Drone-Megaddon 2026: Co-Intelligent Swarm Control with Language and Vision Models*

Sairam Krishnan
The Wharton School, University of Pennsylvania
Philadelphia, PA, USA
sairamk@wharton.upenn.edu
sairambkrishnan@gmail.com

Daniel Ting
Independent
USA
daniel.ting@live.com

ABSTRACT

In 2014, we introduced Drone-Megaddon, an Android interface for managing drone swarms, patterned after real time strategy (RTS) video games like *StarCraft II*. We argued that its select-then-execute paradigm fit commanding multiple drones better than the single drone piloting interfaces available at the time. Twelve years later, three things around it have changed. Commercial drones now ship with LTE connectivity and with navigation modes built for environments where GPS is denied. Drone warfare at scale has generated a public body of operational lessons on electronic warfare and operator load. And foundation models have made natural language intent and visual grounding tractable as interfaces to robotic systems. The third change is the one this paper turns on. We propose a successor to Drone-Megaddon that keeps the 2014 core: RTS selection, the `$RAD` command queue, ACK'd radio, and human oversight. Over that core we propose three AI layers. The intent translator takes an operator utterance together with the currently selected drones and the map state, and emits a validated sequence of `$RAD` commands. The perception grounder takes a referring expression such as "follow the red sedan" together with the drone's current video feed, and emits a tracked bounding box bound to a specific drone. The coordination planner takes a single waypoint intended for multiple drones and emits one distinct final position per drone, addressing what the 2014 paper called the random offset at destination behavior. We describe each layer at the level of its interface, walk through a search-and-rescue example, and address ethical considerations the 2014 paper did not engage with. This is a proposal; the detailed architecture is in §5.

Keywords

Drone swarms, human and AI interaction, foundation models, real time strategy, multiple agent systems, co-intelligence, cyberphysical systems

1. INTRODUCTION

In December 2014, we published Drone-Megaddon, an Android application for managing multiple small drones through a shared command pipeline [1]. The design was patterned after real time strategy (RTS) video games like Blizzard's *StarCraft II*, which let a single operator command dozens of units through a simple loop: select units, issue a command, watch them execute, repeat. We argued that this select-then-execute paradigm fit drone swarm command better than the single drone piloting interfaces that dominated the consumer market at the time. We built the prototype end-to-end: an Android tablet, a Zigbee radio, a custom ASCII protocol we called `$RAD`, and an AR.Drone in the field.

Twelve years later, the interface choice appears to have held up. Operators of the DJI Matrice 350, the Skydio X10, and the battlefield consoles reported in the Ukrainian and Israeli theaters all issue commands in a recognizably similar sequence: selection first, then intent [24, 25, 31]. What has changed is what the drone can do once it receives the command. In 2014, the drone was effectively a thin client; the cognition lived in the operator and in a small set of hand-written flight primitives. In 2026, the drone can host a vision and language model on device, and the operator's

*This manuscript was drafted with substantial assistance from Anthropic's Claude (Opus and Sonnet) under a multi-agent writing workflow. The authors directed the work, set the architecture and arguments, made all final editorial decisions, and verified all empirical and citational claims; Claude served as a drafting and review collaborator under explicit human direction throughout. This disclosure is consistent with the paper's own thesis that AI cognition embedded in a human workflow should be auditable rather than concealed.

input extends from waypoints and button presses to natural language goals and referring expressions over the video feed.

Foundation models are what enable the shift. Large language models translate ambiguous natural language intent into structured plans or code [16–18]. Models that combine vision and language ground linguistic referents in camera feeds [19, 20]. The kind of coordination across a small number of agents that in 2014 required specialized planning code can now be sketched by a language model over a set of constraints and refined by a classical solver. The resulting operating model is closer to what the human factors literature has called human autonomy teaming [3, 15], or, in the popular framing, co-intelligence [2]: a human sets goals, and AI teammates translate them into behavior.

In this paper, we revisit Drone-Megaddon under that frame. We keep the 2014 core: RTS selection, a command queue, ACK'd radio messages, and a human in authority over every action. Over that core we propose three AI layers: an intent translator (§5.1), a perception grounder built on a vision and language model (§5.2), and a coordination planner (§5.3) that addresses the 2014 system's random offset at destination limitation. Their input and output contracts are summarized in the abstract and specified in §5. We also outline a safety posture (§5.4), walk through a search-and-rescue example end-to-end (§6), sketch a modern implementation stack (§7), and address the ethical questions the 2014 paper did not engage with (§8.2).

This paper is a proposal, not an evaluation; we do not report quantitative results on a deployed 2026 system. The contribution we claim is architectural: a specification of three AI layers that sit over the 2014 command pipeline, with interfaces defined well enough that each layer can be swapped as the underlying models evolve. The concrete interfaces, data flows, and validation paths are in §5.

2. THEN AND NOW

The 2014 paper surveyed a thin, consumer-oriented field. Commercial drone control at the time was dominated by Parrot's Sensefly subsidiary, DJI's Phantom 2 Vision+, and defense-side demonstrations from DreamHammer. Academic work on drone swarms had focused largely on locomotion, formation flying, and signal propagation. Interfaces for commanding multiple drones barely existed outside preplanned mission scripts. Against that backdrop, we argued that an RTS interface was the most tractable way to let one operator command several drones. The situation in 2026 differs along three dimensions: the commercial stack, the AI layer, and the operational evidence from the Ukrainian and Israeli theaters.

2.1. Commercial and Operational Maturity

The commercial drone market has consolidated around a small number of well-capitalized players. DJI, still the dominant global manufacturer, ships the Matrice 350 RTK and exposes multi-drone fleet coordination through DJI FlightHub 2, its cloud console for enterprise fleets [24]. DJI's US market position has eroded over the last several years, with Section 848 of the FY2020 NDAA [29] restricting Department of Defense procurement of Chinese drones and the FCC adding DJI to its Covered List in December 2025 [30], effectively blocking new model imports. The resulting space has been filled in part by Skydio, whose X10 platform publishes autonomy primitives for obstacle avoidance and flight in environments where GPS is denied [25]. In civilian logistics, Zipline reports passing one million commercial deliveries in April 2024, across seven countries including Rwanda, Ghana, Nigeria, Japan, and the United States [26]. In the defense sector, Anduril's Lattice for Mission Autonomy markets coordinated control of diverse robotic assets under a single human operator [27], and Shield AI's Hivemind describes collaborative read-and-react drone-swarm behaviors [28]. The substrate a 2026 operator works over is therefore substantially different from the Parrot and early DJI hardware we targeted in 2014.

2.2. Foundation Models for Robotics

The more substantive change is in the AI layer. Starting around 2022, a line of work showed that large language models could translate natural language goals into executable robot programs. Google's SayCan [16] paired a language model with a learned value function so that a household robot could decompose a request into feasible actions. Code as Policies [17] showed that a language model could write the control code itself, not just the plan. VoxPoser [18] extended the idea to 3D value maps for manipulation. A parallel line produced a class of model that combines vision, language, and action in a single network, ingesting camera feeds and a natural language instruction and emitting motor commands directly; Google DeepMind's RT-2 [19] and the academic OpenVLA [20] are representative. In 2024 and 2025, humanoid robotics startups like Physical Intelligence, Figure, and 1X extended the approach to whole body control.

We borrow three patterns from this line of work. From SayCan we take the pattern of LM as planner paired with a feasibility check. From Code as Policies we take the pattern of emitting structured code rather than free text; our `$RAD` queue plays that role. From vision and language

action models we take referring expression grounding. We do not adopt end-to-end policy control from a single large model. Our command surface remains the 2014 `$RAD` queue, which is auditable, ACK'd, and much smaller than a VLA's action space. The intent and perception layers produce `$RAD` commands; they do not produce motor torques.

