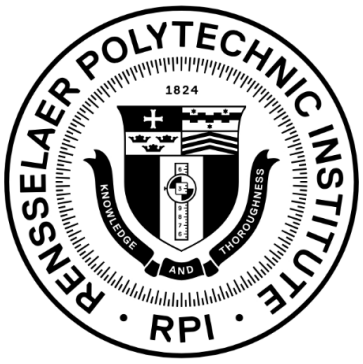
PREDICTIVE SECTORIZATION AND BAYESIAN OPTIMIZED CONSENSUS FOR ADMISSION CONTROL IN AUTONOMOUS AIRSPACE OPERATIONS

Aditya Dhodapkar

Submitted in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Approved by:
Dr. Carlos Varela, Chair
Dr. Stacy Patterson
Dr. Radoslav Ivanov
Dr. Markus Endler



Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, USA

[May 2026]

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Firstly, I would like to thank my advisor, Professor Carlos Varela, for his guidance and support throughout this research. His overall vision and constant feedback shaped every stage of this work, and I am grateful for the opportunity to learn from him. He constantly pushed me to explore my own ideas and never held back when pointing out flaws, making me a more complete researcher.

I would also like to thank my lab mates in the Worldwide Computing Lab: Avery Smidt, Aaron Verkleeren, and Dylan Le. Their input during discussions and their willingness to help debug and test made this project better than it would have been on my own.

I am grateful to the FAA System Wide Information Management (SWIM) program for granting me access to use real time SWIM flight data. Along with that I want to thank Lindsey Steven from the RPI Lab Staff for giving me access to the Linux servers used to run all my programs and help with any server related issues.

Finally, and most importantly, I thank my parents. Every opportunity I have ever had, especially this opportunity to contribute to science, is because of the sacrifices they made and the support they gave me. I owe them everything.

ABSTRACT

Modern air traffic management distributes authority across many controllers and facilities, but within each sector a single controller is the sole authority for separation and sequencing. As traffic volumes grow and next generation concepts such as NASA's Advanced Air Mobility push toward increasingly autonomous flight, this per sector human dependency becomes a scaling bottleneck. This thesis explores a multi stage pipeline for adaptive airspace sectorization and decentralized admission control that can reduce per sector workload without removing human oversight.

Partitioning airspace into sectors is not straightforward. The grid size affects workload, handoff frequency, and the capacity of whatever coordination mechanism operates within each sector. The right partition depends on traffic density, flow direction, and altitude distribution, and the consensus protocol itself introduces tunable parameters that interact with the sectorization choice. Finding good configurations for both requires a data driven approach.

We begin by simulating replays of real FAA SWIM flight plan trajectories across a three dimensional discretized airspace grid. By scoring every valid grid partition on safety, efficiency, and density balance criteria, the simulator produces a labeled dataset mapping traffic features to optimal sectorization grids. A two stage XGBoost predictor trained on this dataset first determines whether sectorization is needed, then states the optimal grid configuration. The predicted grid defines sector boundaries that are handed to a Paxos based consensus protocol, in which the aircraft coordinate sector transitions. Because the protocol exposes several tunable parameters, a Bayesian optimization loop driven by a Gaussian Process surrogate, searches for configurations that maximize entry success rate while rejecting any setting that produces a near mid air collision.

Results show that the predictor reliably selects appropriate grid configurations across diverse traffic conditions, the consensus protocol maintains high entry success rates as traffic scales from light to heavy demand, and Bayesian Optimization discovers that each operating environment requires a qualitatively different protocol tuning, confirming that no single default setting is sufficient. Together, these stages demonstrate that machine learned sec-

torization combined with consensus based admission control can adapt to varying traffic conditions while maintaining separation safety, offering a viable path toward autonomous airspace operations.

1 Introduction

1.1 Motivation

Air traffic control operates reliably almost all the time. The problem is what happens when it doesn't. The FAA has reported staffing shortages at major facilities for years. Controllers work overtime and six day weeks just to keep things running. When traffic picks up or weather forces flights off their normal routes, everything gets harder at once. More handoffs, less spacing, less time to think.

The way the system is built, one controller is responsible for every aircraft in the region they monitor. They give each one individual instructions over the radio. On a normal day that is manageable. But it does not take much to push a sector past what one person can handle. A few extra flights or an unexpected reroute and the workload goes from fine to overwhelming fast.

This thesis explores whether some of that coordination can happen between the aircraft themselves. Controllers would still be in charge of policy and exceptions. But if aircraft can share where they are and where they plan to go in the near term, basic separation tasks do not all need to go through one voice on one frequency. The question is whether this kind of decentralized coordination actually works, whether it reduces conflicts and lowers workload in cruising airspace, and whether it can eventually extend to terminal areas too.

1.1.1 Why this matters now

Evidence points to two pressure sources. First, staffing and training: multiple ATC facilities remain below target levels, extending certification times and increasing dependence on overtime [1, 2]. Second, exogenous shocks: in Europe, industrial action has repeatedly produced network level delays and cancellations. This serves as an operations research reminder that thin margins make disruptions even more problematic [3]. Together these motivate objective measures of traffic complexity and controller workload that are comparable across concepts.

1.1.2 Safety signal: close calls and coordination load

Recent runway incursion investigations emphasized workload, communication, and conformance under time pressure. Official dockets (e.g., the Austin near collision case) provide transcripts and timing that illustrate how quickly complexity compounds in busy phases of flight [4]. Our goal is not to relitigate incident causality, but to supply clean, replayable 4D traces and time indexed state so analysts can compute density, proximity, and handoff load and run "what if" boundary tests against the same underlying data.

1.2 Contributions of this Thesis

This thesis investigates whether machine learning and automated optimization can automate routine airspace configuration and sector coordination while preserving human oversight. The central hypothesis is that a data driven pipeline, one that learns sectorization from traffic patterns, coordinates aircraft through consensus, and tunes its own parameters, can match or exceed the safety of manual approaches while scaling to traffic volumes that would overwhelm a single human controller per sector.

Manual sectorization and protocol tuning rely on experience and work for stable, predictable traffic but break down when patterns shift. Machine learning can capture the complex interactions between density, flow geometry, and altitude distribution that manual rules cannot, and Bayesian Optimization can search a high dimensional parameter space far more efficiently than trial and error. The results confirm that both deliver: the predictor generalizes to unseen airports and the optimizer discovers that each environment needs its own configuration.

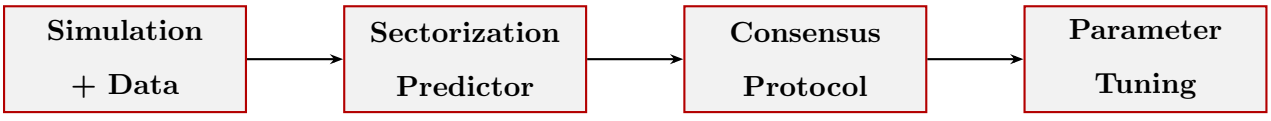


Fig. 1.1 Pipeline overview.

The pipeline has four stages (Figure 1.1). First, we replay real FAA SWIM trajectories through a three dimensional airspace simulation that scores every valid grid partition on

safety, efficiency, and density balance, producing a labeled dataset that maps traffic features to optimal sectorization grids.

Second, an XGBoost predictor trained on this dataset determines whether sectorization is needed and, if so, selects the optimal grid configuration. The hypothesis was that engineered traffic features like density, flow directionality, and altitude complexity contain enough signal to predict good sectorizations without rerunning the full simulation at inference time. The results confirm this: the predictor generalizes across airports it was not trained on.

Third, the predicted grid defines sector boundaries that are handed to a leaderless Synod consensus protocol (the single decree core of Lamport's Paxos family) in which aircraft coordinate sector entries among themselves, with no permanent coordinator that could become a single point of failure.

Fourth, Bayesian Optimization with a Gaussian Process surrogate automatically tunes the eight protocol parameters for each operating environment, since the results show that no single default setting works across airports.

Together, these stages form a closed loop: simulation generates training data, machine learning predicts sectorization, consensus coordinates aircraft, and optimization tunes the protocol.

1.3 Structure of this thesis

The layout of this thesis is as follows. Section 2 describes the core methodology for sector configuration, including the airspace discretization, safety metrics, and traffic data preparation pipeline. Section 3 presents the XGBoost predictive model that maps traffic features to optimal sectorization decisions. Section 4 introduces the Paxos based decentralized consensus protocol that lets aircraft coordinate sector transitions among themselves. Section 5 describes the Bayesian Optimization framework used to automatically tune the consensus protocol for different airports and traffic densities. Section 6 discusses the limitations of the current system. Section 7 outlines extensions and future work, including a Multi Agent Reinforcement Learning formulation. Section 8 concludes the thesis, and Section 9 is the appendix where we share other results.

2 Sector Configuration: Core Methodology

2.1 Chapter contributions

This section explains how we turn real traffic seeds into a concrete sector layout recommendation. We start with a brief historical context, then formalize notation, describe how we generate candidate 3D grids, how we simulate traffic over each grid, what we measure (safety and workload), and how we score and select a configuration. Throughout, we keep assumptions minimal and auditable.

2.2 Background

Sectorization is how we divide airspace so responsibilities for surveillance, separation, and sequencing are clear. The design is always a trade off: sectors must keep *safety* margins (separation minima), stay within *controller workload* limits, and preserve *efficiency* (throughput, limited path stretching). In practice, boundaries and staffing are still largely planned from historical demand patterns and facility constraints, then tweaked tactically day to day. That approach works on average days, but it can struggle when traffic patterns shift, weather compresses flows, or staffing is tight [1, 2, 3].

2.2.1 Sectorization goals and current boundary practices

The operational baseline is simple: keep aircraft separated (horizontally and vertically) while maintaining orderly flows. Controller procedures encode the minima (e.g., typical terminal and en route radar separations) and the rules for coordination and handoffs [5]. Even when automation helps with detection or metering, the controller remains responsible for the sector's safety and the timing of transfers to neighbors. From a configuration perspective, the questions are: (i) how big should sectors be, (ii) where should boundaries fall relative to dominant flows, and (iii) how should we balance load so no one sector becomes the bottleneck?

Most boundaries reflect a mix of history (legacy routes, traffic counts), geography (fixes,

navaids, flow gates), and staffing realities. On a good day, that yields predictable coordination patterns. On a bad day, sudden weather changes, reroutes, demand spikes, or short staffing, those same static boundaries can concentrate complexity and increase handoffs across already busy edges [1, 2, 3]. This is why the literature has repeatedly explored more *dynamic* approaches to sector design and allocation.

2.2.2 Complexity metrics and dynamic sectorization

A sector's "difficulty" is not just the number of aircraft, geometry and intent matter. Research on *dynamic density* treats workload as a weighted combination of traffic count and factors like proximity, closure rates, heading/speed changes, and crossing streams [6, 7]. A generic form is:

$$\mathrm{DD}(t) \;=\; \sum_{k} w_k \, f_k\big(\mathcal{S}(t)\big),$$

where $\mathcal{S}(t)$ is the sector state at time t (tracks, headings, speeds, conflicts), $f_k(\cdot)$ are measurable features (e.g., aircraft count, pairs within d NM, crossing angles), and w_k are weights calibrated from controller studies. While implementations differ, the core point is stable: similar counts can yield very different workloads depending on geometry and intent.

Two lines of work recur in the literature. First, *algorithmic sectorization*: methods that place or move boundaries to equalize load, minimize inter sector crossings, or align with dominant flows (graph cuts, clustering, optimization) [8]. Second, *dynamic airspace configuration (DAC)*: time varying sector shapes/allocations tailored to evolving traffic and weather [9]. Both families report potential improvements, but with warnings: you must protect separation rules and minimize churn in coordination patterns; you also need robust, standardized data so results aren't tool specific. European network reports and U.S. planning documents point to the same takeaway: resilience improves when you can see complexity building and adjust boundaries or staffing proactively [2, 3].

2.2.3 Data foundations and motivating gaps

Objective analysis depends on authoritative, machine readable data. In the U.S., *SWIM* provides operational information services; flight intent/state are exchanged using the *Flight Information Exchange Model (FIXM)* [10, 11]. *NASR* publishes the official catalog of fixes, navaids, and related aeronautical data [12]. Terminal procedures and runway geometry are distributed in the *FAACIFP18* (CIFP) dataset [13, 14]. Combining these lets you reconstruct 4D tracks anchored to real fixes and procedures, then compute time indexed measures (density, proximity, handoffs) consistently across regions (terminal adjacent or pure en route).

Three practical gaps motivate a clean, replayable baseline before proposing any new control concept. (1) **Static designs under stress:** fixed boundaries can concentrate complexity when flows shift; you need measurements that reveal this early [3]. (2) **Workforce constraints:** persistent staffing shortfalls raise fatigue/coordination risks, making balanced sectors and predictable handoffs more valuable [1, 2]. (3) **Data fragmentation:** without a transparent pipeline from SWIM/FIXM through NASR/CIFP, comparisons devolve into tool fights. A standardized, auditable reconstruction layer avoids that [10, 11, 12, 13].

2.3 Preliminaries: Domain, State, and Discretization

This subsection sets the mathematical outline we use throughout: we define the spatiotemporal domain (airspace and time), specify flight state variables, describe how the airspace is discretized into sectors, and formalize adjacency. The outputs here feed the safety/workload metrics in §2.4 and, later, the learning and search components (Sections 3 and 4).

2.3.1 Airspace region

We model the operational airspace as a bounded three dimensional region defined by geographic limits and altitude constraints, the place where sectoring and traffic management actually occur. Formally,

$$\mathcal{A} \subset [-90, 90] \times [-180, 180] \times \mathbb{R}_{\geq 0}$$

denotes the study region in (ϕ, λ, h) (latitude [deg], longitude [deg], altitude [ft]), with altitude band $[H_{\min}, H_{\max}]$.

2.3.2 Temporal and state representation

We analyze traffic on a discrete time line with fixed step Δt, over a finite horizon T:

$$\mathcal{T} = \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + N\Delta t\}, \qquad N = \left\lfloor T/\Delta t \right\rfloor \tag{2.1}$$

This makes all counts and rates explicitly time indexed and comparable.

At each $t \in \mathcal{T}$, let $\mathcal{F}(t)$ be the set of flights active in $\mathcal{A}$ and $\mathcal{F} = \bigcup_{t\in\mathcal{T}} \mathcal{F}(t)$ the union over the horizon. Each flight $f \in \mathcal{F}(t)$ has state

$$\mathbf{x}_f(t) = \big(\phi_f(t),\, \lambda_f(t),\, h_f(t),\, v_f(t),\, \psi_f(t)\big), \tag{2.2}$$

where v_f is ground speed [kts] and ψ_f is heading [deg].

2.3.3 Candidate sector grid and adjacency structure

To make the problem tractable we partition $\mathcal{A}$ into a 3D grid $G = (R, C)$ with R rows, C columns, and a *fixed* $L = 3$ altitude layers based on standard airspace stratification:

- Layer 0: [0, 9,999] ft, terminal operations (Class B/C/D airspace)
- Layer 1: [10,000, 17,999] ft, transition altitude
- Layer 2: [18,000, ∞) ft, cruise operations (Class A airspace)

This yields $M = 3RC$ sectors. The sector indexer maps positions to sector IDs:

$$s:\ \mathcal{A} \to \{1, \ldots, M\}, \qquad s(\phi, \lambda, h; G) = k \tag{2.3}$$

The k-th sector volume is then

$$S_k = \big\{(\phi, \lambda, h) \in \mathcal{A}\ :\ s(\phi, \lambda, h; G) = k\big\} \tag{2.4}$$

Convention. The indexer uses only position components (ϕ, λ, h). When needed later, writing $s(\mathbf{x}_f(t); G)$ is shorthand for $s(\phi_f(t), \lambda_f(t), h_f(t); G)$.

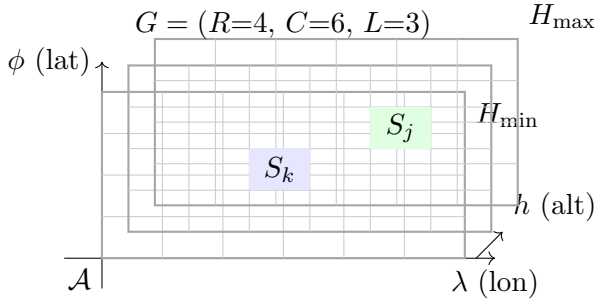


Fig. 2.1 Study region $\mathcal{A}$ and candidate discretization $G = (R, C, L)$. Layers illustrate altitude bands; grid cells are sectors S_k.

Once the grid is defined, the next step is to formalize how its sectors connect to one another in three dimensions. Sectors k and j are adjacent if they share a face or edge. Let the grid indices be (r_k, c_k, ℓ_k) and (r_j, c_j, ℓ_j). Then

$$\begin{aligned} k \sim j \iff & \text{(horizontal adjacency)} \quad \ell_k = \ell_j \text{ and } \max(|r_k - r_j|, |c_k - c_j|) = 1, \\ & \text{or} \\ & \text{(vertical adjacency)} \qquad r_k = r_j,\ c_k = c_j, \quad \text{and } |\ell_k - \ell_j| = 1 \end{aligned} \tag{2.5}$$

i.e., horizontal neighbors are 8 connected (N, S, E, W, and four diagonals within the same layer), while vertical neighbors share the same (r, c) cell in adjacent altitude layers.

2.4 Separation, Safety, and Operational Metrics

We now build operational metrics from the primitives above. Occupancy and handoffs quantify workload; risk pairs measure predictive safety; aggregate statistics capture system wide balance. These metrics will define the objective and constraints used in our configuration search (later in this chapter) and will also serve as labels/targets for predictive modeling in Section 3.

2.4.1 From separation primitives to sector dynamics

Great circle horizontal distance and vertical separation for aircraft i, j are

$$d_{ij}(t) = \mathrm{GC}\big((\phi_i(t), \lambda_i(t)), (\phi_j(t), \lambda_j(t))\big) \quad [\mathrm{NM}], \tag{2.6}$$

$$\Delta h_{ij}(t) = \big|h_i(t) - h_j(t)\big| \quad [\mathrm{ft}] \tag{2.7}$$

Let $d_{\min}^{\mathrm{hor}}$ and $h_{\min}^{\mathrm{vert}}$ denote the applicable minima (region/phase dependent per [5]).

We detect a pointwise violation via:

$$\mathbb{I}_{ij}(t) = \mathbf{1}\big\{d_{ij}(t) < d_{\min}^{\mathrm{hor}} \ \wedge \ \Delta h_{ij}(t) < h_{\min}^{\mathrm{vert}}\big\} \tag{2.8}$$

These primitives form the building blocks for higher level operational metrics that describe how sectors behave and how safety and workload evolve over time.

We now define operational metrics built from these primitives. *Sector occupancy* and *hand-off counts* quantify workload; risk pairs measure predictive safety; and aggregate statistics capture system wide balance.

