llama.cpp and the grammar) runs beside it in the background at about 45 s per decision on the laptop CPU, and the app shows whether the two agree. The phone runs no model; the soldier acknowledges alerts, switches modes or takes manual control. It is a laptop at ground level, not a drone, it still uses the run 6 Q8_0 file (the corrected file of Section 9.6 has not been put into it yet), and the demonstrator has not been evaluated: we report no results from it.

11. Limitations and responsible use

- **Synthetic labels.** All decision data is generated and labelled by SOP rules. The model learns those rules, not real-world judgement; hard_test measures generalisation to new inputs and wording, not whether the rules are right.
- **hard_test is no longer fully independent.** Datasets v6-v8 were designed after looking at hard_test failures. Dataset v9 (20 October 2026) therefore adds a sealed **fresh_test**: 1,360 rows from a new sampler that draws every field independently over its whole range, generated after all other data and frozen by a hash. From run 5 it is the headline for whether the Brain passes alone. It was written by the same author with the same rules, so it is independent of the training samplers, not of the rules. Run 5 scored 74.5% on it; because that result informs dataset v10, fresh_test v1 cannot be the independent test for run 6. Dataset v10 (built 20 October 2026) therefore adds a new sealed **fresh_test_v2** from another new sampler, frozen by a hash before any training; run 6 is reported on it, and its rows have not been opened. If dataset v11 is

designed from run 6's results, a further sealed test is needed.

- **Proposed rules.** R0, R9, R12, R14-R17, R1c, P10-P12, the weather rule, the v6 risk rule and the Gate's newer checks are proposals that need review by the people who own the SOP.
- **The Brain alone does not pass yet.** In run 6 it fails the bar on the sealed fresh_test_v2 by 2 unsafe rows, which the Safety Gate blocks; the full pipeline passes. The scorer became stricter in v10, so run 6 is compared with run 5 only where both are scored the same way (Section 8), and the sealed test was written by the same author with the same rules.
- **Detectors.** Model A has not flown. Model B has one training run on a mixed-modality public dataset with some machine-made labels; its Gun and Mine scores may reflect the source datasets, and it has not seen real drone footage. A missed detection upstream cannot be fixed by the Brain.
- **Speed and the on-device file.** Measured decision times are seconds (about 45 s on the demonstrator laptop), not the design target (Section 9.4). The corrected Q6_K file matches the trained model's decisions on 150 hard_test rows, but has not yet been run on the sealed fresh_test_v2, on all 1,300 hard_test rows or on a Jetson (Section 9.6).
- **The demonstrator is not a drone.** It runs on a laptop webcam at ground level with simulated drone data; Model B sees grayscale webcam images, not thermal ones; it has not been evaluated.
- **No validation.** Nothing has been flight-tested or validated by any army.
- **Responsible use.** The system is designed for surveillance and alerting only. Its action space must never be extended to weapon release, targeting or engagement, and it should only be operated in a lawful, authorised security context with a soldier in the loop.

12. Conclusion

GARUDA keeps the learned parts small and bounded - two detectors and one decision model - and puts every safety-relevant check in deterministic code. Over four training runs with identical settings, the decision model went from 79% to 96% on the same held-out inputs and the full pipeline met the pre-registered pass bar. A fifth run, scored on a sealed test that no design decision had touched, showed that much of that progress was specific to the design test: 74.5%, 4 missed urgent alerts and 8 unsafe rows, a clear fail. For the sixth run we moved the remaining simple read-outs into code, added an alert and reason guard and much wider training data (dataset v10), and kept the settings: on a new sealed test the Brain scores 95.7% with 0 missed urgent alerts, but still fails the bar alone by 2 unsafe rows, which the Safety Gate blocks; the full pipeline passes. An export check traced the on-device accuracy loss to merging the adapter into the wrong base weights; merged into the base it was trained on, a 1.42 GB file makes the same decisions as the trained model on the hard_test rows checked. A laptop demonstrator now connects all three models to a soldier's phone app, with simulated drone data. The next steps are the sealed-test check of the corrected on-device file, a sealed test written outside the project, per-source results and real footage for Model B, timings on a Jetson, and only then integration on a drone.

Appendix A - Reproducibility

Dataset	train	val	test	hard_test (SHA-256, first 16 hex)
v5	73f42454ebc18671	97e8379f28d9f446	d5fa34b7f79d6627	a2e9866b06b75108
v6	9ccf8bc35584deca	9745416a905a101a	2e5934f143d9276a	ead2bee88644f0ba
v7	03226b3bbcac99d2	ca1dfb3928a8b0d4	36d9d4f7491de99b	e61dff90af3135f8
v8	e836322e66a84dcb	2ad6a69f9eb422e0	841af6e985ed0119	ab86fccf90a09334
v9	ee9521f28b69bf58	e56989961ae9f3c8	68905da5e11df4db	6905571ffc4061f3 (fresh_test be77ffe8e44dfacb)
v10	7f4386fa096203e5	5d4fdd7cde7fee8d	b49962266f8c0910	4c051c813cf743a8 (fresh_test 4b2fcd5abbba95d9; fresh_test_v2 5313c5488db63678)

The `hard_test` files differ only in row IDs and in labels where a rule changed; their inputs are byte-identical (G38). The datasets, the generator (`generate_dataset.py`, which holds the rule engine), the scorer (`score.py`) and the plain-code layer (`pipeline.py`) are not public; they are available on request for research review (jay@seerror.com), and the SHA-256 prefixes above let anyone verify a copy received that way. The training and export-check notebooks and the aggregate results of every run are public with this report.

Appendix B - Example decision

Input (abridged): follow mode, night, border fence, 2 people, bag, gun_conf 0.71 in 12 of 30 frames, thermal not available, battery 64%, GPS ok, airborne.

Output: `{"action": "track_target", "risk": "urgent", "reason": "Gun-like object 71% in 12/30 frames, thermal not available. Track and report only. Hold standoff.", "alert": "URGENT: gun-like object 71% in 12/30 frames, thermal not available. Not confirmed as a weapon. Drone only tracks and reports. Your decision.", "attach_location": true}`

Contact

Jay Tiwari, Seerror Technologies, Bengaluru - jay@seerror.com - www.seerror.com - www.jaytiwari.com