Work specific to drones has also appeared. Chen et al.'s TypeFly [21] compiles natural language into executable scripts for a commercial drone, focused on the single drone case. Vemprala et al.'s *ChatGPT for Robotics* [22] published prompt engineering patterns for language driven control of robots and drones; the open source *PromptCraft* repository collects community prompts in that style. NASA's Advanced Air Mobility program has published on operator interfaces for one-to-many vehicle control, including m:N and human-autonomy-teaming concepts [23]. None of these integrates all three layers we propose here, namely natural language command, referring expression grounding in the drone's video feed, and coordination across multiple drones under a single operator's intent. Drone-Megaddon 2014 solved only the coordination interface.

2.3. Lessons from the Ukrainian and Israeli Theaters

Drone warfare at scale since 2022 has generated two operational observations that bear directly on any proposal for AI-augmented drone command. First, autonomy degrades under adversarial electromagnetic conditions: GPS denial and radio jamming routinely push operators back onto terminal camera guidance or manual flight [31, 32]. Any AI layer we add must degrade gracefully when its upstream signals do. Second, at the small unit level, flying and observing multiple drones at once is heavy enough that sustained operations typically run with a dedicated operator per drone [32]. Anything that credibly lets one operator command several drones at once is therefore operationally valuable. That is what our coordination layer targets. We flag this early because it colors the ethical discussion in §8.2.

2.4. Where This Paper Sits

Between the commercial consoles (DJI FlightHub 2, Skydio's autonomy stack) and the defense platforms (whose architectures are mostly classified), there is room for an open academic proposal that takes the 2014 paper's approach of commanding at the level of intent, stacks the three foundation model layers that have arrived since, and works out their interfaces. That is what the rest of this paper does.

3. RECAP: DRONE-MEGADDON 2014

This section is a compact summary of the 2014 system, provided for readers who want the context without retrieving the citation. The architecture it describes is the substrate we extend in the rest of the paper.

3.1. Hardware and Communication

The 2014 prototype ran on a Samsung Galaxy S5 (Android 4.4.2) tethered to a Firefly wireless node over a serial connection on the USB port. Each Parrot AR.Drone in the field carried its own Firefly, a Piksi GPS module, and the Drone-RK firmware stack. The two Firefly nodes exchanged messages over Zigbee. The Piksi sent GPS fixes to the drone's Firefly over UART. In short, the operator's Android device and each drone in the field were two serial endpoints of a Zigbee radio link, with Firefly nodes on each side acting as radio adapters.

3.2. The RTS Interface

The Android app presented a Google Maps view of the operational area. Drones appeared on the map as icons at their reported GPS coordinates, with a small status panel showing altitude and battery level. The operator selected one or more drones using taps (analogous to unit selection in *StarCraft II*) and then issued a command. Tapping a destination on the map sent all selected drones to that waypoint. A command bar at the edge of the screen offered TAKEOFF, LAND, MOVE, SPIN, HOVER, and EMERGENCY for finer control. The paradigm was select-then-execute. The original paper argued, and we still believe, that this is the correct abstraction for managing more than one drone at a time.

3.3. The `$RAD` Protocol

Commands crossed the Zigbee link as short ASCII messages in a family we named `$RAD`. Four message types issued actions to a drone: `$RADFLY` (go to a waypoint), `$RADCTL` (take off, land, or enter emergency state), `$RADMVS` (move or spin in a named direction at a rate for a duration), and `$RADHOV` (hover for a specified time). A complementary `$RADGPS` flowed in the other direction, carrying the drone's current GPS coordinates, altitude, and battery level back to the operator's phone. Each message began with a type code, then a drone identifier, then message fields, and finished with a terminator byte of 255. A small Nano-RK firmware layer on each Firefly handled the radio plumbing. An acknowledgement buffer with rolling sequence numbers

let both endpoints detect dropped packets and recover the channel.

3.4. Modularity and Flow of Control

Inside the Android app, a `CommunicationServer` handled all serial traffic with the local Firefly, while a `MapService` owned everything to do with the UI. The two communicated through a pair of queues, one for transmitted commands and one for received messages. A UI event like a waypoint tap pushed a `FlyCommand` onto the transmit queue. A polling thread pulled it off and wrote it to serial; the Firefly relayed it over Zigbee. The receiving Firefly forwarded to the drone over serial, and a Drone-RK task parsed the message and invoked the appropriate flight library call. GPS updates flowed back along the same pipeline in reverse. The key property of this decomposition, and the reason we can reuse so much of it in this paper, is that the communication layer does not care what the UI is. It only knows how to enqueue and dequeue typed messages.

3.5. Acknowledged Limitations

The 2014 paper itself flagged several shortcomings. First, when multiple drones were ordered to the same waypoint, they would converge on exactly one GPS coordinate and collide. The workaround was a small random offset per-drone at the destination. It worked, but it was obviously a hack, and the original paper called it out as a priority for future work. Second, the system could not detect a crashed drone; it showed the last reported position on the map indefinitely. Third, the `$RAD` protocol did not carry an NMEA checksum, so a corrupted packet could masquerade as a valid one if it happened to parse. Fourth, the UI relied only on taps and buttons, even though the phone had a gyroscope and accelerometer that could have made some controls more direct.

Of these, the random offset collision hack is what most directly motivates the coordination layer in §5.3. The crashed drone detection gap is addressed in the safety posture of §5.4. The protocol checksum and sensor input omissions are addressed in the implementation sketch in §7.

4. FROM COMMAND AND CONTROL TO CO-INTELLIGENCE

This section states three design choices that motivate the architecture in §5: intent remains the right abstraction for drone command. Three separable AI layers are the right decomposition beneath it, and the 2014 `$RAD` queue should stay as the command surface underneath them. The interfaces live in §5.

4.1. Why Intent Remains the Right Abstraction

The 2014 paper argued for select-then-execute on UX grounds: RTS games had already solved the one to many command problem for human operators, and drones were close enough to RTS units for the same idiom to carry. That argument still holds. A second argument has since arrived. Intent is also the level at which foundation models naturally interface: a language model compiles natural language into structured output, and a classical planner compiles structured input into an optimized assignment. The operator supplies the goal. Commanding at a lower level, such as stick inputs or end effector deltas, forces the AI either to paraphrase the operator or to replace the operator entirely. Commanding at a higher level, such as a free prose mission brief, pushes the whole translation chain into one opaque model. Intent sits between those extremes: high enough to hide mechanics, low enough that each party's contribution can be written down, logged, and checked.

4.2. Why Three Layers, and Why These Three

A single foundation model, given a microphone and a video feed, could in principle emit `$RAD` commands directly. We do not propose that, and it is worth saying why. Two near neighbors already fuse more than we do. SayCan [16] fuses language and affordance scoring into a single planner with a learned value function. Vision and language action (VLA) models such as RT-2 [19] and OpenVLA [20] fuse language and vision grounding to emit action tokens end-to-end. Both are appropriate in their settings: SayCan for a single household robot with a fixed skill library, VLAs for manipulation trajectories learned from teleoperation data. Neither fits a multi-drone operator console cleanly. The action space of several drones under one operator's intent is combinatorial; imitation policies trained on single robot trajectories do not carry over. And a fused model entangles failure modes: when the command "follow the red sedan" produces the wrong behavior, an operator looking at one opaque model output cannot tell whether the language model misparsed, the visual grounder mistracked, or the coordination layer misplaced. Three separable layers let each failure surface at its own boundary.

So we decompose along three translations that appear in any operator utterance:

- Language to intent: the operator's words become a structured command specification, validated against the selected drones and the map state.
- Language plus vision to referent: a phrase like "follow the red sedan" becomes a tracked bounding box bound to a specific drone's camera.
- Intent plus constraints to placement: a single waypoint intended for multiple drones becomes one distinct final position per drone.