2.4.2 Sector load and predictive safety metrics

For brevity, let

$$s_f(t) \ := \ s\big(\phi_f(t), \lambda_f(t), h_f(t); \, G\big)$$

denote the sector assignment of flight f at time t.

Occupancy.

$$N_k(t) = \sum_{f \in \mathcal{F}(t)} \mathbf{1}\{\, s_f(t) = k \,\} \tag{2.9}$$

Handoffs. We distinguish two types of sector transitions. A *horizontal handoff* occurs when a flight crosses a geographic sector boundary (change in row or column index) while remaining in the same altitude layer:

$$H^{\mathrm{hor}}(t) = |\{\, f \in \mathcal{F}(t) \ : \ (r_f(t), c_f(t)) \neq (r_f(t-\Delta t), c_f(t-\Delta t)) \ \wedge \ \ell_f(t) = \ell_f(t-\Delta t) \,\}| \tag{2.10}$$

A *vertical handoff* occurs when a flight transitions between altitude layers:

$$H^{\text{vert}}(t) = |\{\, f \in \mathcal{F}(t) \;:\; \ell_f(t) \neq \ell_f(t - \Delta t)\,\}| \tag{2.11}$$

The total handoff count at time t is simply

$$H_{\text{total}}(t) = H^{\text{hor}}(t) + H^{\text{vert}}(t) \tag{2.12}$$

While occupancy and handoff rates capture current sector load, a complementary view considers future conflicts: how many aircraft pairs are on track to violate separation within a look ahead window. With horizon $\tau > 0$, the set of pairs predicted to violate separation within $[t, t+\tau]$ is

$$\mathcal{RP}(t;\tau) = \Big\{(i,j) :\; i<j,\; i,j \in \mathcal{F}(t),\; \exists\, t' \in [t,t+\tau] \cap \mathcal{T} \text{ s.t. } \mathbb{I}_{ij}(t') = 1\Big\} \tag{2.13}$$

Plain English: flights that are forecast to break minima within the look ahead window.

2.4.3 Workload evaluation and configuration optimization

Let $\bar{N}(t) = \frac{1}{M}\sum_{k=1}^{M} N_k(t)$. We use:

$$\sigma_N(t) = \sqrt{\frac{1}{M}\sum_{k=1}^{M}\big(N_k(t) - \bar{N}(t)\big)^2}, \qquad N_{\max}(t) = \max_{k\in\{1,\dots,M\}} N_k(t) \tag{2.14}$$

A feature driven proxy ("dynamic density") is

$$\text{DD}_k(t) = \sum_m w_m\, f_m(\mathcal{S}_k(t))\,, \tag{2.15}$$

where $f_m(\cdot)$ may include counts, proximity pairs, crossing angles, closure rates, etc. This follows established ATM complexity models [6], [7].

Use in this thesis: we primarily evaluate configurations with σ_N and $N_{\max}$; DD_k is used when a richer, feature based workload model is needed.

These workload measures, together with safety indicators, form the objective landscape

over which sector configurations are optimized. Let Θ denote the space of allowable sector configurations; each $\theta \in \Theta$ specifies a discretization (e.g., grid dimensions R, C, L) and any associated hyperparameters (e.g., thresholds or weights). An objective $J : \Theta \to \mathbb{R}$ scores a configuration using the safety/workload metrics above.

2.5 Traffic Data Preparation and Conflict Free Dataset Generation

The configuration search and learning methods in subsequent chapters (§3, §5) require clean, conflict free baseline traffic scenarios. Real operational data often contains conflicting flight plans, incomplete state information, or trajectories that would violate separation minima if simulated as is. This subsection describes the pipeline from raw SWIM messages to simulation ready datasets: we ingest and parse FIXM encoded flight data, apply spatial filtering, resolve waypoint coordinates, and resolve pairwise conflicts by adapting a formally verified speed assignment algorithm. The result is a validated set of flight seeds suitable for systematic configuration evaluation. Figure 2.2 illustrates the complete pipeline.

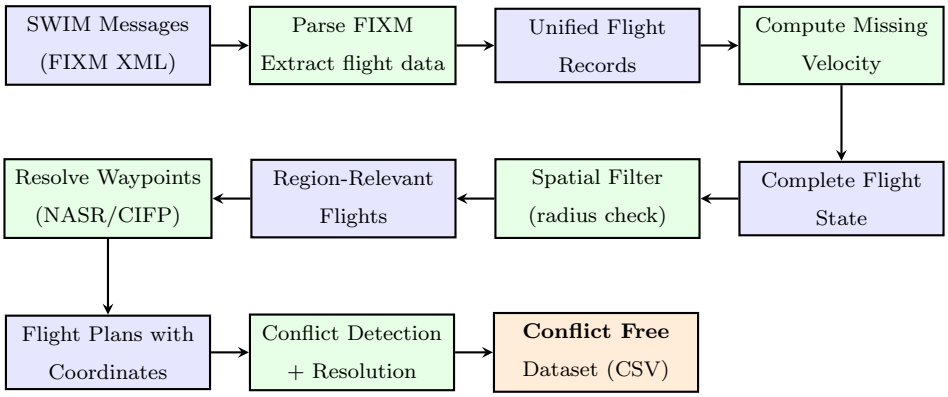


Fig. 2.2 Data Preparation Pipeline.

Raw SWIM messages are processed through parsing, aggregation, velocity reconstruction, spatial filtering, waypoint resolution, and conflict resolution to produce a validated, conflict free dataset for configuration evaluation

2.5.1 SWIM data ingestion and spatial filtering

Operational flight intent and state data in the U.S. National Airspace System are distributed via the System Wide Information Management (SWIM) framework using the Flight Information Exchange Model (FIXM) [10, 11]. FIXM messages encode flight plans, position updates, and trajectory changes in XML format following NAS FIXM 3.0 specifications. Our pipeline begins by parsing these XML messages to extract:

- **Flight identification**: Globally Unique Flight Identifier (GUFI) and aircraft callsign.
- **Route information**: Departure and arrival airports, textual route descriptor, and expanded route waypoints with estimated times of arrival.
- **Position snapshots**: Latitude, longitude, altitude, and optionally velocity components (ground speed, heading).
- **Aircraft metadata**: ICAO aircraft type designator, assigned/cleared altitude.

A single flight may generate multiple SWIM messages over time (track updates, flight plan amendments). We aggregate messages by callsign to construct a unified view of each flight's latest known state and intent.

Velocity reconstruction. SWIM position reports do not always include heading and groundspeed explicitly. When these fields are missing, we compute them from consecutive position snapshots using great circle formulas. Given two positions $p_1 = (\phi_1, \lambda_1, t_1)$ and $p_2 = (\phi_2, \lambda_2, t_2)$ separated by time interval $\Delta t_{\text{obs}} = (t_2 - t_1)$, the heading (true bearing from p_1 to p_2) is:

$$\psi = \text{atan2}\big(\sin(\Delta\lambda) \cdot \cos(\phi_2),\ \cos(\phi_1) \cdot \sin(\phi_2) - \sin(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\Delta\lambda)\big), \tag{2.16}$$

where $\Delta\lambda = \lambda_2 - \lambda_1$, and the result is normalized to $[0, 360)$ degrees. Ground speed is computed as:

$$v_g = \frac{\text{GC}(p_1, p_2)}{\Delta t_{\text{obs}}/3600} \quad [\text{kts}], \tag{2.17}$$

where $\mathrm{GC}(p_1, p_2)$ is the great circle distance in nautical miles. This approach reconstructs missing kinematic data with acceptable accuracy for trajectory simulation, provided Δt_{obs} is reasonable (typically $1 - 600$ seconds).

Spatial filtering. To focus on a specific airspace region (e.g., terminal adjacent or en route sectors near a major airport), we apply a radius based filter. Given a reference point (either an airport's published coordinates or a custom center (ϕ_c, λ_c)) and radius r_{NM}, we retain only those flights f satisfying:

$$\mathrm{GC}\big((\phi_f, \lambda_f), (\phi_c, \lambda_c)\big) \leq r_{\text{NM}} \tag{2.18}$$

This ensures that downstream processing focuses on traffic relevant to the study region.

Waypoint resolution. Expanded route waypoints from FIXM reference navigation fixes by name. We resolve each fix name to coordinates (ϕ, λ) using the FAA's National Airspace System Resources (NASR) database [12], which provides authoritative positions for fixes, navaids, and waypoints. Any waypoints that cannot be resolved are logged and either skipped (if optional procedural waypoints) or cause the flight to be dropped (if part of the core en route structure). For terminal procedures, we augment routes with STAR (Standard Terminal Arrival Route) waypoints extracted from the CIFP (Computer Navigable Procedure) dataset [13], applying published altitude and speed constraints at each procedure waypoint to improve trajectory fidelity.

2.5.2 Flight plan segments and conflict detection

Once SWIM data are parsed and filtered, we convert each flight into an internal segment based representation suitable for pairwise conflict detection. A flight plan is decomposed into consecutive segments, each connecting two waypoints. Formally, let flight f have waypoints $w_0, w_1, \ldots, w_n$. The i-th segment spans from w_i to w_{i+1} and is characterized by:

- Start and end positions: $(\phi_{\text{start}}, \lambda_{\text{start}}, h_{\text{start}})$ and $(\phi_{\text{end}}, \lambda_{\text{end}}, h_{\text{end}})$
- Ground speed v_g [kts] and vertical speed v_z [ft/min]
- Heading ψ [deg] and segment duration δt [s]

This representation preserves the geometric path (waypoint sequence) while allowing speed variations to be introduced for conflict resolution.

Problem statement. Given a set of existing conflict free flight plans Φ and a proposal flight plan F_a, determine whether $\Phi \cup \{F_a\}$ remains conflict free with respect to separation standards $D_{\min}^{\text{hor}}$ (horizontal) and $H_{\min}^{\text{vert}}$ (vertical). If conflicts exist, find a modified plan $\bar{F}_a$ such that $\Phi \cup \{\bar{F}_a\}$ is safe.

We adopt the conflict detection methodology developed by Saswata Paul [15], which provides formally verified correctness guarantees. The core idea is to sample the time intervals between waypoint events and check for separation violations using a piecewise linear motion model. Algorithm 1 summarizes the pairwise detection logic.

Algorithm 1 Pairwise Conflict Detection [15]

```
function CONFLICT(F_a, F_b, D_NM, H_ft)
    T ← GETALLTIMES(F_a, F_b)              ▷ Merge waypoint times from both plans
    for each consecutive pair (t_i, t_{i+1}) ∈ T do
        if EXISTSVIOLATION(t_i, t_{i+1}, F_a, F_b, D, H) then
            return True                    ▷ Conflict detected
        end if
    end for
    return False                           ▷ No conflicts found
end function
```

The GETALLTIMES function constructs a non decreasing vector of all waypoint times from both flight plans. For each time interval $[t_i, t_{i+1}]$, EXISTSVIOLATION interpolates aircraft positions using the segment velocities and checks whether horizontal and vertical separation fall below minima. Specifically, it evaluates:

$$\exists\, t \in (t_i, t_{i+1}] : \ \|s_{xy}(t)\| < D_{\min}^{\text{hor}} \ \wedge \ |s_z(t)| < H_{\min}^{\text{vert}},$$

where $s_{xy}(t)$ and $s_z(t)$ are the relative horizontal and vertical positions at time t, computed from the linear motion model. This approach is conservative (it may over approximate

conflicts slightly due to discretization), but it is sound: if CONFLICT returns `False`, the two plans are guaranteed conflict free under the piecewise linear assumption.

2.5.3 Conflict resolution via backtracking speed assignment

When a proposed flight plan F_a conflicts with the existing safe set Φ, we apply a conflict resolution strategy that adjusts segment ground speeds to eliminate violations. This method is adapted from Paul's formally verified flight planning algorithm [15], which guarantees both *safety* (if a solution is found, it is conflict free) and *completeness* (if no solution exists within the search space, the algorithm correctly reports failure).

Speed assignment as a combinatorial search. Let F_a have n segments and let $\xi = \{\xi_1, \xi_2, \ldots, \xi_m\}$ be a discrete set of allowable ground speeds (e.g., $[120, 140, 160, \ldots, 300]$ kts in 20 knot increments for typical en route cruise). We represent a candidate solution as an assignment matrix $M \in \{0,1\}^{n \times m}$, where $M[i,j] = 1$ indicates that segment i is assigned speed ξ_j. A complete assignment satisfies exactly one $M[i,j] = 1$ per row.

Vertical speed adjustment. When a segment's ground speed is changed from v_g to v_g', the vertical speed must be adjusted to preserve the vertical flight path angle $\gamma = \arctan(v_z/v_g)$. The corrected vertical speed is:

$$v_z' = v_g' \times \frac{v_z}{v_g} \tag{2.19}$$

This maintains the altitude profile while allowing temporal adjustments through speed changes. Figure 2.3 illustrates how speed adjustment eliminates a conflict between two flight plans.

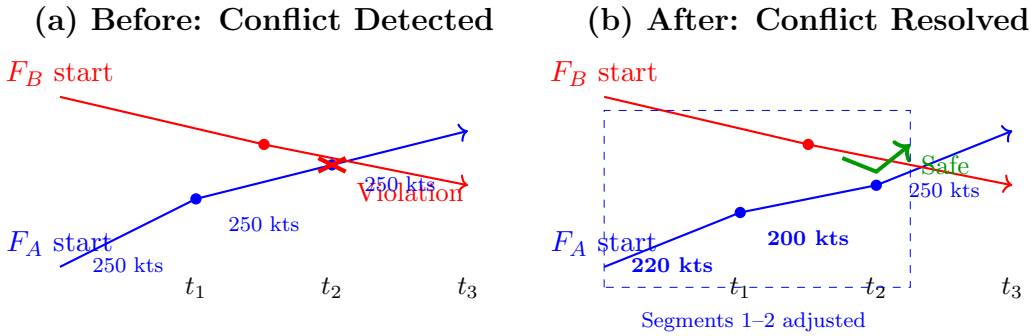


Fig. 2.3 Conflict Resolution Example. (a) Two flight plans F_A (blue) and F_B (red) with a predicted separation violation at t_2. (b) After applying Paul's backtracking algorithm, segments 1–2 of F_A are assigned reduced speeds (220 and 200 kts), delaying F_A's arrival at the conflict point and eliminating the violation. Flight F_B remains unchanged.

Backtracking search. The core resolution algorithm (SOLVE) explores the assignment space via backtracking. It attempts to assign a speed to each segment sequentially, checking at each step whether the partial assignment yields a safe intermediate flight plan (no conflicts with Φ) and whether the assignment allows valid assignments for subsequent segments. Algorithm 2 shows how we apply this to batch traffic processing.

Algorithm 2 Batch Conflict Resolution Pipeline

$\Phi_{\text{safe}} \leftarrow \emptyset$ ▷ Conflict free flight set
$\xi \leftarrow [120, 140, \ldots, 300]$ kts ▷ Discrete speed options
for each flight seed F in input dataset **do**
    if SAFE$(\Phi_{\text{safe}} \cup \{F\}, D, H)$ **then**
        $\Phi_{\text{safe}} \leftarrow \Phi_{\text{safe}} \cup \{F\}$ ▷ No conflict, add as is
    else
        $M_0 \leftarrow$ empty matrix ($n \times m$ zeros)
        $M^* \leftarrow$ SOLVE$(M_0, F, \Phi_{\text{safe}}, \xi, D, H)$ ▷ Paul's algorithm [15]
        if $M^* \neq$ `null` **then**
            $F' \leftarrow$ PLANFROMMATRIX(M^*, F, ξ) ▷ Apply speed assignment
            $\Phi_{\text{safe}} \leftarrow \Phi_{\text{safe}} \cup \{F'\}$
        else
            Drop flight F ▷ Unresolvable within speed options
        end if
    end if
end for
return Φ_{safe}

The SOLVE function (detailed in Paul's dissertation, Chapter 6, Figures 6.13 and supporting lemmas [15]) recursively tries each speed option for the first unassigned segment, checking whether the resulting partial plan is safe and whether subsequent segments can also be assigned safely. If a complete, conflict free assignment is found, it is returned; otherwise, the algorithm backtracks and tries alternative speeds. Our implementation follows Paul's specification closely, with the following practical modifications:

- **Iteration cap:** We impose a maximum iteration count (default 10,000) to prevent excessive computation on degenerate cases.
- **Drop unresolvable flights:** Unlike real time flight planning (where rejection is costly), our preprocessing pipeline can drop flights that cannot be resolved within the given speed options, ensuring the final dataset is strictly conflict free.

- **Batch incremental construction:** We process flights one by one, building Φ_{safe} incrementally. This mirrors operational arrival sequencing and allows early conflicts to be resolved before later flights arrive.

Correctness guarantees. Because we use Paul's formally verified core logic, Algorithm 2 inherits the safety property: if a flight is added to Φ_{safe} (either as is or after resolution), the resulting set is guaranteed conflict free. The completeness property ensures that if a resolvable speed assignment exists within ξ, the algorithm will find it (subject to the iteration cap).

2.5.4 Dataset export and quality assurance

Once all flights have been processed, we export the conflict free dataset as a CSV file with the following schema:

- **GUFI**: Globally unique flight identifier.
- **Callsign**: Aircraft identification.
- **Departure/Arrival**: ICAO airport codes.
- **Start state**: Latitude, longitude, altitude, heading, ground speed, and time offset (seconds from scenario $t = 0$).
- **Route waypoints**: Ordered list of (fix name, lat, lon, altitude constraints, speed constraints).
- **Aircraft type**: ICAO designator (for future performance modeling).

Quality metrics. For each preprocessing run, we report:

Table 2.1: Dataset Quality Metrics (Example)

Metric	**Count (%)**
Total SWIM messages ingested	120
Flights after aggregation	60
Passed spatial filter	50
Conflict free (as is)	46
Resolved via speed adjustment	13
Unresolvable (dropped)	3
Final dataset size	**47 flights**

Validation checks. Before exporting, we perform final validation:

1. **Pairwise conflict check:** Re run $\textsc{safe}(\Phi_{\text{safe}}, D, H)$ to confirm no conflicts exist in the final set.

2. **Segment validity:** Verify all segments have positive durations and ground speeds within operational bounds (e.g., 100 to 600 kts).

3. **Altitude/speed constraint preservation:** Confirm that STAR procedure constraints and assigned altitudes are maintained after speed adjustments.

Integration with subsequent chapters. The conflict free dataset produced by this pipeline serves as input to:

- **Section 3 (XGBoost):** Supervised learning models use this traffic as ground truth to predict workload and conflicts under various sector configurations.

- **Section 5 (Bayesian Optimization):** Configuration search evaluates candidate grids by simulating this traffic scenario and measuring safety/workload metrics (§2.4).

By ensuring the baseline traffic is conflict free and well resolved spatially, we isolate configuration effects from data artifacts and enable fair comparisons across different sector layouts.

3 Predictive Workload Modeling with XGBoost

3.1 Chapter contributions

This chapter trains a two stage XGBoost classifier that predicts the optimal sector grid directly from traffic features, eliminating the need to rerun the full simulation at inference time.