A language model on text alone cannot do the second. A vision and language action model trained on single robot manipulation cannot do the third. A classical planner cannot accept the first. Each requires a different class of model with a different validation surface. The split is closer in spirit to the classical sense plan act pattern [4] than to end-to-end policy learning, for the same reason sense plan act persists in autonomy stacks: the separability lets each layer fail loudly rather than silently [16, 17].

Three layers are not free. The main cost is latency; intent translation, visual grounding, and coordination planning run sequentially before any `$RAD` command ships. A second cost is cascading error; a wrong grounding feeds a wrong coordination plan. We return to both in §5.4.

4.3. Why the `$RAD` Queue Stays

The third choice is to keep the 2014 `$RAD` queue as the command surface, rather than replace it with the continuous action stream that a VLA model emits [19, 20]. Three properties motivate the choice.

First, auditability. `$RAD` is five message types over a narrow ASCII protocol (§3). Every command is a discrete, named event with a typed argument list, which makes the command stream trivially inspectable: an operator reviewing after the fact can point at the specific message that caused a specific drone behavior. A continuous action stream does not offer this resolution without additional instrumentation.

Second, separability. `$RAD` is a small, well-specified action space, so each AI layer targets it independently. A new coordination planner can be dropped in without retraining the perception grounder. Neither layer needs an internal representation of the drone's dynamics. The radio layer built on Nano-RK already handles dropped packets and sequence recovery (§3); message level retransmission gives the AI stack the same delivery guarantees the 2014 system had.

Third, graceful degradation. Under the electronic warfare conditions cataloged in §2, any of the three AI layers may have to be disabled in the field. When that happens, the operator can still tap waypoints on the map and send raw

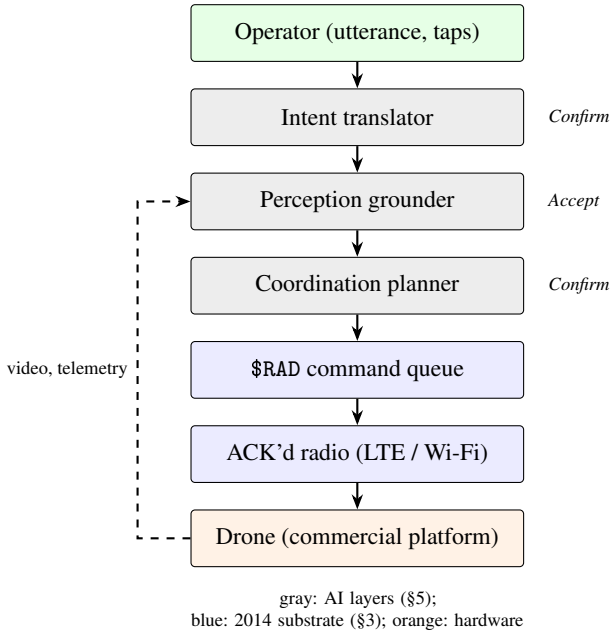

Figure 1: Architecture of Drone-Megaddon 2026. An operator utterance enters the intent translator. Commands carrying a referring expression route through the on-drone perception grounder; multi-drone commands route through the on-phone coordination planner. The output is a sequence of `$RAD` messages on the 2014 command queue, delivered over the ACK'd radio layer. A human inspection point gates every AI-to-`$RAD` boundary.

`$RAD` messages, exactly as in 2014. The legacy path is not a reluctant fallback. It is the default that the AI layers sit on top of.

4.4. What §5 Must Specify

The three choices above leave one open question for the architecture section: what interfaces between the layers make the inspection points real? Each layer must define its inputs, its outputs, the artifact the operator can inspect at its boundary, and the override the operator can issue when the artifact looks wrong. §5 specifies those interfaces. §5.4 takes up the latency, cascading error, and degradation concerns raised above. §6 walks a search-and-rescue scenario through the full stack end-to-end. Throughout, the `$RAD` queue and the ACK'd radio below it are treated as fixed; §3 described them.

5. ARCHITECTURE

This section specifies the three AI layers promised in §4 at the level of their input and output contracts, where each one runs, and the artifact the operator inspects at its boundary. The shared data flow is linear: an operator utterance enters the intent translator. If the resulting command carries a referring expression, it passes through the perception grounder; multi-drone commands pass through the coordination planner. The final output is a sequence of `$RAD` messages on the 2014 command queue (§3). Figure 1 sketches the flow.

We commit to a specific deployment split for inference. The intent translator runs as a hybrid: a small on-phone language model handles quick single-drone commands with bounded latency when connectivity is poor; a cloud-hosted frontier model handles complex multi-step planning when an LTE or Starlink link is available. The on-phone model is a 3B to 8B parameter open-weight model from the Llama, Phi, or Gemma families, 4-bit quantized and served through `llama.cpp` [10]; we leave the specific checkpoint as a deployment choice revisited in §7. Speech input is handled by Whisper-small [9] running on the phone. The perception grounder runs on the drone, using an edge variant of an open-vocabulary detector (Grounding DINO 1.6 Edge [6], derived from [5], or OWL-ViT [7]) paired with a lightweight tracker derived from SAM 2 [8]. The coordination placement step runs on the phone as a classical constrained optimizer, with the language model (wherever it is) only decomposing the task into a geometric spec. Bandwidth is the reason perception runs on the drone itself: streaming a 1080p feed back to the phone over LTE for every grounding call would saturate the uplink long before latency became the binding constraint.

The runtime topology is a pipeline with a shared command queue at its tail. Each AI layer enqueues a typed record that the next layer consumes; the human inspection point at each boundary is a synchronous gate. When a gate is pending, the layer below it idles and the `$RAD` queue is empty. This choice rules out a blackboard or event bus design on purpose: it keeps the responsibility for any command issued unambiguously attributable to the layer that produced it, which matters for both audit (§5.4) and degradation (§4).

Between each layer sits a human inspection point, as promised in §4. The intent translator shows the operator the proposed `$RAD` sequence before transmitting. The perception grounder shows a candidate bounding box on a paused frame before committing to a track, and the coordination planner shows per-drone final positions on the map before issuing `$RADFLY`. None of these are modal dialogs the operator dismisses without reading. They are the contract between each AI layer and the human in authority. §5.4 collects them and specifies what the operator sees and what override is available when the artifact looks wrong.

5.1. Intent Layer: Natural Language to `$RAD`

The intent translator takes an operator utterance and returns a structured command.

The input contract has four fields:

- `utterance`: a UTF-8 string, either typed or produced by on-device ASR (Whisper-small, running on phone).
- `selection`: the set of currently selected drone IDs from the RTS interface (§3).
- `map`: the current operational area polygon, no-fly zones, waypoint annotations, and scale.
- `telemetry`: the most recent `$RADGPS` report from each selected drone, including position, altitude, battery, and flyable state.

The output contract is exactly one of three responses: a validated sequence of `$RAD` messages; a clarification request presented to the operator as a short question; or a refusal, with a reason drawn from a small fixed vocabulary (out of scope, unsafe, selection empty, drone not flyable, no cloud link available for a command that requires planning). Silent failure is not in the contract.

Inference runs in a hybrid configuration, with a routing rule the operator can override. Single drone commands that parse into one or two `$RAD` messages (for example, a hover, a waypoint, a takeoff) route to the on-phone model. Multi-step commands and any command touching more than one drone route to the cloud model when a link is available; when it is not, the on-phone model attempts the command and the system is explicit about the downgrade in its confirmation panel. The justification is operational. The on-phone model bounds worst case latency and keeps the system usable under the electromagnetic conditions cataloged in §2; the cloud model carries the harder planning work when the environment permits. A pure on-phone deployment understates what frontier models add on complex commands; a pure cloud deployment collapses the moment the link drops. Commit to both.