3.2 Overview

This chapter develops a supervised learning model that predicts the optimal horizontal sector grid configuration directly from airspace traffic features. Given a snapshot of traffic state, the model outputs the grid dimensions (R^*, C^*) that minimize the workload objective $J(\theta)$ defined in §2.4. We describe the label generation process, dataset construction, feature engineering, a two stage model architecture, training protocol, results, feature importance analysis, and runtime integration with the configuration search of Chapter 5.

Role in the pipeline. This chapter sits between data collection and the consensus protocol. The data pipeline from the previous chapter produces labeled traffic snapshots. The XGBoost model trained here takes those features and predicts the best grid configuration, which then defines the sector boundaries that the consensus protocol in Chapter 4 uses to coordinate aircraft.

3.3 Why XGBoost

The prediction task defined above maps 23 heterogeneous traffic features (counts, distances, angular spreads, and time of day indicators) to a grid label. This tabular, fixed width input has no spatial or sequential structure that convolutional or recurrent architectures could exploit. Recent large scale benchmarks confirm that gradient boosted tree ensembles consistently match or exceed deep learning accuracy on such tabular datasets [16].

Within the tree ensemble family, XGBoost's boosting strategy focuses successive trees on the hardest to classify examples, which is advantageous here because the high density

configurations that matter most for safety are also the rarest in the training set. Random Forests, by contrast, draw bootstrap samples uniformly and can under represent minority classes [17]. LightGBM and CatBoost offer comparable accuracy on moderate sized datasets, but XGBoost was preferred for its reproducibility of results across platforms.

Linear and logistic models were ruled out because the mapping from features to optimal grid is non linear: the interaction between aircraft density, geometric spread, and altitude distribution cannot be captured by additive terms alone. Neural networks were considered but rejected given the moderate dataset size ($\sim$25 grid configurations $\times$ multiple traffic snapshots), the absence of spatial structure to exploit, and the higher risk of overfitting. XGBoost additionally provides native feature importance scores and is compatible with SHAP based interpretability, which we use in §3.10.

Finally, inference requires only a single tree traversal per stage, making it fast enough to sit inside the Bayesian Optimization screening loop described in Chapter 5.

3.4 Problem Framing and Labels

Prediction task. Given a snapshot of airspace traffic features at time t, predict the optimal horizontal grid configuration (R^*, C^*) that minimizes the composite workload objective $J(\theta)$ introduced in §2.4. Vertical layering is held fixed at $L = 3$ layers (0–9 999 ft, 10 000–17 999 ft, 18 000+ ft) for all configurations.

Label generation. For each traffic snapshot, we perform an exhaustive search over all grid configurations $(R, C) \in \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\}$ using the safety and workload metrics from §2.4. The best configuration is defined as:

$$(R^*, C^*) = \arg\min_{(R,C)} J(R, C, L{=}3), \tag{3.1}$$

where J is the composite objective combining occupancy variance, handoff counts, and risk pair penalties.

Configuration space bounds. We restrict the search space to configurations with at most 5 rows and 5 columns, yielding a maximum of $5 \times 5 = 25$ horizontal sectors (75 total

volumes when accounting for 3 altitude layers). This upper bound reflects several operational and computational constraints:

1. **Handoff frequency scaling.** The number of sector boundaries grows as $O(R + C)$ for an $R \times C$ grid. Each boundary crossing requires a handoff: the transfer of communication, control responsibility, and situational awareness. Beyond 5×5, the handoff rate becomes operationally burdensome: aircraft in a 10×10 grid would cross roughly $4\times$ as many boundaries as in a 5×5 grid for the same trajectory, significantly increasing coordination overhead.

2. **Consensus and coordination complexity.** In decentralized architectures where sectors must reach agreement on spacing, sequencing, or conflict resolution, message complexity scales with the number of adjacent sector pairs. For a 2D grid, each interior sector has up to 4 neighbors (8 if diagonals are considered), yielding $O(RC)$ coordination links. Finer grids amplify latency and the probability of coordination failures, particularly under high traffic loads.

3. **Minimum viable sector dwell time.** A sector must be large enough for an aircraft to spend meaningful time under its jurisdiction. At typical en route speeds (400 to 500 kts), a sector spanning only 10 to 15 NM would yield dwell times under 2 minutes, which is insufficient for controllers (or automation) to build situational awareness, issue clearances, and coordinate handoffs. The 5×5 bound ensures minimum sector dimensions remain operationally viable for the airspace regions studied.

4. **Diminishing returns in load balancing.** Workload variance across sectors decreases with finer partitioning, but the marginal benefit is expected to diminish while coordination costs grow. Beyond 5×5, the additional workload balancing gains are likely outweighed by increased handoff frequency and inter sector coordination overhead.

Target variable. The target is the configuration class (R, C) drawn from the 25 configurations in the dataset. We treat this as a multi class classification problem with Stage 1

distinguishing 1×1 from non 1×1, and Stage 2 predicting the exact configuration among the 24 non 1×1 classes.

Formal problem statement. Let $\mathbf{z} \in \mathbb{R}^d$ be the feature vector extracted from a traffic snapshot (defined in §3.6). The learning task is a two stage classification:

$$f_{\text{stage1}} : \mathbb{R}^d \to \{0, 1\}, \tag{3.2}$$

$$f_{\text{stage2}} : \mathbb{R}^d \to \{1, \ldots, 24\}, \tag{3.3}$$

where Stage 1 predicts whether the optimal configuration is 1×1 (class 0) or requires partitioning (class 1), and Stage 2 predicts the specific non 1×1 configuration class.

Class distribution analysis. To ensure reliable learning across all configurations, we balance the dataset through extended simulation campaigns targeting underrepresented configurations. Figure 3.1 summarizes the resulting distribution across 25 configurations from 1×1 to 5×5. Most configurations receive approximately 2,700 samples each, with 1×1 receiving 5,350 samples (reflecting its higher real world frequency) and 5×5 receiving 300 samples (reflecting its rarity). The total dataset size is ∼65,000 samples, comprising approximately 40,000 real simulation snapshots and 25,000 synthetic samples generated using a physics informed feature synthesis algorithm (§3.5). This distribution enables the model to learn meaningful decision boundaries for all configuration classes while reflecting realistic operational patterns.

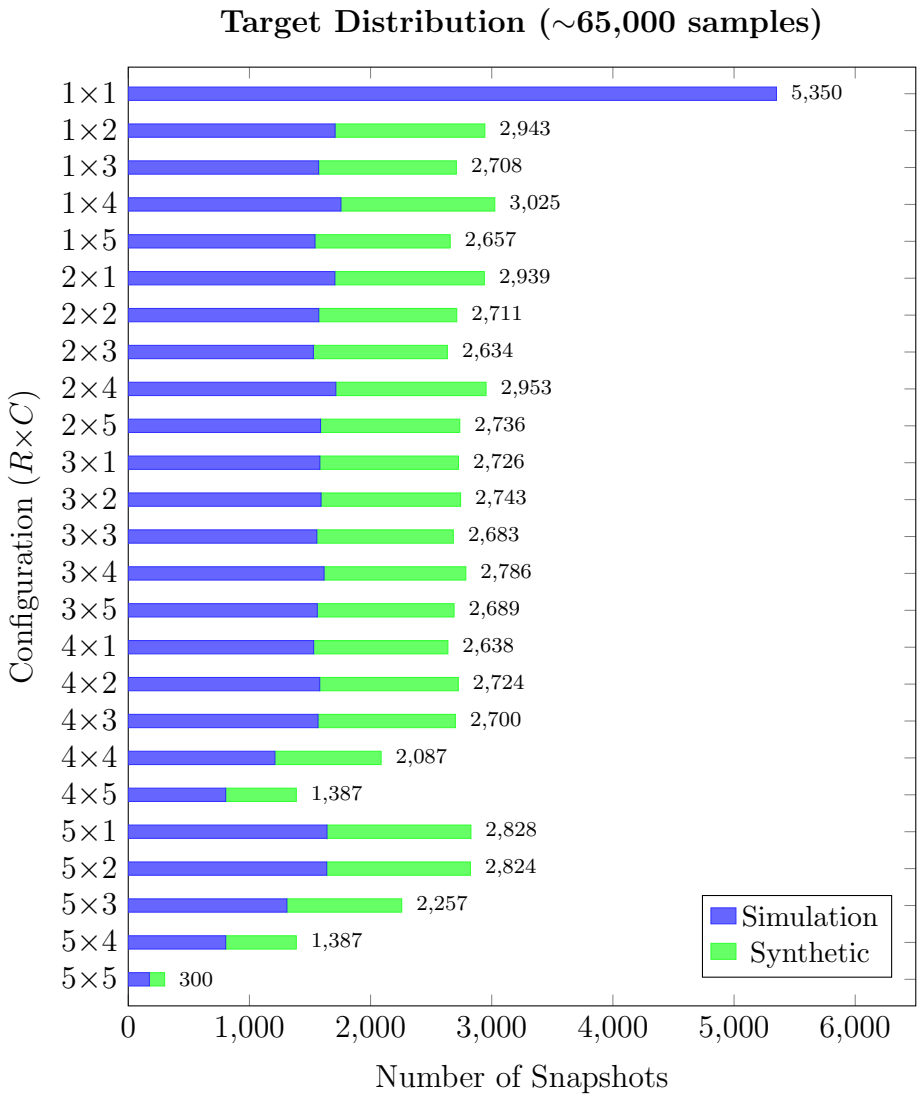


Fig. 3.1 Target configuration distribution. Blue segments represent ∼40,000 simulation snapshots; green segments represent ∼25,000 synthetic samples. 1×1 is entirely simulation data (5,350 samples). Most non 1×1 configurations receive approximately 2,700 samples total. 5×5 has 300 samples (rare). Total: ∼65,000 samples across all 25 configurations.

3.5 Dataset Construction

Data sources. The dataset comprises approximately 65,000 labeled samples drawn from two sources: ∼40,000 traffic snapshots generated from extended simulation campaigns using the conflict free datasets produced by the pipeline in §2.5, and ∼25,000 synthetic samples

generated using a physics informed feature synthesis algorithm. Each sample captures the complete airspace state and is paired with the optimal configuration label obtained via exhaustive grid search over all 25 target configurations.

Balancing through extended simulations and synthetic augmentation. We balance the dataset using a combination of targeted simulation campaigns and synthetic data generation. Simulation campaigns employ:

- **High density scenarios**: Increased aircraft injection rates to produce traffic conditions requiring multi sector configurations.
- **Complex flow patterns**: Scenarios with crossing traffic flows and multi directional operations that necessitate finer grid partitions.
- **Multi altitude operations**: Layered traffic with aircraft at multiple flight levels, exercising the vertical dimension of sectorization.

Synthetic data generation. The real world SWIM data, while rich, captures a limited range of operational conditions: the traffic scenarios tend to cluster around similar density levels, flow patterns, and time of day profiles typical of the observed airspace regions. As a result, certain sectorization configurations, particularly high partition counts such as 4×5, 5×4, and 5×5, are rarely triggered because the observed traffic seldom reaches the density or complexity that would justify them. To enable the model to generalize to realistic but unobserved operating conditions, we employ a class separable feature synthesis algorithm that generates samples with physics informed feature signatures. For a target configuration (R, C), the algorithm constructs each feature using domain knowledge of how traffic characteristics drive sectorization need:

- **Traffic density** scales with total sector count $(R \times C)$ and row count, reflecting that finer partitions are triggered by higher traffic loads. Each configuration receives a distinct base density with controlled noise.
- **Average proximity** decreases with sector count and column count, capturing the operational reality that denser traffic (closer aircraft spacing) motivates more sectors.

- **Altitude mix** increases with row count, as vertically stratified traffic requires more horizontal partitions to manage layered flows.
- **Conflict risk** incorporates grid asymmetry $|R - C|$, reflecting that imbalanced grids create uneven workload distributions.
- **Flow direction and concentration** encode the directional structure of traffic, distinguishing configurations like $N \times M$ from $M \times N$ based on whether traffic flows predominantly along rows or columns.

Each feature is drawn from a Gaussian distribution centered on the configuration specific base value, with standard deviations calibrated to produce realistic spread while maintaining separability between neighboring configurations.

Edge case handling. Extreme configurations receive special attention. The 5×5 configuration (25 horizontal sectors) appears in only 300 samples, reflecting its rarity in operational settings: it requires exceptionally high traffic density across all altitude bands. Similarly, asymmetric configurations such as 4×5 and 5×4 (each with $\sim$1,400 samples) capture scenarios where traffic demand is highly directional, requiring finer partitioning along one axis. Despite their limited representation, the model achieves over 90% accuracy on these rare configurations through the combined real and synthetic training strategy.

Geographic coverage. Samples span diverse geographic regions to ensure the model learns location agnostic patterns rather than memorizing specific airspace structures. The feature set explicitly excludes location identifiers, forcing the model to rely on traffic characteristics alone.

Temporal coverage. Scenarios cover weekday and weekend operations across morning, midday, afternoon, evening, and night periods. Time of day information is encoded through categorical features (`time_morning`, `time_noon`, `time_evening`, `time_night`) and day-type features (`day_weekday`, `day_weekend`).

Train/validation/test split. The $\sim$65,000 samples are divided 70/15/15 into training, validation, and test sets ($\sim$45,800 / $\sim$9,800 / $\sim$9,800) using stratified random sampling to preserve class proportions across splits.

3.6 Features: Counts, Proximity, Flow Geometry

We extract 23 location agnostic features from each traffic snapshot, organized into three groups: raw airspace features, engineered features, and time/day categorical encodings. By excluding geographic identifiers, the model learns generalizable patterns applicable to any airspace region. All notation follows §2.3–§2.4.

3.6.1 Raw airspace features (8)

1. **Traffic density**: `traffic_density` $= |\mathcal{F}(t)|$, the number of active aircraft in the study region at time t.

2. **Average proximity**: `avg_proximity_nm` $= \frac{2}{|\mathcal{F}(t)|(|\mathcal{F}(t)|-1)} \sum_{i<j} d_{ij}(t)$, the mean pairwise great circle distance in nautical miles.

3. **Altitude mix**: `altitude_mix_ft` $= \sqrt{\frac{1}{|\mathcal{F}(t)|} \sum_{f \in \mathcal{F}(t)} \big(h_f(t) - \bar{h}(t)\big)^2}$, the population standard deviation of altitudes.

4. **Conflict risk**: `conflict_risk` $= \frac{1}{|\mathcal{F}(t)|} \sum_{f \in \mathcal{F}(t)} \min_{g \neq f} d_{fg}(t)$, the mean per aircraft minimum separation distance.

5. **Primary flow direction**: `primary_flow_direction` $= \frac{1}{|\mathcal{F}(t)|} \sum_{f \in \mathcal{F}(t)} |\sin(\psi_f(t))|$, where ψ_f is heading. Values near 0 indicate predominantly N to S flow; values near 1 indicate E to W flow.

6. **Flow concentration**: `flow_concentration` $= \sigma_{\text{circ}} = \sqrt{-2 \ln \bar{R}}$, where $\bar{R} = \left| \frac{1}{|\mathcal{F}(t)|} \sum_f e^{i\psi_f(t)} \right|$ is the mean resultant length of headings. Low values indicate uniform (unidirectional) flow; high values indicate dispersed headings.

7. **Time of day**: `time_of_day_hour` $\in [0, 23]$, the hour of the scenario.

8. **Airspace size**: `airspace_size_nm` $= 100\,\text{NM}$, the study region radius (constant across the baseline dataset).

3.6.2 Engineered features (9)

1. **Congestion index**: $\texttt{congestion\_index} = \texttt{traffic\_density}/(\texttt{avg\_proximity\_nm}+\epsilon)$, where $\epsilon = 10^{-6}$ prevents division by zero. Higher values indicate dense, tightly packed traffic.

2. **Traffic altitude complexity**:

$$\texttt{traffic\_altitude\_complexity} = \texttt{traffic\_density} \times \texttt{altitude\_mix\_ft},$$

   capturing the interaction between traffic count and vertical spread.

3. **Hotspot indicator**: $\texttt{hotspot\_indicator} = \texttt{conflict\_risk}/(\texttt{avg\_proximity\_nm}+\epsilon)$, a dimensionless ratio indicating how close aircraft are relative to the overall airspace spread.

4. **Traffic density squared**: $\texttt{traffic\_density\_sq} = \texttt{traffic\_density}^2$, quadratic density term for nonlinear scaling effects.

5. **Log average proximity**: $\texttt{log\_avg\_proximity} = \ln(1+\texttt{avg\_proximity\_nm})$, log transformed proximity for diminishing returns at large distances.

6. **Traffic level**: $\texttt{traffic\_level} \in \{0, 1, 2\}$ (low/medium/high), binned traffic density with thresholds at 5 and 15 aircraft.

7. **Proximity level**: $\texttt{proximity\_level} \in \{0, 1, 2\}$ (close/medium/far), binned average proximity with thresholds at 20 and 40 NM.

8. **Conflict risk normalized**: $\texttt{conflict\_risk\_normalized} = \texttt{conflict\_risk}/(\texttt{traffic\_density}+\epsilon)$, per aircraft conflict risk normalized by traffic count.

9. **Flow directionality**: $\texttt{flow\_directionality} = |\texttt{primary\_flow\_direction} - 0.5| \times 2$, measures how strongly traffic aligns with either N to S or E to W axes. Values near 1 indicate strong axial alignment; values near 0 indicate diagonal or mixed flow. This feature is important for distinguishing configurations like $N \times M$ from $M \times N$.

3.6.3 Time and day features (6)

Rather than encoding specific locations or detailed time periods, we use six binary features that capture operationally relevant temporal patterns:

- **Day type** (2 features): `day_weekday`, `day_weekend`
- **Time of day** (4 features): `time_morning` (6–11h), `time_noon` (11–14h), `time_evening` (14–20h), `time_night` (20–6h)

Total feature count. The full feature vector $\mathbf{z} \in \mathbb{R}^d$ has $d = 23$ dimensions (8 raw + 9 engineered + 6 time/day). The model is explicitly location agnostic: no geographic identifiers are included, ensuring learned patterns transfer across airspace regions.

Table 3.1: Complete feature list (23 features, location agnostic).