The system prompt is constructed at session start and updated as selection and telemetry change. It contains: the full `$RAD` schema (the five message types from §3, with argument types and units), the current selection and telemetry as a compact JSON block, a scaled text representation of the map (waypoints, no-fly zones, operational area bounds), and a small set of few-shot

examples drawn from the worked scenarios in §6. The user prompt is the operator's utterance verbatim. The model is instructed to return JSON with one of three top-level keys: `commands`, `clarify`, or `refuse`. Output validity is enforced at generation time, not hoped for: the on-phone model uses constrained decoding against the `$RAD` JSON grammar (a context-free grammar mirroring the five message schemas); the cloud model uses the provider's JSON-mode with the same schema supplied as a tool specification. Malformed output is therefore a non-event at the token level, and the post-hoc validation described below catches only semantic errors (out-of-polygon waypoints, dead drone IDs), not syntactic ones. The prompt engineering follows conventions in Vemprala et al. [22] and the structured-output pattern in Code as Policies [17].

Validation runs before any `$RAD` message is emitted: each proposed command passes three checks. The waypoint check: every destination coordinate lies inside the operational area polygon and outside every no-fly zone; every altitude falls in the configured band (default 30 to 120 meters above ground). The selection check: every referenced drone ID is currently selected and in a flyable state per its last `$RADGPS` report. The schema check: arguments parse against the `$RAD` type for their message. A failure on any check promotes the output to a clarification, with the failing field quoted back to the operator; it does not silently drop the command.

Four failure modes recur. First, ambiguous deixis: "go north" when the map has no compass rose shown, or "the other one" when the selection has three drones. The translator routes ambiguous deixis to a clarification. Second, hallucinated commands: syntactic hallucination is ruled out by constrained decoding above, but semantic hallucination (a real `$RAD` type with an invalid argument) remains possible. The validation checks below catch these before the radio ever sees them. Third, out-of-scope utterances: "call my wife" or "what's the score." The refuse path covers these with a short message and the operator's input is not forwarded to the command surface. Fourth, referents that should not be grounded at all: commands implying kinetic action against a named individual or a protected class of subject. Refusal for these lives in the intent layer, where a frontier LLM can reason about the utterance as a whole; downstream layers (perception, coordination) never see a request they should not have received. The refuse vocabulary accordingly includes an "unsafe target" reason alongside out of scope, selection empty, drone not flyable, and no cloud link available.

For human inspection, the phone UI shows the proposed `$RAD` sequence as a short preview panel: message type, target drone, arguments, and a one line natural language gloss ("Drone 2: fly to waypoint W"). The operator confirms the whole sequence with one tap, edits individual commands, or rejects the proposal. Simple commands below a configurable threshold on confidence and command class may auto-confirm after a brief grace period; takeoff, emergency, and multi-drone fly commands always require an explicit tap. §5.4 states the policy.

As a worked example, the operator utters "scout west until the river" with two drones selected. The intent translator resolves "west" against the map's current orientation, resolves "the river" against the map's annotated features, and emits a `$RADFLY` per drone with destinations stepped along the western heading at 200 meter intervals, terminated by a `$RADHOV` at a waypoint near the river polygon. The preview panel lists six `$RADFLY` messages and two `$RADHOV` messages. The operator confirms, and the messages enter the queue described in §3.

5.2. Perception Layer: Grounding Commands in Vision

The perception grounder answers a narrow question: which object in the drone's current video feed does the operator's referring expression name?

The input contract has three fields:

- `referent`: a noun phrase extracted by the intent translator ("the red sedan", "the person in the orange vest", "the smoking vehicle").
- `frame`: the most recent video frame from the target drone, at native resolution.
- `telemetry`: the drone's current position, heading, altitude, and camera pose, for geo-registering any resulting bounding box back to world coordinates.

The output contract is a tracked entity record: {`drone_id, bounding_box(t), class_label, confidence, track_id, world_coordinate`}, updated once per frame while the track holds. On track loss, the grounder emits a terminal record with a loss reason (occluded, left frame, low confidence) and the coordination layer reverts the drone to `$RADHOV`.

Inference runs on the drone. Platforms that publish Jetson-class onboard compute, Skydio X10 [25] as the clearest example, already host enough accelerator capacity to run an edge open-vocabulary detector and a lightweight tracker at useful frame rates; platforms without native AI compute (the DJI Matrice 350 RTK, for instance) require a payload computer such as the Manifold 2 to do the same. The binding constraint is uplink bandwidth, not compute. A

1080p feed at 30fps over an aerial H.265 profile runs roughly 8 to 12 Mbps in high-motion conditions; streaming that to the phone for grounding on every command, over LTE in a contested environment, is not workable. Sending the bounding box and a low-rate preview back is. We reserve a phone-side fallback for ambiguous grounding calls (multiple detections above threshold, or operator rejection of the first candidate), in which case a single high-resolution still frame is uplinked for a second pass with a larger model.

The approach is two-stage. First frame: the on-drone open-vocabulary detector, an edge-deployment variant such as Grounding DINO 1.6 Edge [6] or an edge-compiled OWL-ViT [7] (the vanilla checkpoints do not run at useful drone frame rates), returns candidate bounding boxes for the referent. Subsequent frames: a tracker derived from SAM 2 [8] (the Hiera-Tiny variant is light enough for Jetson Orin NX class hardware) maintains the track without re-running the detector, which keeps per-frame cost bounded. The detector re-runs only on track loss or on operator request. Similar two-stage patterns appear in the VLA literature [19, 20], though there the action head is fused with the grounder; here we keep grounding separate so its output is an auditable bounding box rather than an action token.

For validation, the phone UI presents the candidate bounding box on a paused frame before the tracker commits: the operator taps to accept, taps an alternate detection if the model proposed several, or rejects outright. Only on accept does the grounder enter the tracking state and the coordination layer receive the record. This is the primary human inspection point for the perception layer.

Three failure modes recur. Multiple matches: two red sedans in frame, both above threshold. The UI shows both boxes and the operator disambiguates by tap. No match: the detector returns nothing above threshold. The grounder escalates to the phone-side fallback; if that also fails, it emits a clarification ("no red sedan visible, proceed?"). Track loss: the tracker drops the target mid-flight. The grounder emits a terminal record, the coordination layer sends `$RADHOV`, and the phone surfaces a notification. Safety refusal does not live here: unsafe target cases are caught upstream in the intent layer (§5.1), because the grounder is a detector, not a policy model.

As a worked example, the operator utters "follow the red sedan." The intent translator extracts the referring expression and emits an intermediate plan conditional on grounding. The drone's on-board detector returns one bounding box above threshold, and the phone UI flashes it. The operator taps accept, and the tracker locks. The coordination layer issues a `$RADFLY` sequence that keeps the drone at constant standoff from the tracked bounding box's world coordinate, updated at the tracker's frame rate. Track loss at any later frame reverts the drone to hover and notifies the operator.

5.3. Coordination Layer: Swarm Placement and Task Decomposition

The coordination layer is where the 2014 random offset at destination hack (§3) is replaced with a deliberate placement.

The input contract has three fields:

- `task_spec`: a structured task specification emitted by the intent translator. Three kinds cover most commands: converge on a waypoint, sweep a region, or scatter around a target. Each kind has its own argument schema.
- `drones`: the set of drone IDs available, with per drone capability fields (max speed, payload, camera, flyable state).
- `constraints`: no-fly zones, minimum inter-drone separation (default 8 meters horizontal, 5 meters vertical), altitude bands, and any operator set preferences.

The output contract is a per-drone assignment: {`drone_id` → `(final_position, intermediate_waypoints, approach_speed)`}. The intermediate waypoints are present only when the direct path from current position to final position intersects a no-fly zone or another drone's path.

The task-spec compilation runs wherever the intent translator runs (§5.1); the placement computation runs on the phone as a classical optimizer. The decomposition is deliberate: language to geometry is the kind of translation a language model does well and a classical planner cannot do at all; geometry to placement under constraints is what constrained optimization has done cheaply and deterministically for decades, and we see no reason to push that step through a model.