Feature	**Description / Formula**	**Range**
Raw airspace features (8)		
`traffic_density`	Aircraft count $\lvert\mathcal{F}(t)\rvert$	$[0, \sim 150]$
`avg_proximity_nm`	Mean pairwise great circle dist.	$[0, \sim 80]$ NM
`altitude_mix_ft`	Std. dev. of altitudes	$[0, \sim 15{,}000]$ ft
`conflict_risk`	Mean min. separation per aircraft	$[0, \sim 60]$ NM
`primary_flow_dir`	Mean $\lvert\sin(\psi)\rvert$	$[0, 1]$
`flow_concentration`	Circular std. dev. σ_{circ}	$[0, \sim 1.5]$
`time_of_day_hour`	Scenario hour	$[0, 23]$
`airspace_size_nm`	Region radius	100 NM
Engineered features (9)		
`congestion_index`	density / (proximity + ϵ)	$[0, \sim 10]$
`traffic_alt_complexity`	density $\times$ altitude_mix	$[0, \sim 10^6]$
`hotspot_indicator`	conflict_risk / (proximity + ϵ)	$[0, \sim 4]$
`traffic_density_sq`	density^2	$[0, \sim 22{,}500]$
`log_avg_proximity`	$\ln(1 + \text{proximity})$	$[0, \sim 4.4]$
`traffic_level`	Binned density (low/med/high)	$\{0, 1, 2\}$
`proximity_level`	Binned proximity (close/med/far)	$\{0, 1, 2\}$
`conflict_risk_norm`	conflict_risk / (density + ϵ)	$[0, \sim 60]$
`flow_directionality`	$\lvert\text{flow\_dir} - 0.5\rvert \times 2$	$[0, 1]$
Time/day features (6)		
`day_weekday`	Weekday indicator	$\{0, 1\}$
`day_weekend`	Weekend indicator	$\{0, 1\}$
`time_morning`	6–11h	$\{0, 1\}$
`time_noon`	11–14h	$\{0, 1\}$
`time_evening`	14–20h	$\{0, 1\}$
`time_night`	20–6h	$\{0, 1\}$

3.7 Model Architecture

We employ a two stage architecture based on XGBoost [18]. XGBoost is a gradient boosted decision tree framework that learns an additive ensemble of trees by iteratively fitting residuals using a second order Taylor expansion of the loss function and applying L_1/L_2 regularization to control model complexity. Its efficiency, built-in handling of missing values, and strong performance on tabular data make it well suited for our structured feature

space.

3.7.1 Two stage approach

We decompose prediction into two stages to handle the structural difference between 1×1 (single sector, no partitioning) and multi sector configurations:

Stage 1: Binary classifier. An `XGBClassifier` distinguishes 1×1 snapshots (label 0) from all non 1×1 snapshots (label 1). This stage achieves near perfect recall (99.29%) on non 1×1 samples, ensuring complex traffic scenarios are rarely misclassified as trivial.

Stage 2: Multi class classifier. An `XGBClassifier` with `objective=multi:softmax` predicts the exact configuration among the 24 non 1×1 classes. Unlike regression approaches that predict rows and columns independently, multi class classification treats each (R, C) combination as a distinct class, allowing the model to learn joint dependencies between row and column counts. The 24 non 1×1 classes cover all combinations from 1×2 to 5×5, including intermediate configurations such as 3×1, 3×2, 3×3, 3×4, 3×5, 1×3, 2×3, 4×3, and 5×3.

At inference, a sample classified as 1×1 by Stage 1 receives $(R, C) = (1, 1)$ directly; otherwise, Stage 2 predicts the full configuration.

3.7.2 Architectural justification

We evaluated a simpler single stage approach where a single 25 class `XGBClassifier` directly predicts the configuration without decomposition. Table 3.2 compares the two architectures on the same dataset (∼65,000 samples, ∼2,700 per class).

Table 3.2: Single stage vs. Two stage architecture comparison.

Metric	**Single Stage**	**Two Stage**
Overall Test Accuracy	85.12%	**91.38%**

The two stage architecture achieves approximately 6% higher overall accuracy. The improvement stems from task decomposition: Stage 1 specializes in distinguishing simple (single sector) from complex (multi sector) scenarios, while Stage 2 focuses exclusively on the

harder 24 class discrimination problem among multi sector configurations. This separation allows each classifier to learn more discriminative decision boundaries for its specific task.

Handling data imbalance. In operational airspace, the 1×1 configuration (no partitioning needed) dominates: our raw simulation data contained over 100,000 1×1 instances, sufficient to train a model by itself. We downsampled 1×1, 1×2, and 2×1 entries to achieve balanced representation. The two stage architecture naturally handles this imbalance: Stage 1 quickly identifies the dominant 1×1 case without requiring the multi class classifier to discriminate against an overwhelming majority class.

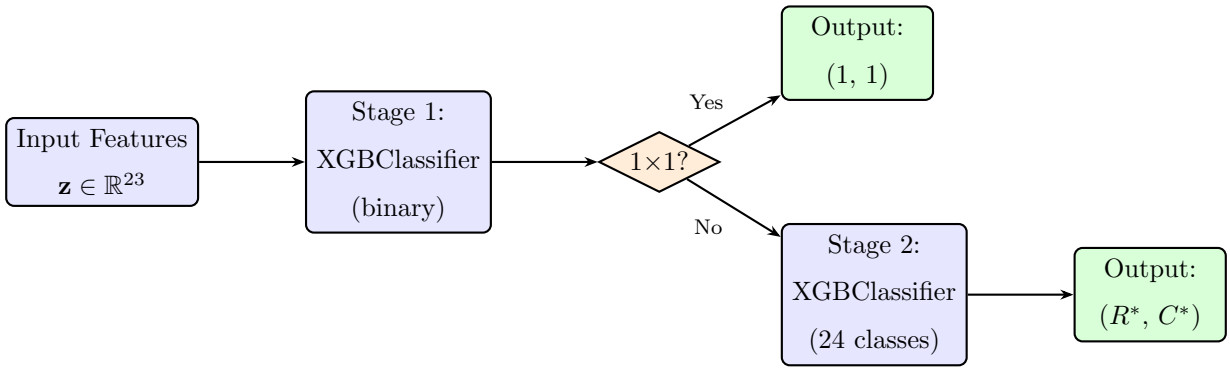


Fig. 3.2 Two stage XGBoost architecture. Stage 1 is a binary classifier that identifies 1×1 snapshots. Non 1×1 samples proceed to Stage 2, a multi class classifier that predicts the exact configuration among 24 classes.

Table 3.3: Two stage model hyperparameters.

Stage	Hyperparameter	Value
Stage 1 (Binary)	n_estimators	500
	max_depth	6
	learning_rate	0.05
	subsample	0.8
	objective	binary:logistic
Stage 2 (Multi class)	n_estimators	800
	max_depth	14
	learning_rate	0.02
	subsample	0.85
	colsample_bytree	0.85
	min_child_weight	1
	reg_alpha	0.1
	objective	multi:softmax

3.8 Training, Validation Protocol, and Calibration

3.8.1 Data split

The ∼65,000 samples are divided 70/15/15 into training, validation, and test sets (∼45,800 / ∼9,800 / ∼9,800), using stratified random sampling to preserve class proportions across splits.

3.8.2 Class balance through combined data collection

We address class imbalance through a combination of targeted simulation campaigns and physics informed synthetic augmentation:

1. **Extended simulation campaigns.** Additional simulation runs with high density traffic and complex flow patterns generate realistic samples for underrepresented configurations.

2. **Synthetic augmentation.** A class separable feature synthesis algorithm generates additional training samples for minority configurations, using physics informed feature distributions that preserve the statistical signatures of each configuration class.

3. **Balanced representation.** Most configurations receive approximately 2,700 samples, with 1×1 at 5,350 samples (reflecting its higher operational frequency) and 5×5 at 300 samples (reflecting its rarity), ensuring representation across all 25 configuration classes.

3.8.3 Training convergence

Figures 3.3 and 3.4 show the training curves for both stages. Both models converge smoothly without overfitting, with validation metrics closely tracking training metrics throughout training.

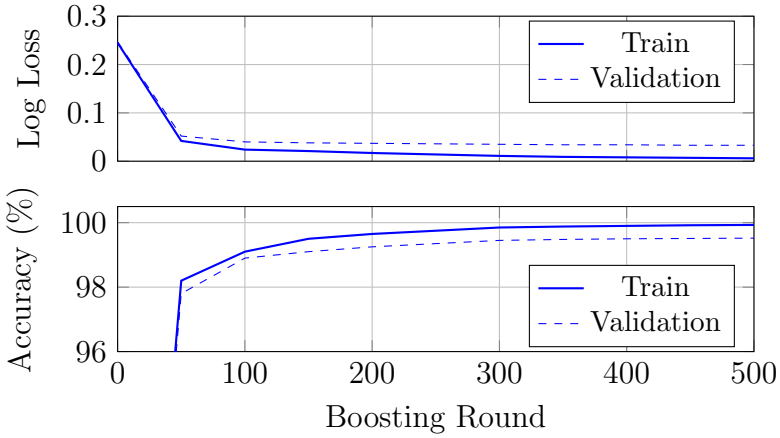


Fig. 3.3 Stage 1 (binary classifier) training curves over 500 boosting rounds. Top: log loss converges rapidly as the model learns to distinguish 1×1 from non 1×1 configurations. Bottom: accuracy reaches 99.93% on training and 99.52% on validation, with minimal overfitting gap.

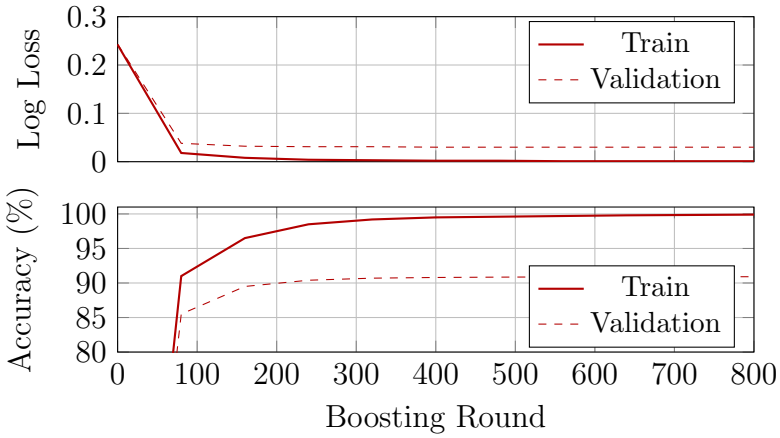


Fig. 3.4 Stage 2 (24 class multi class classifier) training curves over 800 boosting rounds. Top: log loss converges more slowly than Stage 1 due to the harder 24 class discrimination task. Bottom: training accuracy reaches 99.9% while validation plateaus at 90.91%, reflecting the difficulty of distinguishing similar configurations (e.g., 2×3 vs. 3×2).

3.8.4 Evaluation metrics

We report the following metrics on the held out test set:

- **Overall accuracy (%)**: Fraction of samples where the predicted (R, C) exactly matches the true configuration.
- **Stage 1 recall**: Fraction of true non 1×1 samples correctly identified as non 1×1 (critical for avoiding under sectorization).
- **Stage 1 precision**: Fraction of predicted non 1×1 samples that are truly non 1×1.
- **Per configuration accuracy**: Breakdown by each of the 25 configurations.

3.9 Results

3.9.1 Overall performance

Table 3.4: Two stage model: overall accuracy.

Metric	Value
Overall accuracy	91.38%
Stage 1 recall	99.29%
Stage 1 precision	99.68%

The two stage model achieves 91.38% overall accuracy on the held out test set. Stage 1 exhibits near perfect recall (99.29%), meaning the model almost never misclassifies a complex traffic scenario as trivial. This is operationally critical: under sectorization (predicting 1×1 when multiple sectors are needed) could lead to controller overload.

3.9.2 Stage analysis

We decompose Stage 2 performance to understand row and column prediction accuracy:

Table 3.5: Stage 2 multi class classifier performance breakdown.

Metric	Value
Rows correct	95.5%
Columns correct	94.8%
Both correct (non 1×1 accuracy)	90.91%

The multi class classifier achieves 95.5% accuracy on row prediction and 94.8% on column prediction when evaluated independently. The combined accuracy (90.91%) is lower because both dimensions must be predicted correctly, an error in either row or column results in an incorrect overall prediction.

3.9.3 Per configuration accuracy

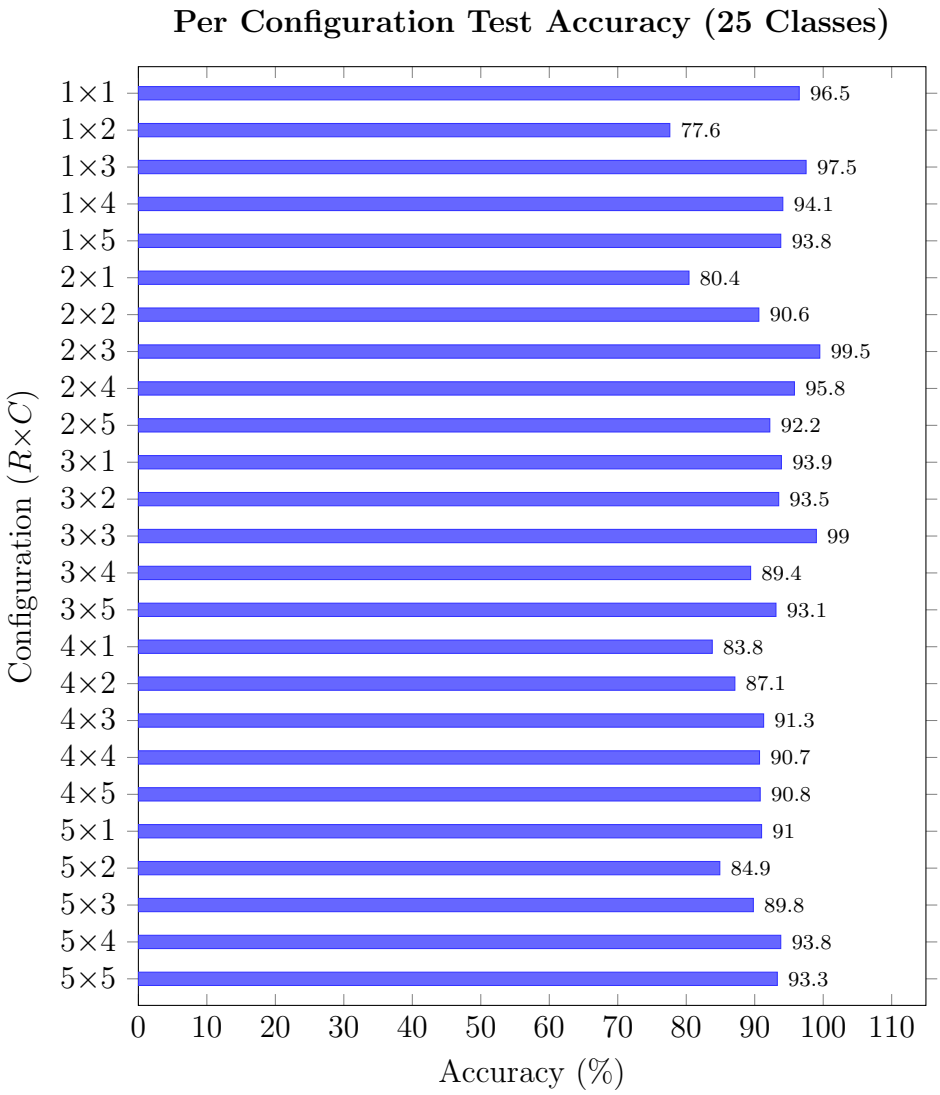


Fig. 3.5 Per configuration accuracy on the test set. All 25 configurations achieve 78–99% accuracy. Configurations with distinctive traffic patterns achieve highest accuracy.

3.9.4 Error analysis

Figure 3.6 shows the full 25×25 confusion matrix for the two stage model on the held out test set. The diagonal dominance confirms that the model predicts the correct configuration for the vast majority of samples. Off diagonal entries reveal where errors concentrate and expose systematic confusion patterns.

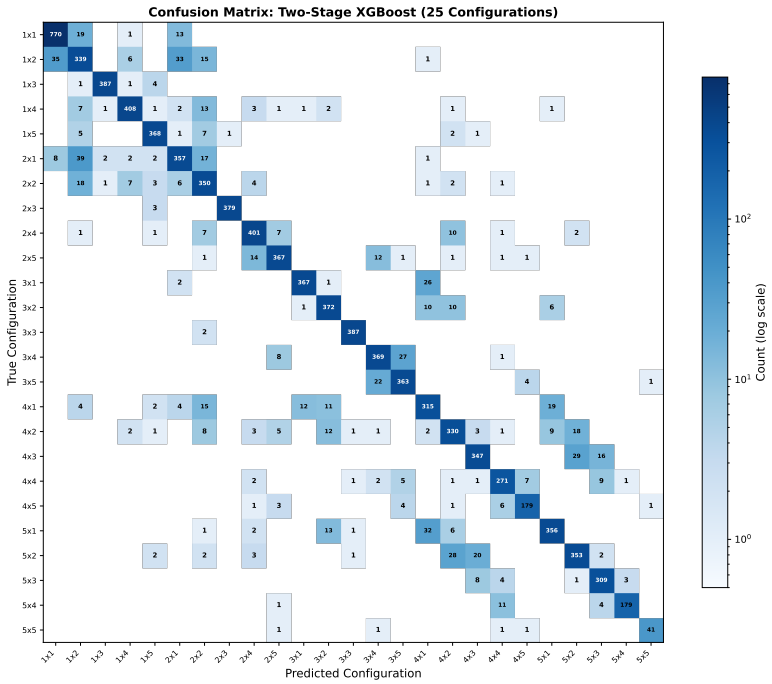


Fig. 3.6 Confusion matrix for the two stage model (25 configurations). Rows represent true labels, columns represent predictions. Cell values show sample counts on the test set. The strong diagonal confirms high accuracy across all configurations; off diagonal entries cluster among structurally similar configurations.

Three error patterns emerge from the confusion matrix:

1. **Low partition confusion.** The most common error pair is $1 \times 2 \leftrightarrow 2 \times 1$: the model confuses these two configurations because both represent a single bisection of the airspace, differing only in orientation (horizontal versus vertical). Similarly, 1×2 is sometimes predicted as 1×1 (35 cases), reflecting borderline traffic scenarios near the sectorization threshold.

2. **Row count adjacency errors.** Configurations that differ by one row are frequently confused: $5 \times 1 \rightarrow 4 \times 1$ (32 cases), $4 \times 3 \rightarrow 5 \times 2$ (29 cases), $5 \times 2 \rightarrow 4 \times 2$ (28 cases), and $3 \times 1 \rightarrow 4 \times 1$ (26 cases). These errors occur because adjacent row counts produce similar traffic density profiles, and the model must rely on secondary features (altitude mix, flow directionality) to discriminate.

3. **Column swap errors.** Configurations like $3 \times 4 \leftrightarrow 3 \times 5$ (27 and 22 cases respectively)

are confused when the model correctly identifies the row count but misassigns the column count by one. This reflects the inherent difficulty of distinguishing fine grained column partitions when traffic flow concentration varies smoothly.

Table 3.6 reports per class precision, recall, and F1 score. Configurations with distinctive traffic signatures (2×3, 3×3, 1×3) achieve F1 scores above 0.98. The weakest performers are 1×2 (F1 = 0.79) and 4×1 (F1 = 0.82), consistent with the confusion patterns described above.

Table 3.6: Per class precision, recall, and F1 score on the test set.