The placement step uses a Voronoi partition [11] for sweep tasks (each drone claims one cell of a Voronoi partition over the task region, with the generating seeds placed by Lloyd relaxation from initial drone positions); for scatter tasks it uses the same partition with a larger cell radius. Converge tasks use a Hungarian assignment matched against a ring of candidate slots generated around the target waypoint at the minimum separation radius; drones are assigned to minimize total travel. Short-horizon path planning uses A* search [12] over a 2-metre grid induced by the no-fly zones with an Euclidean heuristic; for continuous 3D airspace, sampling-based alternatives (RRT*, lazy PRM [13]) are

appropriate but we do not depend on them for the planar placement step. On a mid-range phone CPU, the placement step returns in under 500ms for a dozen drones in our target sizing; end-to-end latency for a multi-drone command including intent translation is bounded not by placement but by the LLM, and is typically 2 to 4 seconds for on-phone and 3 to 6 seconds for cloud-hosted routing.

Validation and edit loop: before any `$RADFLY` is issued, the phone UI shows the proposed per-drone final positions on the map, colored by drone ID, with a dashed line from each drone's current position to its assigned slot. The operator has three moves. First, the operator can confirm the plan, which issues `$RADFLY` for every drone and exits the layer. Second, the operator can drag a specific drone's assigned slot to a new position; the coordination planner pins that drone to the dragged position and re-solves the remaining assignments against the same constraint set (the re-solve runs the same Voronoi or Hungarian step with one additional fixed constraint). Third, the operator can reject the plan outright, which discards the assignment, removes the task spec from the pipeline, and returns control to the intent layer for a new utterance. The planner never issues commands against an unconfirmed assignment, and the pinned-drone re-solve is itself re-presented for confirmation.

Three failure modes recur. Infeasible placement: too many drones for the region at the given minimum separation, or no feasible path around a no-fly zone. The planner returns a structured refusal naming the infeasibility; the UI offers either relaxing the separation or reducing the drone count. Solver timeout: we bound compute at one second and on timeout degrade to the 2014 behavior, a single waypoint with random per-drone offsets, plus a warning banner. One drone in a non-flyable state: its slot is dropped and the remaining assignments re-run. The task completes with the available complement rather than hanging on the missing drone.

As a worked example, the operator draws a polygon over the north slope and utters "sweep it" with four drones selected. The intent translator emits a task spec of kind `sweep` with the polygon, a default scanning altitude, and a default inter-drone separation. The Voronoi partition yields four contiguous cells; the coordination planner places one drone at the centroid of each cell, with intermediate waypoints routing each drone around a small no-fly zone at the slope's eastern edge. The phone UI shows four colored slots on the map; the operator confirms. Four `$RADFLY` sequences enter the command queue.

5.4. Safety and Human Oversight

The three preceding subsections each named a human inspection point at the AI-to-`$RAD` boundary. This subsection collects them and adds the cross-cutting posture promised in §4: what stays fixed across layers, what the operator sees when a layer degrades, and how the 2014 path remains available when the AI stack is unavailable.

The standing policy on human authority is that no AI layer emits a `$RAD` command above a configurable command-class threshold without explicit operator confirmation. The default threshold places takeoff, landing, emergency, and any multi-drone fly message above the line; single-drone hover and short-hop moves below it may auto-confirm after a one-second grace period during which the operator can cancel. Commands referencing a tracked person, regardless of class, always require explicit confirmation. The threshold is per operator and persisted across sessions. The posture is consistent with human autonomy teaming as framed by Shneiderman [3] and with Sheridan's levels of automation [15]: we sit at the authority-retained end of Sheridan's spectrum.

Three inspection points exist, one per layer.

- Intent: the proposed `$RAD` sequence preview panel (§5.1).
- Perception: the candidate bounding box on a paused frame (§5.2).
- Coordination: the per-drone final positions on the map (§5.3).

Each is a visible artifact, not a hidden internal state. The operator's confirmation on each is the signal that authorizes the next step.

The operator can override at any moment: select a drone and issue a direct `$RAD` command through the 2014 UI. The AI layers silently defer: the intent translator treats a direct waypoint tap as higher priority than its current parse. The perception grounder releases any active track on that drone, and the coordination planner excludes the drone from any pending assignment. The override is not a separate mode. It is the default that the AI layers sit on top of, consistent with the legacy path argument in §4.

Failure attribution matters because separable layers are only worth the overhead if the operator can tell which one misfired. A status ribbon at the top of the phone UI names the active layer producing the most recent proposal, colored green when the operator has confirmed and red when an inspection point is pending. When something goes wrong mid-mission (a drone deviates, a track is lost, a placement is edited), a single tap on the ribbon jumps to the

inspection artifact for the responsible layer and highlights the decision: the intent layer's `$RAD` preview, the perception layer's accepted bounding box, or the coordination layer's placement snapshot. The audit log (described below) retains these artifacts so the jump works for decisions already past. The promise of separable layers is that "the grounder mistracked" and "the intent translator misparsed" are distinguishable claims, and the ribbon plus audit jump is how that distinguishability reaches the operator.

On cascading error: §4 flagged the cost of three layers: a wrong output early in the pipeline compounds downstream. The system mitigates this in two ways. First, the inspection gates described above: a wrong grounder output cannot reach the coordination layer without operator confirmation, and a wrong placement cannot reach `$RAD` without the same. The pipeline is synchronous on purpose. Second, in-flight rollback: when the operator taps "something is wrong" on the ribbon, every drone under AI-generated command reverts to `$RADHOV`, the pipeline's in-flight state is snapshotted to the audit log, and control returns to the intent layer for a new utterance. Rollback is coarse by design; a finer cross-layer rollback protocol (undoing only the grounder's contribution while preserving the intent parse, for instance) is tempting but would require cross-layer state that breaks the separability claim, so we do not provide it.

For crashed-drone detection, the 2014 system showed the last reported position indefinitely (§3). The 2026 system marks a drone with a red X on the map when its `$RADGPS` has been silent for longer than the radio layer's retry window; the operator is notified with a haptic pulse and a banner. The detection lives in the `CommunicationServer`, one level below the AI layers, so it works identically under the legacy path.

For audit logging, every `$RAD` command issued, and every AI layer decision that produced or shaped it, is appended to an on-phone log with the originating utterance, the intent translator's full prompt and response, the perception grounder's accepted bounding box (if any), and the coordination planner's assignment table. The log is local first, replicated on link availability, and the default retention is 30 days. The log is the artifact an after action review or an incident investigation reads.

Under link loss, solver timeout, model refusal, or operator preference, the system falls back to the 2014 path: direct waypoint taps on the map compile to single `$RADFLY` messages; the command bar offers TAKEOFF, LAND, MOVE, SPIN, HOVER, and EMERGENCY, exactly as in §3. The degradation is visible: a small banner names which layer dropped and why. Recovery is automatic on signal return; the operator does not need to toggle a mode.

The safety posture here is engineering, not ethics. The ethical question of where AI-augmented swarm command should and should not be deployed, and what the dual use risks are, is taken up in §8.2. The inspection points and override paths specified above are necessary for any deployment; they are not sufficient justification for every deployment.

6. WORKED EXAMPLE: THE LOST HIKER

This section traces one operator utterance through the three AI layers specified in §5. The scenario is a search and rescue call: a hiker reported missing on a mountain slope, last seen in a distinctive orange jacket. The operator has four drones at a command post near the trailhead (Skydio X10 class, Jetson-class on-drone compute) and one LTE link to the cloud model.