Config	**Prec.**	**Rec.**	**F1**	**Config**	**Prec.**	**Rec.**	**F1**
1×1	0.95	0.96	0.95	3×4	0.91	0.91	0.91
1×2	0.78	0.79	0.79	3×5	0.91	0.93	0.92
1×3	0.99	0.98	0.99	4×1	0.81	0.82	0.82
1×4	0.96	0.93	0.94	4×2	0.84	0.83	0.84
1×5	0.95	0.96	0.95	4×3	0.91	0.89	0.90
2×1	0.85	0.83	0.84	4×4	0.91	0.90	0.91
2×2	0.80	0.89	0.84	4×5	0.93	0.92	0.93
2×3	1.00	0.99	0.99	5×1	0.91	0.87	0.89
2×4	0.93	0.93	0.93	5×2	0.88	0.86	0.87
2×5	0.93	0.92	0.93	5×3	0.91	0.95	0.93
3×1	0.96	0.93	0.94	5×4	0.98	0.92	0.95
3×2	0.91	0.93	0.92	5×5	0.95	0.91	0.93
3×3	0.99	0.99	0.99				

3.9.5 Cross validation

To confirm that the reported accuracy is not a result of a single train/test split, we perform five fold stratified cross validation on the full two stage pipeline. Each fold replicates the complete training procedure: Stage 1 binary classification with threshold tuning, followed by Stage 2 multi class prediction for non 1×1 samples. Synthetic augmentation is applied

to the training portion of each fold.

Table 3.7: Five fold stratified cross validation results.

Metric	**Value**
Overall accuracy	$91.1 \pm 0.3\%$
1×1 accuracy	$95.4 \pm 0.7\%$
Non 1×1 accuracy	$90.8 \pm 0.3\%$

The narrow standard deviation ($\pm 0.3\%$) across five folds confirms that the 91.38% accuracy reported on the primary test set is representative and not an artifact of a fortunate split. The 1×1 classifier shows slightly higher variance ($\pm 0.7\%$), reflecting sensitivity to which borderline low traffic samples appear in each fold, but remains consistently above 94%.

3.9.6 Key findings

1. **Representative data enables reliable learning.** With ∼2,700 samples per configuration (1×1: 5,350, 5×5: 300), all 25 configurations achieve 78–99% accuracy. The distribution reflects realistic operational patterns while ensuring all classes are learnable.

2. **Configurations with distinctive patterns are easiest to predict.** Configurations like 2×3 (99.5%), 3×3 (99.0%), 1×3 (97.5%), and 2×4 (95.8%) achieve highest accuracy. These correspond to scenarios with distinctive feature signatures that are easily distinguishable from other configurations.

3. **Errors cluster among structurally similar configurations.** The confusion matrix reveals that misclassifications are not random but concentrate among configurations that differ by one row or column count, or that share the same total sector count but differ in orientation. This suggests the model captures the underlying traffic structure and errs only at the boundaries between operationally similar configurations.

4. **Stage 1 near perfect recall ensures safety.** With 99.29% recall on non 1×1 samples, the model almost never under sectorizes. The few false negatives (predicting

1×1 when partitioning is needed) occur in borderline low traffic scenarios where either configuration is operationally acceptable.

5. **Cross validation confirms stability.** Five fold stratified cross validation yields $91.1 \pm 0.3\%$ overall accuracy, confirming that the result is robust to the choice of train/test split and not dependent on a particular data partition.

3.10 Feature Importance and Ablations

We analyze feature importance using XGBoost's gain based metric, which measures the total reduction in loss attributable to splits on each feature across all trees in the ensemble. Since the model is location agnostic (no geographic identifiers in the feature set), all importance is attributed to traffic characteristics.

Table 3.8: Top 10 features by average gain across both stages.

Rank	Feature	Avg Gain
1	`traffic_level`	0.463
2	`traffic_density`	0.131
3	`traffic_density_sq`	0.125
4	`traffic_altitude_complexity`	0.052
5	`congestion_index`	0.038
6	`flow_directionality`	0.038
7	`primary_flow_direction`	0.035
8	`altitude_mix_ft`	0.015
9	`flow_concentration`	0.014
10	`day_weekend`	0.014

Key insights:

1. **Traffic level dominates.** The binned `traffic_level` feature (avg gain = 0.463) provides the strongest signal, followed by raw `traffic_density` and its squared term.

Together, these three features account for over 79% of total gain. This confirms that sectorization need is primarily driven by traffic volume.

2. **Flow directionality distinguishes asymmetric grids.** The `flow_directionality` and `primary_flow_direction` features help the model distinguish configurations like $N \times M$ from $M \times N$. When traffic flows predominantly E to W versus N to S, the optimal partitioning orientation differs accordingly.

3. **No location features needed.** The model achieves 91.38% accuracy using only traffic characteristics, with no geographic identifiers. This confirms that sectorization decisions depend on traffic state rather than airspace identity, supporting the model's applicability to unseen airspace regions.

3.11 Runtime Use: Forecasting Load for Configuration Screening

The trained XGBoost models serve as a fast screening layer in the configuration search pipeline of Chapter 5. Rather than evaluating all candidate grid configurations via full simulation (each requiring traffic replay and metric computation), the model provides instant point estimates of (R^*, C^*) that can seed or prune the Bayesian optimization search.

Feature scaling for arbitrary airspace sizes. The training data uses a fixed 100×100 NM study region. To apply the model to differently sized regions (including non square airspaces $L_x \times L_y$), we scale density and proximity features:

$$\texttt{traffic\_density}_{\text{scaled}} = \texttt{traffic\_density} \times \frac{10{,}000}{L_x \cdot L_y}, \tag{3.4}$$

$$\texttt{avg\_proximity}_{\text{scaled}} = \texttt{avg\_proximity\_nm} \times \frac{100}{\sqrt{L_x \cdot L_y}}, \tag{3.5}$$

where L_x and L_y are the airspace width and height in nautical miles. The area-based normalization preserves density semantics, while the linear scaling maintains proximity semantics relative to the reference airspace.

Why scaling outperforms alternatives. Alternative approaches, training separate models per airspace size or using spatial interpolation, require substantially more data and

computational resources. Our scaling approach offers key advantages: (1) a single model serves all airspace dimensions without retraining, (2) density and proximity semantics are preserved since the model learns traffic patterns rather than absolute coordinates, and (3) full simulation for configuration screening at each size would require orders of magnitude more compute. The normalization effectively projects any airspace onto the reference 100×100 NM training domain while maintaining the relative traffic complexity that determines optimal configuration.

Output scaling for target dimensions. Model predictions $(R_{\text{pred}}, C_{\text{pred}})$ for the 100×100 NM equivalent are scaled to the target airspace:

$$R_{\text{actual}} = \text{round}\left(R_{\text{pred}} \cdot \frac{L_y}{100}\right), \tag{3.6}$$

$$C_{\text{actual}} = \text{round}\left(C_{\text{pred}} \cdot \frac{L_x}{100}\right), \tag{3.7}$$

with constraints enforcing minimum sector dimensions (≥ 15 NM for adequate dwell time) and maximum configuration (5×5). Validation across 44 test cases spanning 30×30 NM to 200×200 NM (including non square airspaces) confirms that all outputs satisfy operational constraints.

Why maximum remains 5×5 for all airspace sizes. Although the scaling wrapper supports airspaces exceeding 100×100 NM (up to 200×200 NM in our validation), the maximum output configuration remains 5×5. This is not a model limitation but reflects operational reality: no traffic density in our simulation campaigns, even at the highest injection rates, produced scenarios requiring finer than 5×5 partitioning. Beyond this granularity, the handoff overhead outweighs workload balancing benefits. For a 200×200 NM airspace, a 5×5 grid yields 40×40 NM sectors, still well above the minimum viable size.

Inference performance. Each prediction requires <1 ms on a single CPU core (tree traversal only, no matrix operations). The serialized model occupies $\sim$1 MB, making it suitable for embedding in real time systems. For production deployment, the model can be exported via Treelite for compiled C++ inference or called as a Python subprocess.

Integration pathway. The XGBoost prediction feeds into the Bayesian optimization loop (Chapter 5) in two ways:

1. **Warm start initialization**: The predicted (R^*, C^*) provides an informed starting point for the acquisition function, reducing the number of exploration iterations needed to find the optimum.

2. **Candidate pruning**: Configurations far from the predicted optimum (e.g., $|R - \hat{R}| > 3$) can be excluded from the search space, reducing computational budget without sacrificing solution quality.

Why a learned model instead of exhaustive scoring. The current configuration space contains 25 horizontal grids, which is small enough to evaluate exhaustively at any single time step. A natural question is whether the XGBoost predictor adds value over computing the workload objective $J(R, C)$ for all 25 candidates directly. The answer is threefold. First, computing $J(R, C)$ requires the full aircraft state: every position, heading, speed, and pairwise conflict geometry. The model instead predicts from 23 aggregate traffic features that can be derived from a radar summary without running the scoring pipeline. Second, the approach is designed to scale. The 25 grid search space is intentionally small for this thesis; operational deployments would consider variable altitude layers, non-rectangular partitions, or finer grid resolutions, producing configuration spaces in the hundreds or thousands where brute force becomes impractical. Third, the 91.4% accuracy from 23 features with zero geographic identifiers is itself a scientific finding: it demonstrates that aggregate traffic patterns are sufficient statistics for the sectorization decision and identifies which features drive it (the top three account for 79% of total gain).

4 Decentralized Consensus for Aircraft Coordination

4.1 Chapter contributions

This chapter reimplements the DATC consensus protocol as a standalone library with a per aircraft engine architecture and adds reliability extensions including fuel aware planning and livelock prevention.

4.2 Overview

Once the XGBoost predictor from Chapter 3 recommends a sector grid (R^*, C^*), the airspace is partitioned and aircraft must coordinate their entries, exits, and conflict resolution within the airspace. This chapter describes the consensus protocol that makes that coordination possible without requiring a human controller in each sector.

The consensus protocol builds on the DATC framework [19], [20] and was originally implemented in OMNeT++. This thesis reimplements the protocol as a standalone C++ library with Python bindings, introduces a per aircraft engine architecture to replace OMNeT++'s shared state model, and adds several extensions including fuel aware planning, random backoff for livelock prevention, and bounded retry safety valves for reliability in the per aircraft setting. We describe the protocol in full, discuss the engineering challenges that arose from the architectural shift, and present the mechanisms developed to address them.

Role in the pipeline. This chapter sits between the grid prediction and parameter tuning stages. The XGBoost model from Chapter 3 decides how to partition the airspace. The consensus protocol described here takes that partition and lets aircraft coordinate sector transitions among themselves. The Bayesian Optimization in Chapter 5 then tunes this protocol's parameters for each airport.

4.3 Background and Protocol Provenance

Why consensus for airspace coordination. Sector admission, the decision to allow an aircraft into a sector under a specific flight plan, is fundamentally a distributed agreement

problem. Multiple aircraft sharing a sector must agree on who is present, what plans are active, and whether a new entrant's trajectory is conflict free with respect to everyone else. Today's system distributes authority across many controllers and facilities, but within each sector one controller is the sole authority for separation and sequencing. That per sector single human dependency is the bottleneck: when that controller is overloaded, fatigued, or unavailable, the sector is directly impacted. Aircraft to aircraft consensus replaces this per sector human dependency with automated agreement, allowing coordination to scale with traffic density rather than controller staffing.

Paxos and the Synod protocol. Lamport's seminal work [21] describes a family of consensus protocols. The single decree core of that family is the *Synod* protocol: a leaderless algorithm in which any participant may propose a value, and the group agrees on exactly one through a prepare–promise–accept–vote exchange. This is distinct from *Multi-Paxos*, a common optimization that elects a stable leader to reduce message complexity. A stable leader is a single point of failure, which is unacceptable for airborne systems where any aircraft may need to initiate a sector entry at any time. The DATC protocol therefore uses Synod: every entering aircraft acts as its own proposer, and the sector occupants serve as acceptors. Synod guarantees two safety properties that map directly to airspace needs: (i) *agreement*, all correct participants decide the same value (the same flight plan set φ), and (ii) *validity*, the decided value was actually proposed by some participant. The protocol uses a majority quorum of $Q = \lfloor |C_2|/2 \rfloor + 1$ acceptors, where C_2 is the set of aircraft currently inside the sector. Paul et al. [22] provide a machine checked proof of eventual consensus in the Synod protocol under a failure aware actor model, establishing the formal foundations on which the DATC protocol builds.

Timing independence of safety. A key property of Paxos is that its safety guarantees, agreement and validity, hold under arbitrary message delays, reordering, and loss [21]. No timing assumption is required for correctness; a conflict free flight plan that has been decided will never be contradicted, regardless of how long messages take to arrive. Only *liveness* (the guarantee that a decision is eventually reached) depends on timing: progress requires a majority of sector occupants to respond within the configured timeout window. When liveness fails, the entering aircraft times out, retries, and ultimately falls back to denying

entry, which is a safe outcome: the aircraft does not enter the sector and would, in a real deployment, divert to an alternate route or airport. This denial does not break the formal verification guarantees of the protocol, since the aircraft is removed from the system rather than admitted with a conflicting plan. The simulation presented in this chapter uses fixed, uniform communication delays as a simplification. Under variable or lossy communication, the safety guarantees are preserved; only throughput and convergence speed would degrade.

The DATC protocol. The Decentralized Air Traffic Control (DATC) protocol [19] adapts Paxos into a three phase transaction for each sector admission: (A) discovery, where the entering aircraft learns who occupies the sector, (B) a Paxos SYNOD round where the sector occupants reach consensus on a conflict free flight plan for the entrant, and (C) a Two Phase Acknowledge Protocol (TAP) that disseminates the decided plan to every relevant aircraft so all local copies of φ remain consistent. The conflict aware flight planning approach underlying the protocol was introduced by Paul et al. [15], who developed methods for avoiding near mid air collisions through coordinated trajectory negotiation, and later extended with formal verification of timely knowledge propagation in airborne networks [23]. The same three phase structure handles exits (removing an aircraft's plan from φ) and emergency departures for aircraft that cannot find a conflict free plan. Sections 4.5 through 4.9 describe each component in detail.

Our contributions. We reimplement the DATC protocol's C++ with pybind11 Python bindings, replacing OMNeT++'s shared state simulation with a more realistic per aircraft engine architecture. This architectural shift exposed several liveness problems, including Paxos livelock under multi proposer contention, timeout explosion from stale events, and infinite retries when departed aircraft leave "ghost" entries in sector state. Each required a targeted fix and together they form the reliability layer described in §4.7. We also add fuel aware planning, holding pattern geometry, and a tunable parameter interface that feeds into the Bayesian optimization loop of Chapter 5.

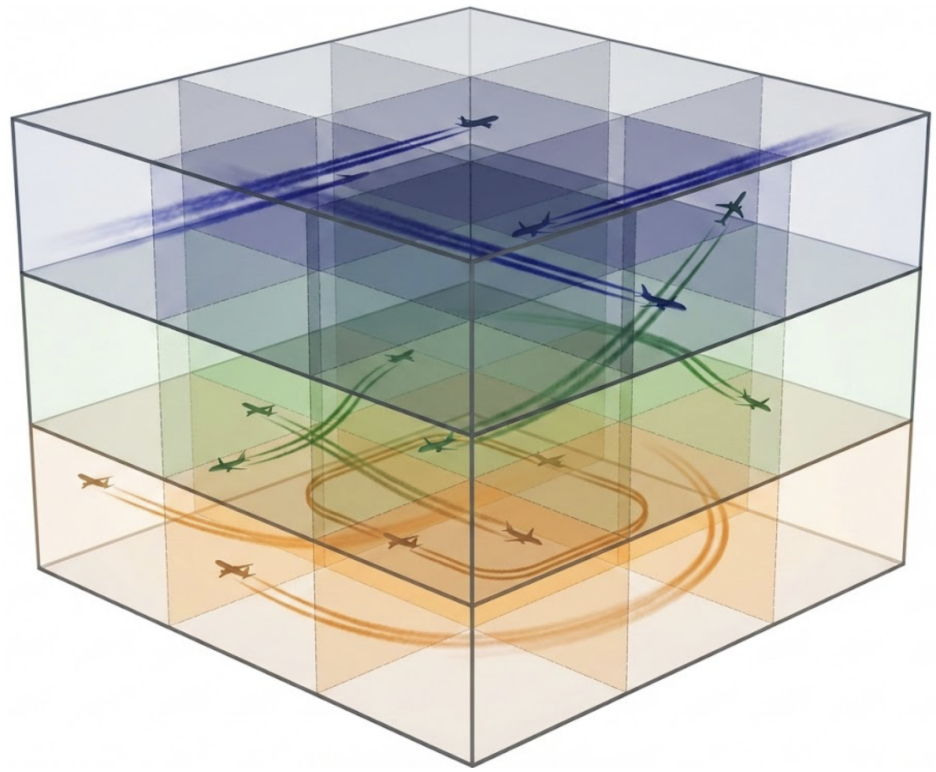


Fig. 4.1 Sectorized airspace volume. A $3 \times 3 \times 3$ grid of sectors across three altitude layers: terminal (orange), transition (green), and cruise (blue). Aircraft traverse sectors along planned trajectories; a racetrack holding pattern is visible in the terminal layer (not to scale) when the speed solver cannot resolve conflicts (§4.6.3).

4.4 System Model and Architecture

4.4.1 From shared state simulation to per aircraft engines

The original implementation lives inside OMNeT++, a C++ discrete event network simulator. In that environment, every aircraft is an OMNeT++ module connected by channels. A global Future Event Set (FES) orders all events across all modules, and modules can read each other's published state. When aircraft A modifies a sector's occupant set, aircraft B sees the change at the very next event. Messages are placed on the FES via `sendCommand2()` and delivered with causal ordering guaranteed by the simulator.

This thesis replaces that architecture with a per aircraft design:

1. Each aircraft (and each phantom representing an existing occupant) gets its own `ConsensusEngineReal` instance, a standalone C++ object with its own event queue,

sector map, and protocol state.

2. A Python orchestration layer (`ConsensusManager`) creates engines, routes messages between them, and enforces global time ordering by always processing the engine whose next event has the earliest timestamp. This global ordering is a simulation convenience, not a protocol requirement; in a real deployment, aircraft would exchange messages over air to air datalinks with no shared clock (see §6 for further discussion).

3. Existing sector occupants that an entering aircraft needs to coordinate with are represented as *phantom engines*, lightweight engine instances bootstrapped directly into the sector's C_2 set without going through consensus.

The per aircraft design has two advantages over the OMNeT++ approach: it removes the dependency on a specific simulator framework (the C++ library compiles standalone), and it exposes a clean Python API for integration with the rest of the airspace simulation pipeline. The cost is that engines are now *isolated*: when aircraft X completes an exit on engine X, engines A, B, and C do not automatically learn about it. This isolation introduces several liveness challenges discussed in §4.7.