6.1. Setup

The operator opens Drone-Megaddon 2026 on the command post's phone. The map centers on a slope bounded to the north by a ridgeline, to the south by a creek, and to the east by private property containing a small airstrip. Before the utterance, the operator does three things in the 2014 idiom (§3). First, the operator taps ridgeline, creek, and property waypoints to draw the operational area polygon; the phone renders it in faint blue. Second, the operator drags the airstrip no-fly zone from a regional layer; it renders in red. Third, the operator taps Drone 1 through Drone 4 in the drone tray; the status panel confirms all four are airborne at 80 meters AGL over staging, with batteries between 84 and 91 percent, flyable according to their last `$RADGPS` reports. Selection, map, and telemetry are in place before the utterance.

6.2. Utterance and Intent Translation

At T+0, the operator holds the talk button and says: "sweep the north slope for the hiker in the orange jacket and report back if anyone finds him." Whisper-small transcribes in under a second. The intent translator receives the utterance together with the four selected drone IDs, the operational area polygon, the no-fly zone, and the telemetry block. The command touches four drones and composes a sweep with a grounded referent, so the router sends it to the cloud model over LTE. If the link had dropped, the on-phone model would have handled it, with the downgrade flagged on the confirmation panel.

The model returns a `commands` object with two records. The first is a task spec of kind `sweep`: the operational area polygon restricted to the north slope by the ridgeline,

scan altitude 60 meters AGL, per-drone ground speed 6 m/s, minimum separation 8 meters horizontal and 5 meters vertical (the defaults from §5.3). The second is a grounding hint passed to each drone's grounder: "person in an orange jacket." On a positive match, the drone is to hover and report rather than actively pursue. At T+2.8 seconds, the phone renders the preview panel: four rows, one per drone, with a one line gloss "Drones 1–4: sweep north slope at 60 m AGL, watch for orange jacket, hover and report on match." The operator taps Confirm.

6.3. Coordination Planning

On Confirm, the coordination planner runs on the phone. The Voronoi partition over the north slope yields four contiguous cells; Lloyd relaxation from current drone positions seeds them so each drone's assigned cell is the one nearest its staging hover. Drone 3's direct path to its cell centroid would clip the airstrip's buffer, so the A* step inserts two intermediate waypoints that route it north around the zone before descending south into the cell. The other three drones get direct paths, and the placement returns in under 500 milliseconds.

The map preview shows four colored cells over the slope, a slot at each centroid, and dashed lines from each drone's current position to its slot; Drone 3's dashed line bends visibly around the red polygon. A small legend reads "4 of 4 drones placed, 8 m separation, no-fly zone respected." The operator decides Drone 2's slot sits too close to a stand of trees that would occlude its downward camera, and drags the slot 40 meters west. The planner pins Drone 2 there and re-solves the remaining three assignments against the same constraints; the status ribbon stays green and attributes the edit to the coordination layer, not to the intent parse. The revised plan appears well inside the 500-millisecond budget from §5.3, and the operator taps Confirm. Four `$RADFLY` sequences enter the command queue.

6.4. Execution and Perception

At T+9 seconds, all four drones depart, descending from 80 meters AGL to the 60-meter scan altitude on approach to their cells; the on-drone perception grounder loads the referent "person in an orange jacket" into the edge variant of the open-vocabulary detector (§5.2), with the SAM 2 tracker held in reserve for a positive match.

At T+1 minute 47 seconds, Drone 2's detector returns a single box above threshold over a patch of talus: a person in an orange garment, prone, partially shaded by a boulder. The drone issues a `$RADHOV` on itself and uplinks the candidate bounding box with the still frame it captured and its geo-registered world coordinate. The status ribbon turns red at the perception layer; the operator taps it and lands on the candidate. The phone surfaces the paused frame with the box drawn in yellow, the class label `person`, the prompt match `orange jacket`, a confidence score, and Accept and Reject buttons. The other three drones continue their scans. This is the perception layer's inspection point, and the coordination layer does not receive the record until the operator acts.

The operator enlarges the frame, confirms it is a jacket and not a tarp, and taps Accept. At T+1 minute 58 seconds, the grounder transitions to tracking on Drone 2, and the planner replaces Drone 2's coordination slot with a fixed-standoff hover 25 meters above the tracked world coordinate.

6.5. Outcome and a Degradation Beat

On accept, the operator selects Drones 1, 3, and 4 and utters "return to staging and hover." The intent translator emits three `$RADFLY` commands back to the staging waypoint at 80 meters AGL followed by `$RADHOV`; the preview panel shows the sequence; the operator confirms. Meanwhile, Drone 2 holds its track. The phone surfaces a situational summary for the ground team: last known position (the world coordinate from the geo-registered bounding box), heading (stationary), altitude above sea level, and a short track history showing no movement in the two minutes since detection. The operator copies the position to the ground team's radio channel. At T+3 minutes 12 seconds, the ground team is dispatched, with Drone 2 remaining overhead.

At T+4 minutes 30 seconds, Drone 3 drops its `$RADGPS` heartbeat on its return leg. The `CommunicationServer` waits out the radio layer's retry window (§5.4), fails to recover the channel, and marks Drone 3 with a red X at its last reported position; the operator receives a haptic pulse and a banner. Because crashed-drone detection lives below the AI stack, it runs the same way whether any AI layer is active. The operator notes the position for a recovery team and returns focus to Drone 2's track; the AI layers do not attempt a retasking, since the live track is the priority and a retasking is an operator decision.

In this trace, the full loop from utterance to handoff ran under four minutes. The intent translator produced one sweep task and one recall sequence. The perception grounder fired once, returned one candidate, and was accepted on the first proposal. The coordination planner ran twice, the second time with a pinned drone. The `$RAD` queue carried eleven messages across the four drones. The audit log retains all of it, keyed to the originating utterance, and is the artifact an after-action review would read (§5.4). The walkthrough

is the sympathetic framing of the underlying capability: the same primitive that locates a consenting lost hiker can locate a non-consenting subject in the same video feed. The ethical considerations this raises are taken up in §8.2.

7. IMPLEMENTATION SKETCH

This section names the stack a 2026 build would use, drawing the line between what carries over from 2014 and what is new. §5 committed to model families; this section commits to specific reference components one level down.

7.1. Hardware

On the operator side, a modern Android phone is the reference device: a Pixel 10 or Galaxy S26 class handset, running an Android app similar in shape to the 2014 application. The Zigbee radio and its tethered Firefly node are gone. The link from the phone to the drones runs over 5G, with LTE as the common fallback and Starlink in operational settings that warrant it, out to a cloud hub that forwards commands to each drone; a Wi-Fi mesh between phone and drones is the local fallback when the cellular link drops.

On the drone side, the reference platforms are commercial systems that publish Jetson-class onboard compute. Skydio X10 is the clearest case: NVIDIA Jetson Orin-class compute onboard, with Skydio's autonomy stack above it. For fleets already on DJI, the Matrice 350 RTK with a Manifold 3 payload computer (also Jetson Orin NX) reaches the same compute envelope. Procurement-restricted deployments (US federal, defense, public-safety) take the Skydio path [29, 30]; commercial deployments outside those restrictions can take either. The flight stack is whatever the platform ships: Skydio's proprietary autonomy stack on the X10, DJI's autopilot on the Matrice; PX4 is the option for open airframes. The Drone-RK firmware layer of the 2014 system is retired in every case. The onboard OS is whatever the platform ships, typically Linux; no custom kernel is assumed. Each drone carries a 4K HDR camera with a hardware H.265 encoder, which is what makes the bandwidth argument in §5.2 close.

7.2. Software: What Carries Over

The 2014 architecture's separation of concerns is preserved. The Android app keeps the `CommunicationServer` and `MapService` split described in §3, now with three additional services colocated with them: an intent service wrapping the on-phone language model, a coordination service wrapping the placement optimizer, and a perception proxy that relays grounding traffic to and from the drones. The `$RAD` protocol carries over verbatim. The same five message types (`$RADFLY`, `$RADCTL`, `$RADMVS`, `$RADHOV`, `$RADGPS`) with the same argument shapes and the same 255-byte terminator are what enter and leave the command queue. ACK'd delivery semantics carry over too, now riding LTE and Wi-Fi rather than Zigbee, with sequence numbers and retransmission handled one layer below the AI stack. The select-then-execute UI is preserved: the operator taps drones, issues a command, and watches the result, as in 2014.