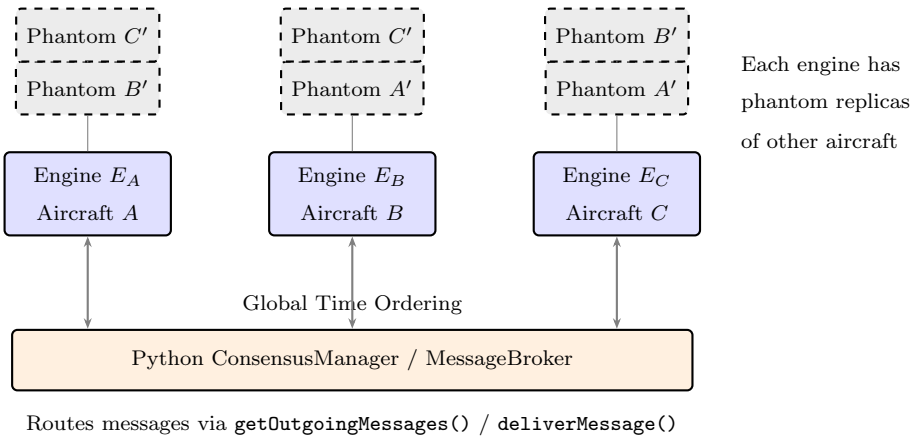


Fig. 4.2 Per aircraft engine architecture. Each aircraft runs its own `ConsensusEngineReal` instance with phantom replicas of other sector occupants. The Python `ConsensusManager` routes messages between engines and enforces global time ordering by processing the earliest pending event across all engines.

4.4.2 Sector state representation

Each engine maintains a `SectorState` struct per sector it participates in. The key fields are:

- **Aircraft sets.** Four overlapping sets track participation:
  - C_1: aircraft admitted by consensus but not yet physically inside the sector
  - C_2: aircraft currently inside the sector (the *acceptors* in Paxos terms)
  - C_3 (expectedAircraft): aircraft expected to arrive (announced via INIT_REQ)
  - S (relevantAircraft): the union $C_1 \cup C_2 \cup C_3$, i.e., everyone who participates in consensus for this sector
- **Admitted plans φ.** The ordered set of conflict free flight plans currently active in the sector. Every consensus round either adds a plan (entry) or removes one (exit). Safety is defined as: $\forall\, F_i, F_j \in \varphi,\ i \neq j$, the pair (F_i, F_j) satisfies the separation standards from §2.4.
- **Flight plan representation.** A flight plan is a tuple $(a, (t_0, W))$ where a is the aircraft ID, t_0 is the start time, and $W = [\mathbf{w}_1, \ldots, \mathbf{w}_n]$ is a sequence of waypoints, each $\mathbf{w}_k = (x, y, z, v, \psi, v_z)$ specifying position (meters), ground speed (m/s), heading (radians), and vertical rate (m/s). Aircraft fly piecewise linearly between consecutive waypoints.
- **Event counter.** A monotonically increasing integer incremented after each completed TAP round. Messages carry the sender's event counter; a receiver with a higher counter knows the message references an outdated φ and responds with a staleness notification (§4.7.3). Because the counter only increments after a TAP round completes and every aircraft in the relevant set has acknowledged, an aircraft in the sector is guaranteed to be at most one epoch behind the current state: if the sector is at epoch k, then TAP for epoch $k-1$ must have completed, meaning every aircraft received and acknowledged that round's decided value.

4.4.3 Deterministic event ordering

Consensus correctness depends on deterministic event processing, the same inputs must always produce the same protocol execution. We enforce this at three levels:

1. **Integer time.** All timestamps are stored as `uint64_t` picoseconds (10^{12} per second). This eliminates floating point comparison issues ("is $t_a < t_b$?" becomes exact integer comparison) and provides a resolution of 10^{-12} s, far below any meaningful protocol timescale. The maximum representable time is $\sim$213 days, sufficient for any simulation.

2. **Deterministic tie breaking.** The event queue is a `std::priority_queue` ordered by the tuple (time, aircraft_id, event_type). When two events share the same timestamp, the lower aircraft ID is processed first; if both match, lexicographic event type ordering breaks the tie. This guarantees identical execution order across runs.

3. **Unique message IDs.** Each engine's message counter starts at `aircraft_id` $\times$ 100,000, preventing collisions between engines in the global deduplication sets.

At the Python/C++ boundary, the pybind11 bindings convert seconds (Python `float`) to picoseconds (`uint64_t`) on entry and back on exit, so all arithmetic inside C++ is purely integer.

4.5 The DATC Protocol: Three Phases

Every sector admission (or exit) runs through three sequential phases: discovery, consensus, and dissemination. Figure 4.3 shows the high level flow.

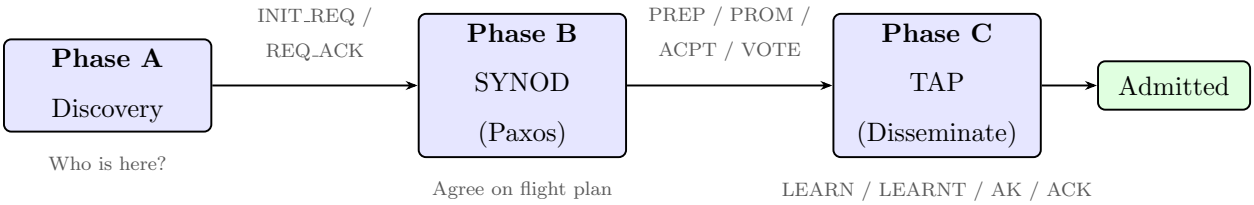


Fig. 4.3 DATC protocol overview. Each sector admission proceeds through discovery (find occupants), SYNOD (Paxos consensus on a conflict free plan), and TAP (reliable dissemination to all relevant aircraft).

4.5.1 Phase A: Discovery (INIT_REQ / REQ_ACK)

An aircraft wishing to enter sector s broadcasts an INIT_REQ message to all known aircraft. Every aircraft that is currently in C_1 or C_2 for sector s responds with a REQ_ACK containing a *frozen snapshot* of its local sector state: φ, C_1, C_2, S, the current event counter, and the highest proposal number it has seen (`maxPrepare`).

The word "frozen" is important. Each REQ_ACK captures the responder's state at send time, packaged into a typed message struct. In the original OMNeT++ implementation, this happened implicitly because all modules shared memory; in the per aircraft architecture, we had to design explicit message types (12 structs in `consensus_messages.h`) with snapshot fields to prevent handlers from reading stale or mutated state.

The proposer merges all received REQ_ACKs: it unions the aircraft sets, adopts the highest event counter, and records the maximum `maxPrepare` value (used to select a winning proposal number in the next phase). If fewer than the expected number of REQ_ACKs arrive within the `timeoutDuration` window, the proposer retries up to `numIRAttempts` times. If all retries are exhausted, the entry attempt fails.

4.5.2 Phase B: SYNOD consensus

The SYNOD phase implements a single round of the Synod protocol [21] adapted for sector admission. The entering aircraft acts as the *proposer*; the aircraft in C_2 act as *acceptors*. The protocol proceeds in four steps:

Step 1: Prepare. The proposer selects a proposal number n guaranteed to be higher than any previously seen. Specifically, it calls `findPropNum(flightID)`, which sets $n \leftarrow$ `flightID` and then increments by 10,000 until $n >$ `maxProp`:

$$n \leftarrow \texttt{flightID}; \quad \text{while } n \leq n_{\max}: \quad n \leftarrow n + 10{,}000 \tag{4.1}$$

It then sends a SYNOD_PREP message containing n and its proposed flight plan to every acceptor in C_2.

Step 2: Promise or NACK. Each acceptor receiving SYNOD_PREP with proposal

number n checks against its local `maxPrepare`:

- If $n >$ `maxPrepare`: the acceptor *promises* not to accept any proposal numbered below n. It sets `maxPrepare` $\leftarrow n$ and sends a SYNOD_PROM back to the proposer, including any previously accepted value (the pair (`accVal`, `accNum`)) and its pre generated alternate flight plans.

- If $n \leq$ `maxPrepare`: the acceptor sends a SYNOD_NACK containing its current `maxPrepare`, so the proposer can retry with a sufficiently high number.

Step 3: Accept. The proposer waits for a *quorum* of promises:

$$Q = \left\lfloor \frac{|C_2|}{2} \right\rfloor + 1 \tag{4.2}$$

Once Q promises arrive, the proposer selects the value to propose:

- If all promises carry empty accepted values: the proposer uses its own flight plan and runs the conflict resolution solver (`solve()`, §4.6.2) against the current φ to find a conflict free assignment. If the solver fails, alternates (holding patterns, §4.6.3) are tried in order.

- If any promise carries a non empty accepted value: the proposer must adopt the value with the highest `accNum` (the Paxos safety invariant that prevents overwriting a previously decided value).

The chosen value and its alternates are sent to all C_2 members in a SYNOD_ACPT message.

Step 4: Vote. Each acceptor receiving SYNOD_ACPT checks the proposed plan against its local φ using the conflict detection algorithm from §2.5.2. If safe, the acceptor sets `accVal` $\leftarrow$ `proposalValue`, `accNum` $\leftarrow$ `proposalNumber`, and sends a SYNOD_VOTE back to the proposer. Once the proposer collects a quorum of votes, the value is *decided* and the protocol moves to Phase C.

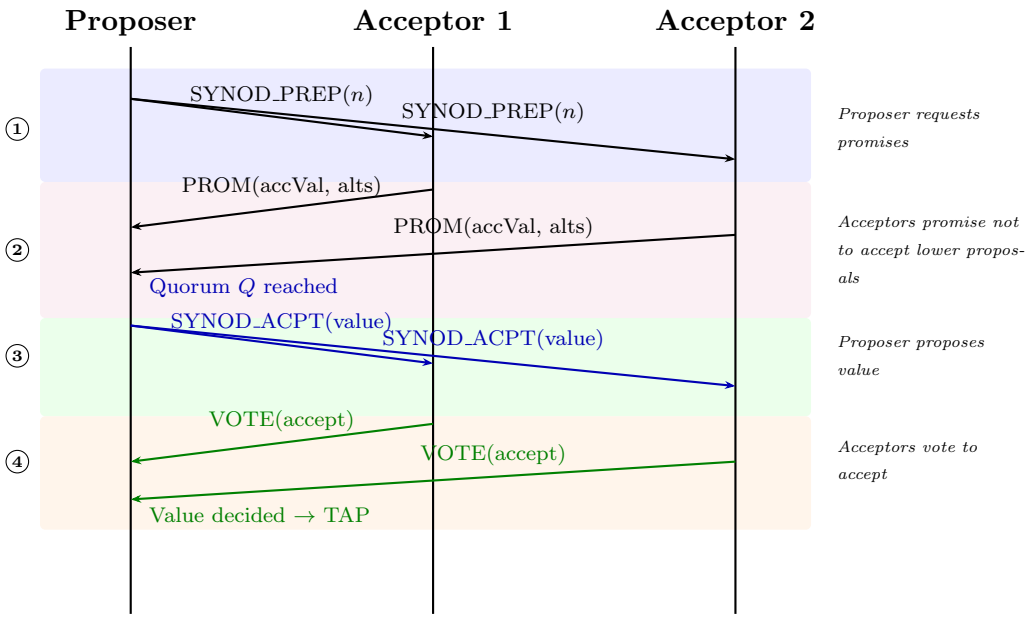


Fig. 4.4 SYNOD message sequence. The proposer sends PREP to all acceptors, collects a quorum of promises, sends ACPT with the chosen value, and collects votes. On quorum of votes, the value is decided and TAP begins.

Algorithm 3 SYNOD Consensus Round

procedure SYNODROUND(aircraft a, sector s, plan F_a)
  $n \leftarrow$ FINDPROPNUM(a.id) ▷ $n > n_{\max}$, increment by 10,000
  Broadcast SYNOD_PREP(n, F_a) to all $c \in C_2(s)$
  Wait for quorum $Q = \lfloor |C_2|/2 \rfloor + 1$ responses:
  if all promises have empty `accVal` **then**
    $M \leftarrow$ SOLVE$(F_a, \varphi(s))$ ▷ Backtracking speed search
    if M is complete **then**
      $v \leftarrow$ PLAN(M, F_a) ▷ Build modified plan from M
    else
      $v \leftarrow$ TRYALTERNATES(alts$(s), \varphi(s)$) ▷ Holding patterns
    end if
  else
    $v \leftarrow$ promise with highest `accNum` ▷ Paxos safety
  end if
  Broadcast SYNOD_ACPT(n, v) to all $c \in C_2(s)$
  Collect votes; on quorum → proceed to TAP
end procedure

4.5.3 Phase C: TAP (Two Phase Acknowledge Protocol)

Once the SYNOD decides a value, every aircraft in the sector's relevant set S must learn the new φ, not just the quorum that voted. The TAP, developed by Paul et al. [20], [23], ensures reliable dissemination through a four message handshake:

1. **TAP_LEARN.** The coordinator (the proposer that won the SYNOD) sends the decided φ and the accepted value to all aircraft in S. The message carries frozen snapshots of C_1, C_2, S, and the coordinator set.

2. **TAP_LEARNT.** Each recipient updates its local φ, resets its `accVal` to empty (critical: this prevents stale accepted values from propagating to future SYNOD rounds), and responds to the coordinator.

3. **TAP_AK (All Know).** Once the coordinator receives TAP_LEARNT from all targets in S, it sends an All Know signal confirming that every participant has the updated φ.

4. **TAP_ACK.** Recipients acknowledge the All Know. The coordinator collects all ACKs, releases the *learning lock*, and increments the event counter. The admission is now complete.

Why TAP is necessary. A natural question is why not skip TAP and simply run another SYNOD round to spread the result. Without TAP, aircraft that did not vote in the round that just finished would still hold an outdated φ. If a new proposal arrives immediately, those aircraft would check it against an old set of admitted plans and might approve a proposal that conflicts with the admission they have not yet learned about.

Quorum intersection guarantees that at least one voter in any new round will have the current state, so safety is never violated. The conflict would be caught and the round would fail with a NACK. But this means wasted rounds, more retries, and lower throughput.

TAP eliminates this by ensuring every aircraft in the sector has the latest φ before any new round begins. After TAP completes, no aircraft is even one epoch behind. The next round succeeds on the first attempt instead of failing due to stale state. TAP is therefore not a safety mechanism. It is an efficiency mechanism.

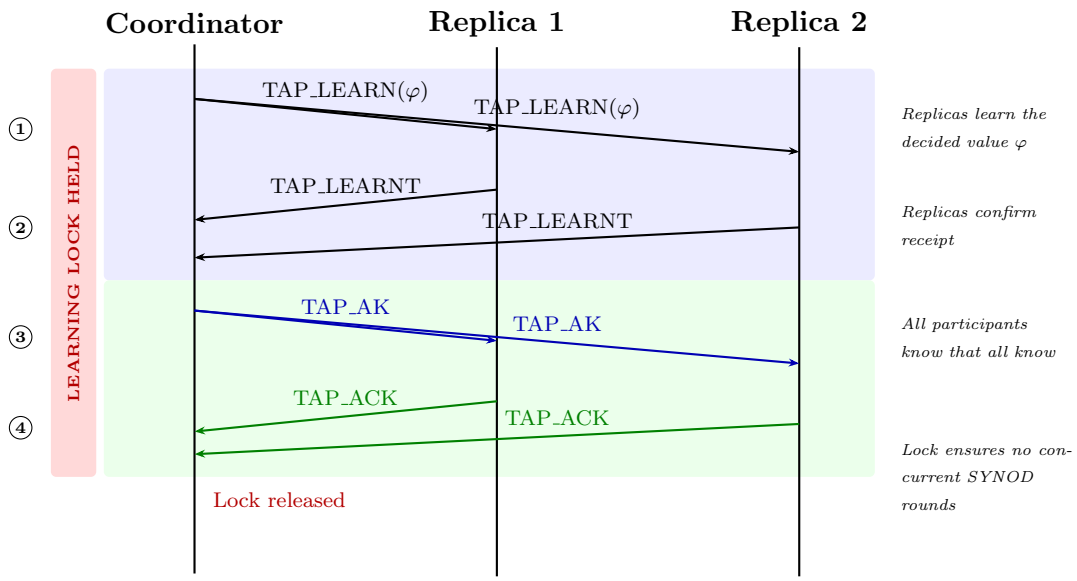


Fig. 4.5 TAP dissemination sequence. The coordinator sends the decided φ to all replicas, collects acknowledgments, sends All Know, and releases the learning lock after all ACKs. The lock prevents concurrent SYNOD rounds from corrupting φ during dissemination.

Learning locks. A learning lock is a set of (aircraft_id, is_exit) pairs that prevents concurrent consensus rounds from running in the same sector while a TAP is in progress. The lock is acquired when the coordinator begins TAP_LEARN and released only after all TAP_ACK messages are received. If a new SYNOD_PREP arrives while a learning lock is held, the event is deferred rather than processed, ensuring φ is never modified by two concurrent rounds.

4.6 Conflict Detection and Resolution

4.6.1 Pairwise conflict detection

The SYNOD phase must verify that a proposed flight plan does not violate separation standards with any existing plan in φ. We use the same piecewise linear conflict detection algorithm from §2.5.2, adapted from Paul et al.'s formally verified approach [15], [23]. Given two flight plans F_a and F_b:

1. Merge all waypoint timestamps from both plans into a sorted vector $\mathcal{T} = \textsc{getAllTimes}(F_a, F_b)$.

2. For each consecutive pair $(t_i, t_{i+1}) \in \mathcal{T}$, interpolate both aircraft positions and check:

$$\exists\, t \in (t_i, t_{i+1}] :\ d_{xy}(t) < D_m \ \wedge\ |\Delta z(t)| < H_m \tag{4.3}$$

where $D_m = 9{,}260\,\text{m}$ (5 NM) and $H_m = 304.8\,\text{m}$ (1,000 ft).

3. If any violation is found, the pair is in conflict. The check is *conservative*: no reported violation guarantees separation under the piecewise linear motion model.

4.6.2 Speed assignment via backtracking search

When a proposed plan conflicts with φ, the solver applies Paul's formally verified backtracking algorithm [15] to eliminate violations by modifying ground speeds on individual flight segments. Let the plan have n waypoint segments and let $\Xi = \{\xi_1, \ldots, \xi_m\}$ be the set of allowable speeds (default: $[240, 200, 180, 160, 140, 120]\,\text{m/s}$, corresponding to approximately 466–233 knots). The solver maintains an assignment matrix $M \in \{0,1\}^{n \times m}$ where $M[i,j] = 1$ means segment i is assigned speed ξ_j.

The algorithm performs depth first backtracking: at each unassigned segment, it tries each speed in Ξ, sets $M[i,j] \leftarrow 1$, builds the resulting partial plan, and checks safety against φ. If safe, it recurses to the next segment. If the partial plan already violates separation, it backtracks. When a speed is changed, the vertical rate is adjusted to preserve the flight path angle:

$$v_z' = v_g' \times \frac{v_z}{v_g} \tag{4.4}$$

If a complete, safe assignment is found, the solver returns the modified plan (counted as a "speed modification" in the statistics). If no valid assignment exists, the solver returns failure and the protocol falls back to alternates. Algorithm 4 summarizes the full conflict resolution procedure.

Algorithm 4 Conflict Resolution

Require: Flight plan F, admitted plans Φ, alternates $\mathcal{A} = [A_1, \ldots, A_k]$

Ensure: Conflict-free plan F^* or DENIED_ENTRY

$M \leftarrow \text{SOLVE}(F, \Phi)$ ▷ Backtracking speed search on base plan
if M is a complete assignment **then**
    return APPLYASSIGNMENT(M, F) ▷ Speed modification
end if
for each alternate $A_i \in \mathcal{A}$ **do** ▷ Holding patterns
    $M \leftarrow \text{SOLVE}(A_i, \Phi)$
    if M is a complete assignment **then**
        return APPLYASSIGNMENT(M, A_i)
    end if
end for
return DENIED_ENTRY ▷ No conflict-free plan exists; initiate C3 exit

4.6.3 Alternate flight plans (holding patterns)

When the speed solver cannot resolve a conflict, the protocol uses pre generated alternate flight plans. These are generated once, at sector entry, by `calcAlternates()` and attached to all subsequent SYNOD messages.

Each alternate is a *racetrack shaped* holding pattern defined by two holding waypoints ($\mathbf{h}_1$, $\mathbf{h}_2$), a holding altitude z_h, and a turn radius r. The racetrack consists of two straight legs (inbound: $\mathbf{h}_1 \to \mathbf{h}_2$; outbound: $\mathbf{h}_2' \to \mathbf{h}_1'$) connected by two semicircular turns. The outbound leg is offset perpendicular to the inbound leg by $2r$, so $\mathbf{h}_1' = \mathbf{h}_1 + 2r\,\hat{\mathbf{n}}$ and $\mathbf{h}_2' = \mathbf{h}_2 + 2r\,\hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is the unit normal to the inbound leg. The aircraft transits from its current position to $\mathbf{h}_1$, orbits the racetrack for L laps (up to $L_{\max} = 5$), and then returns to its original position.

Waypoint discretization. Each semicircular turn is discretized into 6 equally spaced arc points at angular increments of $\pi/7$ rad (≈ 25.7), yielding a total of **14 waypoints per lap**: 1 inbound straight + 6 arc points (turn at $\mathbf{h}_2$) + 1 outbound straight + 6 arc points (turn

at $\mathbf{h}_1$). Including the transit legs, the full holding plan contains $1 + 14L + 1$ waypoints. The choice of 6 arc points per semicircle balances geometric fidelity against computational cost: the conflict solver `solve()` performs pairwise safety checks across all waypoint segments of all aircraft in the sector, with cost quadratic in the total number of segments. At 5 laps, a single holding aircraft contributes $14 \times 5 = 70$ waypoints. Fewer arc points (e.g., 4) introduce large chord errors where the straight line approximation deviates significantly from the true arc; more points (e.g., 8 to 10) yield diminishing improvements in approximation accuracy while substantially increasing `solve()` running time.

Turn radius. The turn radius is

$$r = \min\bigl(r_{\text{physics}},\ r_{\text{sector}}\bigr), \tag{4.5}$$

where $r_{\text{physics}} = v^2/(g \tan \alpha)$ with $v = 240\,\text{m/s}$ (the maximum speed in `validSpeeds`), $g = 9.81\,\text{m/s}^2$, and $\alpha = 25$ (standard rate bank angle); and $r_{\text{sector}} = 0.3 \cdot d(\mathbf{h}_1, \mathbf{h}_2)$. The physics constraint prevents excessive g-load on the aircraft; the sector constraint prevents the racetrack from protruding beyond the sector boundary.

Fuel aware lap count. The number of affordable laps is determined by available fuel. Let $F_{\text{surplus}} = F_{\text{current}} - \eta \sum_k \|\mathbf{w}_{k+1} - \mathbf{w}_k\|$ be the fuel remaining after the base plan, where η is the fuel efficiency rate (`fuelEff`). The transit fuel cost to reach the holding area and return is $F_{\text{transit}} = \eta \cdot 2 \cdot d(\mathbf{p}, \mathbf{h}_1)$, and each lap consumes $F_{\text{lap}} = \eta \cdot (2 \cdot d(\mathbf{h}_1, \mathbf{h}_2) + 2\pi r)$ (two straight legs plus two semicircular arcs). The number of laps is then

$$L = \min\Bigl(\bigl\lfloor (F_{\text{surplus}} - F_{\text{transit}}) \,/\, F_{\text{lap}} \bigr\rfloor,\ L_{\max}\Bigr) \tag{4.6}$$

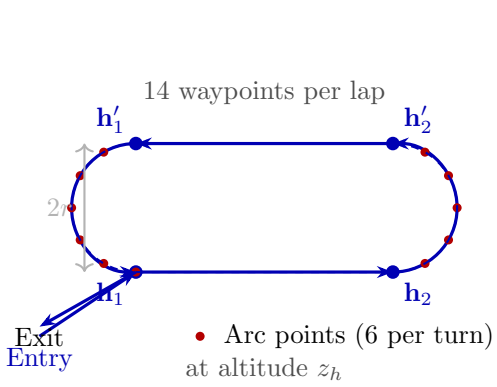
If $L \leq 0$ for all holding altitudes, no alternate can be generated and the aircraft is denied entry and initiates an emergency exit (§4.9.2).

Holding altitudes. In the parametric sweeps of §4.11, three altitude layers provide layer specific holding heights, with 1,000 ft vertical spacing between adjacent heights to ensure

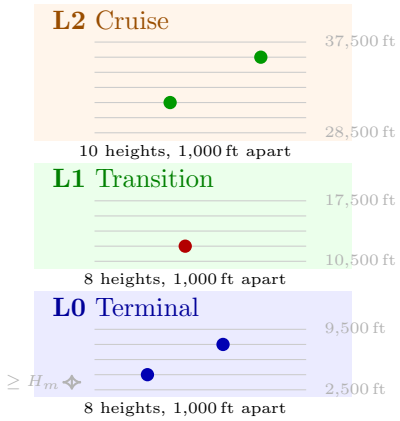
separation of at least $H_m = 304.8\,\text{m}$ (1,000 ft) between concurrent holding patterns:

- **L0 (Terminal, 0–9,999 ft):** 8 heights at 1,000 ft intervals: $\{2500, 3500, \ldots, 9500\}$ ft.
- **L1 (Transition, 10,000–17,999 ft):** 8 heights: $\{10500, 11500, \ldots, 17500\}$ ft.
- **L2 (Cruise, 18,000+ ft):** 10 heights: $\{28500, 29500, \ldots, 37500\}$ ft.

Alternates are generated for each combination of altitude and lap count, up to the configured `numAlternates` limit (default 2).



(a) Racetrack holding pattern with semicircular turns (6 arc points per turn, 14 waypoints per lap).



(b) Layer specific holding altitudes with 1,000 ft vertical spacing.

Fig. 4.6 Holding pattern geometry. (a) The racetrack orbit consists of two straight legs connected by semicircular turns, each discretized into 6 arc points (red dots), totaling 14 waypoints per lap. (b) Three altitude layers provide 8–10 holding heights at 1,000 ft vertical spacing, ensuring separation between concurrent holding patterns.

The conflict resolution hierarchy is summarized in Table 4.1.

Table 4.1: Conflict resolution hierarchy. Actions are tried in priority order; the first success terminates the search.

Priority	Action	Trigger	Outcome
1	Speed modification	`solve()` finds valid M	Modified speeds per segment
2	Holding pattern	`solve()` fails, alternate available	Racetrack orbit at z_h
3	Denied entry / C3 exit	All alternates exhausted	Aircraft diverted from sector

4.7 Liveness Challenges and Solutions

The per aircraft architecture introduced several liveness problems that did not exist in the original shared state OMNeT++ environment. Each was discovered through simulation failures, sometimes after hundreds of successful entry and exit transactions, and required a targeted fix. This section documents the problems and their solutions.

4.7.1 Paxos livelock (multi proposer contention)

Problem. When multiple aircraft attempt to enter the same sector simultaneously, each initiates a SYNOD round with its own proposal number. Aircraft A proposes n_A; aircraft B, having already promised a higher number, sends a NACK. A calls `findPropNum()` and retries with $n_A + 10{,}000$. But B has also been NACKed by A's newer promise and retries with $n_B + 10{,}000$. The proposal numbers escalate indefinitely and neither aircraft achieves a quorum of promises.

Solution. After receiving a quorum of NACKs, instead of immediately retrying, the proposer schedules a delayed SYNOD_PREP event with a random backoff drawn uniformly from $[\tau_{\min}, \tau_{\max}]$ (default: $[50, 200]$ ms). This breaks the symmetry: one proposer's retry fires first, obtains promises from all acceptors before the other proposer wakes up, and completes its SYNOD round. The mechanism is applied in four handlers: promise collection, vote collection, prepare timeout, and accept timeout.

4.7.2 Timeout explosion

Problem. Every SYNOD_PREP and SYNOD_ACPT schedules a timeout event on the local event queue. If the consensus round completes before the timeout fires, the stale timeout still executes, triggering a spurious retry that starts a new SYNOD round, which schedules its own timeout, which triggers another spurious retry, and so on. In high density scenarios, this produced exponential growth in timeout events.

Solution. Each timeout event carries the message ID (`msgID`) under which it was created. When a new consensus round starts, the sector's `lastResetMsgID` is updated to the current `msgID`. Every timeout handler checks `msgID` $\geq$ `lastResetMsgID` before acting; stale timeouts

are silently discarded.

4.7.3 Staleness detection

The `eventCounter` provides an epoch number for each sector's state. Every outgoing message includes the sender's current event counter. When a receiver's local counter is higher (meaning a TAP has completed since the message was sent), the message references an outdated φ. The receiver responds with a STALE_MSG containing its current φ, C_1, C_2, and S, allowing the sender to update its state and retry with current information.

4.7.4 Safety valves for departed aircraft

The core problem. In the original OMNeT++ architecture, all aircraft share one process. When aircraft X exits sector s, the removal is instantly visible: X disappears from S, C_1, and C_2, and no future message targets X. In our per aircraft architecture, each engine is isolated. When X completes its exit on engine E_X, engines E_A, E_B, E_C still have X in their S sets and send TAP_LEARN or SYNOD_PREP messages to X, which never responds. Timeouts fire, retries target X again, and the protocol loops indefinitely.

The Python orchestration layer includes an incremental exit synchronization mechanism that propagates completed exits across engines after each event. However, SYNOD messages carry *frozen* copies of S that can re introduce departed aircraft via state merging. Safety valves provide bounded retry guarantees even when synchronization is delayed:

Table 4.2: Safety valve parameters. Each valve provides a bounded retry guarantee that forces progress when departed aircraft cannot respond.

Valve	**Threshold**	**Action**
TO_LEARN	5 retries	Skip to TAP_AK phase (assume departed learned)
TO_AK	10 retries	Force TAP completion, clear learning locks
TO_PREP/ACCPT lock	50 deferrals	Force clear stale learning locks
SYNOD PREP	$\sim$50 rounds	Remove non responding aircraft from S
Promise/Vote flood	1 (guard flag)	Early return when `promisePhaseComplete` set

Design rule. Counter reset discipline is critical: retry counters must never be reset inside the function that is called on retries. For example, resetting `numLearnTOs` inside

`sendTAPLearn` would cause the valve to never trigger, since every retry starts with a fresh counter. The counters are only reset when a new consensus round begins (at INIT_REQ or SYNOD_PREP initiation).

4.7.5 Exit synchronization and denied entry propagation

Two Python level mechanisms bridge the isolation gap between engines:

- `_sync_exited_aircraft()`: After each processed event, the manager checks all engines for newly completed exits. For each new exit, it calls `removeAircraftFromAllSectors()` on every other engine. An incremental `_synced_exits` set prevents reprocessing.
- `_sync_rogue_aircraft()`: Same pattern for aircraft that have been **denied entry** (exhausted all alternates and initiated emergency C3 exit). The code retains the legacy name `rogue` for this state.

These mechanisms are *optimizations*, not correctness requirements. The safety valves in Table 4.2 guarantee liveness without any synchronization. But the syncs dramatically reduce wasted retry cycles and message traffic, cutting simulation time by an order of magnitude in high density scenarios.

4.8 Fuel Tracking and Aircraft Failure Modes

Each aircraft carries a fuel reserve tracked per aircraft (not per sector) in `aircraftFuel[aircraft_id]`. Fuel is initialized to `initFuel` at registration and consumed once per sector entry at TAP completion. The consumed amount is the Euclidean path distance of the committed flight plan multiplied by the fuel efficiency rate η (`fuelEff`):

$$\Delta F = \eta \sum_{k=1}^{n-1} \|\mathbf{w}_{k+1} - \mathbf{w}_k\|_2$$

A guard flag (`fuelConsumedForEntry`) prevents double consumption when `sendTAPLearn` is called multiple times for the same entry due to retries. The flag is set after the first consumption and reset when a new entry DATC begins.

Parameter calibration. In the parametric sweep experiments (§4.11), the fuel parameters are set to `initFuel` = 230 and `fuelEff` = 1.0 (abstract units per meter of path distance). These are dimensionless quantities rather than physical fuel measures. With $\eta = 1.0$, the fuel budget is simply 230 minus the total Euclidean path length traversed. This calibration was tuned so that aircraft flying standard direct routes retain sufficient fuel for approximately 2 to 3 holding laps, while aircraft that have already traversed multiple sectors or longer routes have less margin and may be denied entry. The C++ header defaults (`initFuel` = 200, `fuelEff` = 7×10^{-5}) reflect an earlier calibration; the sweep runners override them at runtime.

Holding pattern generation is fuel aware: `calcAlternates()` computes the surplus fuel available beyond what the base plan requires, and only generates holding laps that the aircraft can afford. Two failure modes arise from fuel constraints:

- **Denied entry** At planning time, if the solver fails and no alternate plan can be generated (insufficient fuel for holding), the aircraft is denied entry and initiates a C3 exit (§4.9.2), cleanly removing it from the simulation. In a real deployment, this aircraft would divert to an alternate route or airport. Because the aircraft is removed rather than admitted with a conflicting plan, the formal verification guarantees of the protocol are preserved.

- **Fuel exhaustion (simulation: CRASHED).** If fuel drops below zero during flight (after multiple holding patterns), the aircraft is removed from the simulation immediately. This represents a scenario that would not occur in practice: a real aircraft would declare a fuel emergency and receive priority handling from ATC. The simulation uses this as a worst case bound on protocol performance.

4.9 Exit Protocols

4.9.1 Normal exit (exit DATC)

When an aircraft leaves a sector, it initiates an exit DATC: the same three phase protocol (discovery, SYNOD, TAP) but with the `exitDATC` flag set. The SYNOD proposes removal of

the aircraft's plan from φ rather than addition. The aircraft uses `logicalSects`, a secondary sector state map that maintains state for sectors already physically departed, so it can continue responding to protocol messages from aircraft still in the sector.

4.9.2 Emergency exit (C3 exit DATC)

A denied entry aircraft that has exhausted all alternates initiates a C3 exit: the same protocol with the `c3ExitDATC` flag. This aborts the entry from C_1 or C_3 state, removes the aircraft from the sector's sets, and notifies all relevant aircraft that the entry attempt has been abandoned.

4.10 Tunable Parameters

The protocol exposes a set of tunable parameters through the `ConsensusKnobs` struct. Table 4.3 lists each parameter with its default value and operational range.

Table 4.3: Consensus protocol tunable parameters (`ConsensusKnobs`).

Parameter	Default	Unit	Description
`timeoutDuration`	100	ms	Message timeout before retry
`numIRAttempts`	6	-	Max INIT_REQ retries
`D_m`	9,260	m	Horizontal separation (5 NM)
`H_m`	304.8	m	Vertical separation (1,000 ft)
`numAlternates`	2	-	Holding patterns per entry
`initFuel`†	230	units	Initial fuel capacity
`fuelEff`†	1.0	units/m	Fuel consumption rate
`nackBackoffMin`	50	ms	Min random backoff after NACK
`nackBackoffMax`	200	ms	Max random backoff after NACK
`validSpeeds`	[240 . . . 120]	m/s	Speed options for `solve()`
`holdingPatternBaseDelay`	30	s	Base delay per holding orbit
`phaseDelayFactor`	0.25	-	Consensus phase delay multiplier
`MAX_LAPS`	5	-	Maximum racetrack orbits per holding pattern
`holdHeights`	layer specific		8–10 altitudes per layer (§4.6.3)

† Values shown are those used in the parametric sweep experiments (§4.11). The C++ header defaults (`initFuel` = 200, `fuelEff` = 7×10^{-5}) are overridden at runtime by the sweep runners.

These parameters form the search space Θ for the Bayesian optimization loop in Chapter 5. The BO evaluates each configuration by running the full consensus simulation and

scoring on entry success rate, NMAC count, and resource usage.

4.11 Protocol Evaluation: Parametric Sweep

To stress test the consensus protocol across a range of traffic densities and airspace configurations, we designed two parametric sweep experiments: one using synthetically generated traffic and one using real world flight data from the FAA System Wide Information Management (SWIM) program [10]. Both vary grid size (number of sectors) and aircraft count as independent variables, measuring admission success rate, conflict resolution behavior, failure modes (denied entries and fuel exhaustion), near mid air collisions (NMACs), and computational cost.

4.11.1 Synthetic traffic sweep

The synthetic sweep uses randomly generated aircraft with controlled parameters. The airspace is a 100×100 NM square centered at (40.0N, 73.0W), partitioned into an $N \times N \times 3$ grid of sectors, where $N \in \{2, 4, 8, 16\}$ and the 3 altitude layers are fixed: L0 (Terminal, 0–9,999 ft), L1 (Transition, 10,000–17,999 ft), and L2 (Cruise, 18,000+ ft). This yields sector counts of 12, 48, 192, and 768.

Aircraft counts are drawn from $\{10, 20, 40, 80, 160\}$. Each aircraft is assigned to a layer according to layer weights $[0.10, 0.20, 0.70]$ (L0, L1, L2), reflecting the distribution observed in real world traffic where cruise altitudes dominate. Entry positions are placed on the airspace boundary with random headings biased inward. A fixed random seed (42) ensures full reproducibility: the first N aircraft from a count-N run are identical to the first N aircraft from any higher-count run, enabling controlled comparison where adding aircraft never removes existing ones.

4.11.2 JFK real world traffic sweep

The JFK sweep uses 80 real aircraft seeds extracted from FAA SWIM data captured near John F. Kennedy International Airport in December 2025. Each seed provides the aircraft's

callsign, geographic position (latitude, longitude, altitude), ground speed, and heading at a single snapshot in time.