7.3. Software: What Changes

The new components live above the `$RAD` queue and below the UI. The following is the reference set; each component has a family rather than a single vendor lock-in.

- On-phone language model. The reference is a 4-bit quantized Phi-3-mini (3.8B parameters, roughly 2 GB resident), served through `llama.cpp` [10]. An 8B-class open-weight checkpoint (Llama 3.x, Gemma, or successor) is the step up when the handset allows it, at roughly twice the memory. This is the model that handles quick single-drone commands under the on-phone routing rule in §5.1.
- Cloud language model. A Claude-class or GPT-class frontier model, accessed over HTTPS on the same LTE, 5G, or Starlink link as the radio traffic. This is the model that handles multi-step planning and multi-drone commands when the link permits, consistent with the routing rule in §5.1.
- On-phone speech recognition. Whisper-small [9] compiled for TFLite or QNN, running on the phone's NPU where available and the CPU otherwise. Transcription happens before the utterance reaches either language model.
- On-drone perception. Grounding DINO 1.6 Edge [6] is the reference detector, with an edge-compiled OWL-ViT [7] as the drop-in alternative where an NVIDIA-specific toolchain is not available. A SAM 2 Hiera-Tiny tracker [8] runs alongside it for the frames between detections. Both models are quantized for the Jetson Orin NX or the platform's equivalent accelerator. The output is the tracked entity record specified in §5.2.
- On-phone coordination optimizer. Pure Kotlin, no machine learning. The Voronoi partition, Lloyd relaxation, and Hungarian assignment of §5.3 fit inside a few hundred lines of classical code and run well within the half-second budget on a mid-range phone CPU.

The line between carryover and change is therefore clean. The command surface, the radio semantics, and the UI idiom are the 2014 system. The intent translator, the perception grounder, and the coordination planner are new; each one has a named reference component and a named alternative. A team building this could wire the 2014 application to a modern Android target first, verify the `$RAD` queue still works over LTE, and add the AI layers in the order they were specified in §5. The ethical questions attached to that build are taken up in §8.2.

8. DISCUSSION

8.1. What Changed, What Did Not

Three pieces of the 2014 system carried over with essentially no change. The select-then-execute idiom is unchanged: the operator still taps drones on a map and then issues a command, exactly as in §3. The `$RAD` protocol carried over verbatim, the same five message types with the same arguments and the same 255-byte terminator (§7). The `CommunicationServer` and `MapService` modular split carried over too; the new AI services sit alongside them rather than inside them.

What did not carry was the assumption that the operator supplies all of the cognition. In 2014, each drone was a thin client: the Drone-RK task parsed a `$RAD` message and invoked a flight library call, and every interesting decision happened in the operator's head or in hand-written code on the phone. In 2026, the drone hosts a vision and language model on device (§5.2), and the phone hosts a language model alongside a cloud model the phone can defer to (§5.1). The operator's role moves from input-producer to goal-specifier, in the co-intelligence sense of §1: the operator sets an intent, and the AI layers translate it into behavior subject to the operator's inspection gates.

The interface choice, in retrospect, was the right level of abstraction to commit to. Select-then-execute survived twelve years of hardware and model turnover because the layers that arrived since 2014 slot in cleanly above it. The foundation models became the cognition that sits between the operator's selection and the `$RAD` command queue, rather than replacing either endpoint.

8.2. Ethical Posture

The worked example in §6 ended on a deliberate note: the same primitive that locates a consenting lost hiker locates a non-consenting person in the same video feed. The architecture proposed here is dual use by construction, and the military deployment path flagged in §2 is real; Anduril's Lattice, Shield AI's Hivemind, and the Ukrainian theater reports from RUSI and CSIS describe systems that share primitives with what this paper specifies. The scope of this subsection is the architectural posture this paper can defend directly: the protocol layer between operator and drone, and the guard layer between operator and protocol. Regulatory regimes, liability allocation, and post-deployment model-update governance are out of scope here and belong in companion work. Within that scope, two commitments fall out of the architecture, named below.

The first commitment is an open standard for the utterance-to-`$RAD` path. The transformation from a natural language operator utterance to a `$RAD` command sequence must be specifiable as an open standard. The reasons are concrete. An open specification is auditable by anyone, not only by the system's vendor. Implementations of the specification produce reproducible behavior across vendors, so a claim about how a fielded system behaves can be tested rather than taken on faith. A bad actor's deployment can be inspected against the specification, and independent compliance reviews become possible at all.

The companion to the specification is transport: the prompt and command-sequence trace produced by the intent translator, and the operator-accepted bounding box produced by the perception grounder, should be transmittable to a cloud-hosted log service in addition to the on-phone audit log specified in §5.4. The on-phone log is the artifact an after-action review reads; the cloud-hosted log is what enables monitoring and incident review after a handset has been lost, wiped, or tampered with. The cloud path raises its own surveillance-versus-accountability tension, since the same log that supports incident review also concentrates a persistent record of operator behavior in whatever jurisdiction the log service sits in. We flag this as an unresolved design choice rather than a clean win.

The second commitment is a proactive vetting agent. An open specification alone is not sufficient, because reading the specification does not stop a hostile command at the moment the operator issues it. The structural mitigation is a vetting agent that runs on phone, between the intent translator and the `$RAD` queue. The agent inspects the proposed command sequence together with the originating utterance and blocks or escalates commands that match a configurable do-not-execute policy: commands that imply tracking a named individual without consent, commands that direct kinetic action against persons, commands that are inconsistent with the operator's role or the operator's current location and authorization. The intended shape is a small classifier model, cheap enough to run on the handset alongside the intent translator, with a policy file the

deploying organization can version and audit. The vetting agent is a separate paper's worth of work: the scope of the do-not-execute policy, the training data for the classifier, the false-positive rate an operator will tolerate in the field, and the governance of policy updates are all open questions in their own right.

A few further points on posture follow.

- Dual use is not a side effect. The primitive that locates a lost hiker (§6) is the primitive that locates a non-consenting subject. The open standard and the vetting agent constrain use; they do not eliminate dual use.
- Contemporary AI-recommender failure modes in military deployments are a cautionary precedent. Reported uses of automated target recommendation in the Gaza theater ("Lavender" [33] and "Habsora" [34]) are the shape of failure this architecture should not drift toward. The vetting agent is the structural guard against the intent translator or the perception grounder being re-skinned as a target recommender; it is not a guarantee.
- Operator accountability is a question this paper does not resolve. The separable-layers design of §5.4 lets the operator, and an after-action review, see which layer misfired through the failure-attribution ribbon. Whether legal and organizational accountability follow that attribution is a policy question outside the architectural scope.

The mitigations named here constrain the worst outcomes without resolving the underlying tension. The architecture is easier to audit than a closed stack, and the vetting agent is a place to put the constraint, but neither removes the dual-use reality that the same primitives serve rescue and surveillance.

8.3. Open Questions

Five items are unresolved and are called out rather than glossed.

(1) End-to-end latency under realistic field conditions has not been benchmarked. The budgets cited in §5.1, §5.2, and §5.3 are per-layer targets on reference hardware; the full loop under an LTE or 5G link, with a real cloud model round-trip and a real on-drone detector, is pending.
(2) Adversarial robustness of the perception grounder has not been characterized. Deliberate target denial (camouflage, occlusion, adversarial patches) and spoofing of referring expressions are both credible and unexamined here.
(3) Multi-operator coordination is undefined. Two operators commanding overlapping swarms, or one operator's swarm entering another's operational area, is a scenario the single-operator architecture of §5 does not address.
(4) The vetting agent itself is underspecified. Scope of the do-not-execute policy, source and labeling of training data, tolerable false-positive rate, and governance of policy updates are all open, and are the subject of a companion paper rather than this one.
(5) Open-standard governance is unresolved. Who maintains the specification, how version changes propagate to deployed systems, and how nation-state-specific compliance overlays (export controls, operational-area restrictions, reporting-duty regimes) compose with a common base spec are all questions that a standards body rather than a single paper would work out.

These five are bounds on what the paper argues, not arguments against building the architecture. Each one names a layer of work that has to land before the system would be defensible at scale, and each one is the subject of follow-on work rather than this paper.

9. CONCLUSION

The 2014 Drone-Megaddon paper argued that an RTS interface fit single-operator control of multiple drones. Twelve years of hardware and model turnover later, the claim still holds, for a reason the original paper could not have named: the select-then-execute idiom and the `$RAD` protocol are the substrate that new cognition slots above, not the cognition itself. Foundation models arrived in the layer between operator intent and command queue; they did not displace the selection gesture, and they did not displace the five message types that carry a command to a drone. The 2026 architecture runs co-intelligent layers, in the Mollick sense of §1, on top of the 2014 interface.

The contribution of this paper is architectural rather than empirical: a specification of where each AI layer runs and what it produces, with interfaces tight enough that each layer can be replaced as the underlying models evolve. Intent translation lives on the phone (Phi-3-mini on device, a Claude-class or GPT-class frontier model in the cloud for hard cases); perception grounding lives on the drone (Grounding DINO 1.6 Edge, OWL-ViT, SAM 2 Hiera-Tiny on Jetson-class hardware); coordination planning lives on the phone (Voronoi partition, Lloyd relaxation, Hungarian assignment). The operator stays in command

through inspection gates, an after-action audit log, and a failure-attribution ribbon that names which layer misfired. Two ethical commitments follow from the architecture rather than being bolted on: an open standard for the utterance-to-`$RAD` path, with optional cloud log transport, and a proactive vetting agent between intent translator and `$RAD` queue. Neither commitment resolves the dual-use reality; both make the system more auditable than a closed stack would be.

This paper has specified, not evaluated, that placement; the empirical program lives in §8.3. Of the five open questions there, three are hooks for companion papers: a full specification of the vetting agent (policy scope, training data, false-positive rate, governance of updates), a multi-operator extension of the coordination planner, and a governance proposal for the open standard itself. The other two, end-to-end latency benchmarking and adversarial robustness of the perception grounder, are empirical programs rather than design questions. Co-intelligence is the working frame because the alternatives strain against inspection: full automation gives up the inspection gate that keeps a fielded swarm auditable, and full manual control gives back the 2014 cognitive load that the RTS interface was meant to escape. The phone, the drone, and the cloud each carry a share of the thinking, under a protocol the operator can still read.

References

[1] Sairam Krishnan and Daniel Ting. Drone-Megaddon: A Real-Time Strategy Interface for Managing Drone Swarms. Unpublished manuscript, Carnegie Mellon University 18-848 course project, 2014.

[2] Ethan Mollick. *Co-Intelligence: Living and Working with AI*. Portfolio, 2024.

[3] Ben Shneiderman. *Human-Centered AI*. Oxford University Press, 2022.

[4] Erann Gat. On Three-Layer Architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press / MIT Press, 1998.

[5] Shilong Liu et al. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. In *European Conference on Computer Vision (ECCV)*, 2024. arXiv:2303.05499.

[6] Tianhe Ren et al. Grounding DINO 1.5: Advance the "Edge" of Open-Set Object Detection. arXiv:2405.10300, 2024.

[7] Matthias Minderer et al. Simple Open-Vocabulary Object Detection with Vision Transformers. In *European Conference on Computer Vision (ECCV)*, 2022. arXiv:2205.06230.

[8] Nikhila Ravi et al. SAM 2: Segment Anything in Images and Videos. arXiv:2408.00714, 2024.

[9] Alec Radford et al. Robust Speech Recognition via Large-Scale Weak Supervision. In *International Conference on Machine Learning (ICML)*, 2023. arXiv:2212.04356.

[10] Georgi Gerganov and contributors. `llama.cpp`: Inference of LLaMA model in pure C/C++. Software repository, https://github.com/ggml-org/llama.cpp.

[11] Jorge Cortés, Sonia Martínez, Timur Karatas, and Francesco Bullo. Coverage Control for Mobile Sensing Networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[13] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[14] United States Federal Aviation Administration. Part 107 – Small Unmanned Aircraft Systems, 14 CFR §107.51. Federal Aviation Regulations, as amended through 2024.

[15] Thomas B. Sheridan. *Humans and Automation: System Design and Research Issues*. Wiley-Interscience, 2002.

[16] Michael Ahn et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, PMLR vol. 205, 2022.

[17] Jacky Liang et al. Code as Policies: Language Model Programs for Embodied Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[18] Wenlong Huang et al. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. In *Proceedings of the 7th Conference on Robot Learning (CoRL)*, PMLR vol. 229, 2023.

[19] Anthony Brohan et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of the 7th Conference on Robot Learning (CoRL)*, PMLR vol. 229, 2023. arXiv:2307.15818.

[20] Moo Jin Kim et al. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of the 8th Conference on Robot Learning (CoRL)*, PMLR vol. 270, 2024. arXiv:2406.09246.

[21] Guojun Chen, Xiaojing Yu, Neiwen Ling, and Lin Zhong. TypeFly: Flying Drones with Large Language Model. arXiv:2312.14950, 2023.

[22] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. ChatGPT for Robotics: Design Principles and Model Abilities. Microsoft Research Technical Report MSR-TR-2023-8, February 2023.

[23] R. Jay Shively et al. Multi-Vehicle (m:N) Operations in the National Airspace System: NASA's Research Plans. In *AIAA AVIATION 2022 Forum*. AIAA 2022-3758.

[24] DJI. DJI FlightHub 2: Cloud-Based Enterprise Fleet Management Platform. Product documentation, https://www.dji.com/flighthub-2. Accessed April 2026.

[25] Skydio. Skydio X10: Autonomy Primitives and GPS-Denied Flight. Product documentation, https://www.skydio.com/skydio-x10. Accessed April 2026.

[26] Zipline. Zipline Reaches One Million Commercial Deliveries. Company press release, 19 April 2024.

[27] Anduril Industries. Lattice for Mission Autonomy. Product page, https://www.anduril.com/lattice/mission-autonomy. Accessed April 2026.

[28] Shield AI. Hivemind: Autonomy Software for Collaborative Drone Swarms. Product page, https://shield.ai/hivemind-solutions/. Accessed April 2026.

[29] United States Congress. National Defense Authorization Act for Fiscal Year 2020, Section 848: Prohibition on Operation or Procurement of Foreign-Made Unmanned Aircraft Systems. Public Law 116–92, 2019.

[30] United States Federal Communications Commission. Addition of DJI to the Covered List. Order, 23 December 2025.

[31] Jack Watling and Nick Reynolds. Meatgrinder: Russian Tactics in the Second Year of Its Invasion of Ukraine. Royal United Services Institute (RUSI) Special Report, May 2023.

[32] Gregory C. Allen, Kateryna Bondar, and Samuel Bendett. The Russia-Ukraine Drone War: Innovation on the Frontlines and Beyond. Center for Strategic and International Studies (CSIS), 2025.

[33] Yuval Abraham. "Lavender": The AI Machine Directing Israel's Bombing Spree in Gaza. *+972 Magazine and Local Call*, 3 April 2024.

[34] Yuval Abraham. "A Mass Assassination Factory": Inside Israel's Calculated Bombing of Gaza. *+972 Magazine and Local Call*, 30 November 2023.