The airspace is a 100×100 NM square centered at (40.58N, 73.92W), partitioned into $N \times N \times 3$ grids with $N \in \{2, 3, 4, 5, 6\}$, yielding 12, 27, 48, 75, and 108 sectors. Aircraft counts range from 10 to 80 in steps of 10.

Incremental subset selection. To ensure controlled density scaling, all 80 seeds are shuffled with a fixed secondary seed (43) and the first k seeds form the k-aircraft set. This guarantees that set$[k] \subset$ set$[k+10]$: every density level is a strict superset of all lower density levels, so observed differences are attributable solely to the additional aircraft.

Vertical velocity assignment. The SWIM snapshots record only ground speed and heading; vertical velocity is assigned synthetically for L0 (Terminal) aircraft to create realistic mixed traffic:

- **10 descenders**: Selected from aircraft at or above 5,000 ft. Descent rate is set to 60% of the maximum safe rate, where the maximum is computed as $v_{z,\max} = (\text{alt} - 1{,}000)/t_{\text{transit}}$ to ensure the aircraft does not reach the ground during its sector traversal.
- **15 climbers**: Assigned 500, 750, or 1,000 ft/min uniformly at random.
- **Remaining ∼20 cruisers**: $v_z = 0$.

4.11.3 Execution pipeline

Both sweep runners use an identical two phase pipeline:

Phase 1 (First sector entry). All aircraft enter their first sector sequentially, spaced 3 s apart to avoid simultaneous protocol contention. After all entries complete, a global NMAC check verifies separation across all occupied sectors. Aircraft whose entire path lies within a single sector are exited immediately.

Phase 2 (Multi hop traversal and exit). For each subsequent hop index $h = 1, 2, \ldots, h_{\max}$:

1. All aircraft with a hop h enter their next sector (with `previous_sector_id` set for handover).

2. A global NMAC check is performed while all aircraft are present (sector φs are maximally full).

3. Exits from the previous sector are processed, serialized by sector to avoid concurrent exit contention.

4. Aircraft on their final hop exit the airspace entirely.

After all hops, any stuck exits are retried twice with additional event processing. The consensus knobs used in both sweeps are listed in Table 4.3.

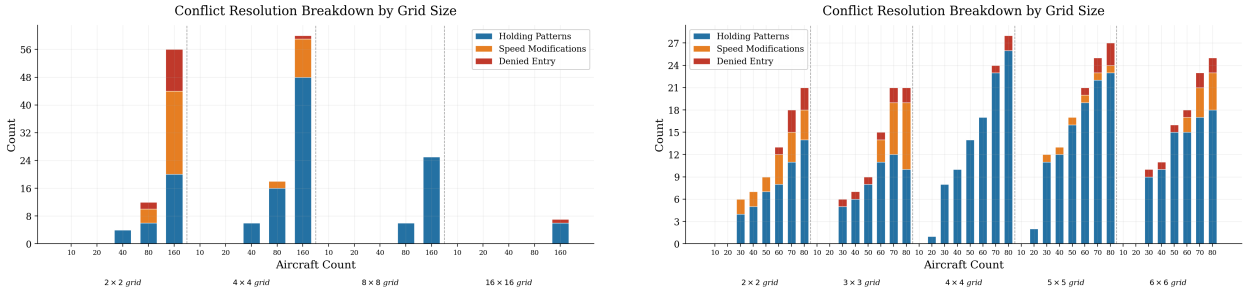
4.11.4 Metrics collected

Each run produces a single CSV row recording the metrics in Table 4.4.

Table 4.4: Metrics collected per parametric sweep run.

Metric	Description
`total_entries_attempted`	Sector entry requests across all hops
`total_entries_admitted`	Entries that completed successfully
`denied_entries`	Entries that failed (denied entry or exhausted retries)
`holding_patterns`	Times a holding pattern alternate was selected
`speed_modifications`	Times `solve()` found a valid speed assignment
`denied_aircraft_count`	Number of aircraft denied entry
`crashed_count`	Aircraft whose fuel dropped below zero
`nmac_count`	Unique aircraft pairs violating D_m/H_m separation
`total_exits_completed`	Sector exits that completed successfully
`soft_exits`	Exits handled by safety valve (forced completion)
`max_quorum_size`	Largest quorum observed in any SYNOD round
`total_retries`	Total protocol message retries across all engines
`avg_holding_rounds`	Mean holding laps per aircraft that held
`phase1_seconds`	Wall clock time for Phase 1
`phase2_seconds`	Wall clock time for Phase 2
`wall_clock_seconds`	Total wall clock execution time

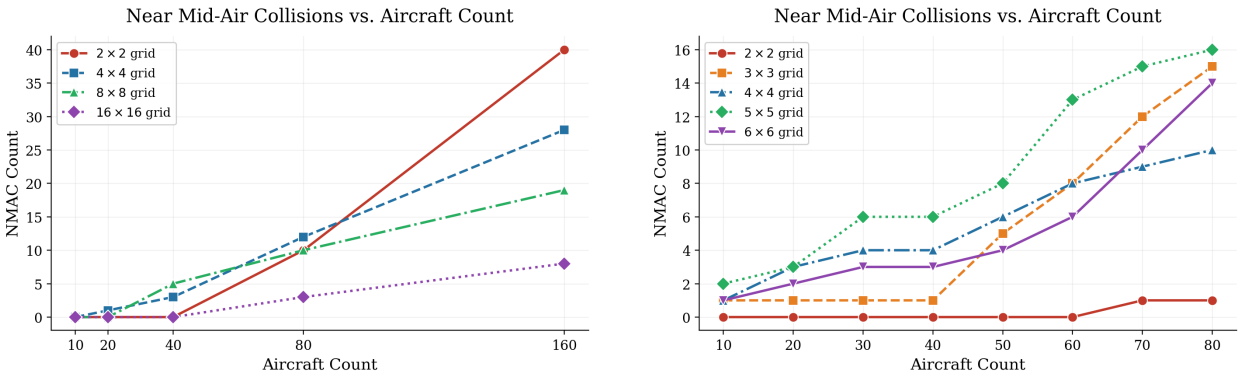
4.11.5 Results



(a) Synthetic sweep **(b) JFK real world sweep**

Fig. 4.7 Conflict resolution breakdown by grid size and aircraft count. Stacked bars show holding patterns, speed modifications, and denied entries for each configuration.

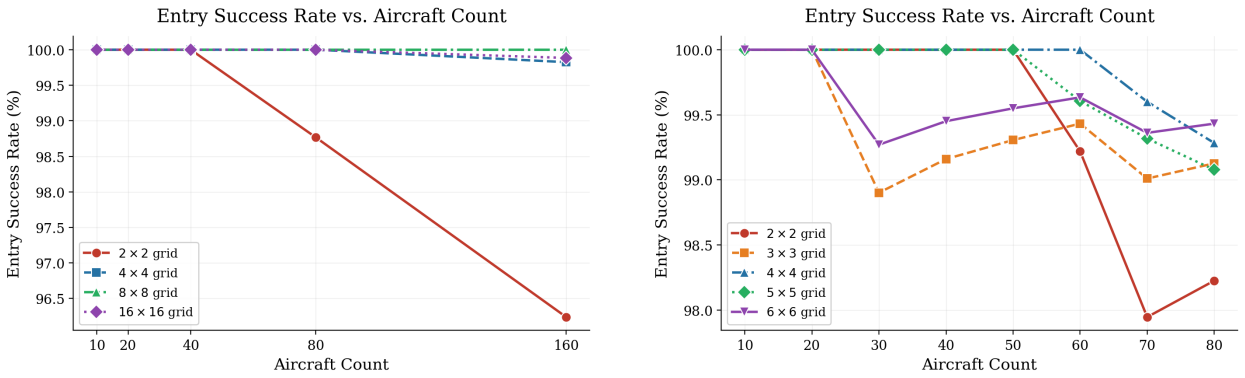
Holding patterns are the most frequently used conflict resolution mechanism across both sweeps. Although speed modification is the first action attempted (Table 4.1), the solver often cannot find a valid speed that resolves the conflict, causing the protocol to fall back to holding pattern assignment. In the synthetic sweep, the 4×4 grid accumulates the most total conflict actions at 160 aircraft ($\sim$60), because its mid range sector density concentrates enough traffic per sector to trigger geometric conflicts while still leaving room for holding orbits. The 2×2 grid shifts heavily toward speed modifications and denied entries at high counts. Its large sectors pack so many aircraft that the holding-area geometry frequently cannot find a conflict free orbit, forcing the protocol to escalate. Fine grids (8×8, 16×16) produce far fewer actions overall because each sector contains only a handful of aircraft and conflicts are rare. In the JFK sweep, conflict actions are more evenly distributed across grid sizes. Real flight paths converge on JFK's approach corridors regardless of how the airspace is partitioned, so even fine grids (5×5, 6×6) see meaningful conflict counts. Denied entries remain a small fraction in both sweeps ($\leq$ 3 in JFK, $\leq$ 12 in synthetic at the extreme), confirming that the holding + speed mechanism resolves the vast majority of conflicts before alternate generation fails entirely.



(a) Synthetic sweep **(b) JFK real world sweep**

Fig. 4.8 Near mid air collisions vs. aircraft count.

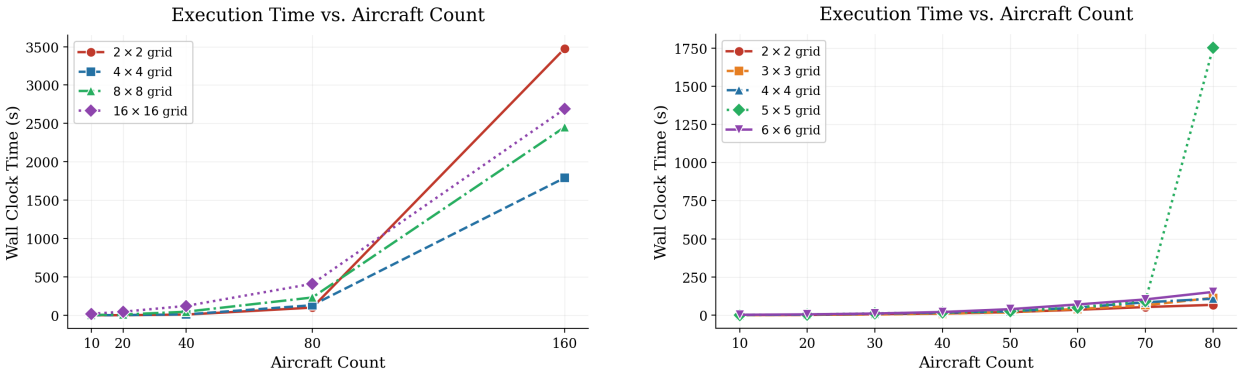
NMACs grow roughly linearly with aircraft count but depend strongly on grid size. Notably, the direction of that dependence reverses between synthetic and real world traffic. In the synthetic sweep, the 2×2 grid is the worst performer, reaching 40 NMACs at 160 aircraft. Larger sectors mean fewer internal boundaries at which the protocol can detect and separate converging traffic, so conflicts slip through unresolved. The 16×16 grid holds NMACs to just 8 at 160 aircraft, confirming that finer partitioning improves separation by giving every aircraft a smaller, more tightly managed volume. In the JFK sweep, however, the 2×2 grid produces the *fewest* NMACs (0–1 across all counts). With only 12 large sectors, most of JFK's converging approach traffic stays within a single sector and is handled by a single consensus round. Finer grids (5×5, 6×6) split approach corridors across multiple sector boundaries, and each boundary crossing creates a new opportunity for an inter sector separation violation that the protocol must catch during transit. The 5×5 grid peaks at 16 NMACs for 80 aircraft. This reversal highlights how uniform grid partitioning interacts with the non uniform route structure of a real terminal area: the "right" grid size for safety depends on traffic geometry, not just density.



(a) Synthetic sweep **(b) JFK real world sweep**

Fig. 4.9 Entry success rate vs. aircraft count.

The entry success rate measures the fraction of attempted sector entries that the protocol admits. An aircraft is denied entry only when all conflict resolution alternatives are exhausted during the consensus process for that sector. In the synthetic sweep, all grid sizes maintain a success rate above 96% even at the highest load of 160 aircraft. The 2×2 grid degrades the most, dropping to $\sim$96.2% at 160 aircraft, while the 8×8 grid stays at 100% across the entire range (zero denied entries). The 4×4 and 16×16 grids each see a single denied entry at the 160 count mark, keeping them above 99.3%. In the JFK sweep the picture is similar: all configurations remain above 98% through 80 aircraft. The 3×3 grid shows the earliest degradation, dipping to $\sim$98.9% at 30 aircraft, because its sector boundaries happen to intersect JFK's busiest approach fixes. Overall, the consistently high entry rates demonstrate that the consensus protocol can absorb realistic traffic loads without rejecting more than a handful of aircraft, a prerequisite for practical deployment.



(a) Synthetic sweep **(b) JFK real world sweep**

Fig. 4.10 Wall clock execution time vs. aircraft count.

Wall clock time scales super-linearly with aircraft count, driven by the quadratic growth in pairwise conflict checks and the increasing depth of the Paxos message tree as quorums grow. In the synthetic sweep, the 2×2 grid is the slowest configuration at high load ($\sim$3 500 s for 160 aircraft) because each of its 12 large sectors hosts a large quorum and every consensus round involves many participants. The 4×4 grid is the most efficient ($\sim$1 800 s at 160 aircraft), striking a balance between sector count and quorum size. In the JFK sweep, most configurations finish in under 200 s for 80 aircraft. The striking outlier is the 5×5 grid at 80 aircraft, which takes $\sim$1 752 s, roughly $10\times$ the next slowest configuration. Inspection of the retry counters shows that 74 380 of the 74 550 total retries in that run occurred in a single sector at quorum size 10. This is classic Paxos livelock: multiple aircraft propose simultaneously, each incrementing its proposal number past the others in an unbroken cycle. The random backoff mechanism described in §4.7.1 mitigates this contention in the common case, but under extreme density and deterministic seeding it does not fully break the symmetry. This livelock is not specific to the 5×5 grid; it can occur in *any* configuration whenever a sector's quorum size and traffic density combine to sustain simultaneous proposals long enough to exhaust the backoff window. In our experiments it manifested only once, in this particular seed and grid pairing, and all other JFK configurations completed without incident. We retain the outlier in our results for transparency, as it illustrates a genuine tail risk of Paxos based consensus under extreme contention rather than a systemic limitation of the protocol. Additional appendix plots are provided in Appendix 9.1.

5 Configuration Search via Bayesian Optimization

5.1 Chapter contributions

This chapter uses Bayesian Optimization with a Gaussian Process surrogate to tune eight consensus protocol parameters per airport, replacing infeasible grid search with a sample efficient approach. We present the optimization formulation, results across three airports, and analysis of why each environment requires a qualitatively different tuning.

5.2 Overview

Using $J(\theta)$ from §2.4, we search the configuration space Θ for high quality sectorizations subject to safety constraints.

Role in the pipeline. This chapter sits at the end of the pipeline. The consensus protocol from Chapter 4 has eight tunable parameters that affect how well it performs at each airport. Bayesian Optimization finds the best settings automatically, removing the need for manual tuning and revealing that different airports need fundamentally different configurations.

5.3 Why Gaussian Process Based Optimization

Each candidate configuration must be evaluated by running the full consensus protocol simulation for 30 simulated minutes, making every function evaluation expensive. The eight protocol parameters that govern timeout behavior, retry persistence, contention backoff, and conflict resolution interact in ways that produce no closed form gradient. This places the problem in the regime of expensive, derivative free, black box optimization [24].

Grid search is infeasible: even a coarse five level discretisation of each of the eight parameters would require $5^8 = 390{,}625$ simulations. Random search improves on this by sampling uniformly, but it allocates trials indiscriminately and cannot concentrate effort on promising regions of the space. Evolutionary strategies such as CMAES [25] require population sizes on the order of tens to hundreds of individuals per generation, which again exceeds the

affordable simulation budget [24].

Bayesian Optimization with a Gaussian Process surrogate addresses all three limitations. The GP builds a probabilistic model of the objective surface from observed trials and provides both a predicted mean and a calibrated uncertainty at every unobserved point [26]. The acquisition function (Expected Improvement) uses this uncertainty to balance exploitation of known good regions with exploration of uncertain ones, selecting the single most informative point to evaluate next. Empirically, GP based Bayesian Optimization converges to near optimal configurations in 30 to 50 evaluations for problems of this dimensionality [27].

Alternative surrogate models were considered. Random forest surrogates (SMAC) produce piecewise constant uncertainty estimates that lack the smooth posterior surface needed for gradient based acquisition optimization. Neural network surrogates require substantially more training data than the 50 trial budget permits. The GP's automatic relevance determination kernel additionally yields per parameter lengthscales whose inverses serve as importance scores (Equation 5.5), providing interpretability that we report in §5.8.

5.4 Search Space and Constraints

Of the parameters listed in Table 4.3, the horizontal separation $D = 5\,\text{NM}$ and vertical separation $H = 1{,}000\,\text{ft}$ are fixed by ICAO and FAA regulation and are not subject to optimization. The remaining eight protocol level parameters form the search space:

$$\Theta = \prod_{k=1}^{8} [\ell_k, u_k] \subset \mathbb{R}^8 \tag{5.1}$$

where ℓ_k and u_k are the lower and upper bounds for parameter k. Two of the eight dimensions (`numIRAttempts`, `numAlternates`) are integers; we optimise over the continuous relaxation and round to the nearest integer before each simulation evaluation.

Table 5.1: BO search bounds. Tightened bounds are based on empirical analysis of a 50 trial campaign on ORD.

Parameter	Original	Tightened	Best (ORD)	Unit	Rationale for tightening
`timeoutDuration`	[0.5, 5.0]	(unchanged)	0.50	s	No timeout correlation observed
`numIRAttempts`	[2, 15]	[2, 10]	15	–	Combinatorial blowup with many conflict pairs
`numAlternates`	[1, 8]	[1, 4]	1	–	4/5 timeouts had value 8; best trial used 1
`phaseDelayFactor`	[0.05, 0.5]	(unchanged)	0.50	–	No timeout correlation observed
`nackBackoffMin`	[0.01, 0.5]	(unchanged)	0.43	s	Cross parameter constraint added
`nackBackoffMax`	[0.1, 2.0]	(unchanged)	0.55	s	Cross parameter constraint added
`startIRTime`	[0.001, 2.0]	[0.001, 1.0]	0.001	s	2 s delay compounds conflicts; best used 0.001
`solveTimeoutSec`	[1, 30]	[1, 10]	1.0	s	30 s solver × 15 attempts = hours

Bound tightening. An initial 50 trial campaign on ORD (Chicago O'Hare, 47 active aircraft) produced five timed out trials (score = −50). All five shared configurations in the extremes of `solveTimeoutSec`, `numAlternates`, `startIRTime`, or `numIRAttempts`. The best scoring trial (score = 127.07) used values well within the tighter ranges. Bounds were reduced to exclude the pathological region while preserving the productive portion of the space.

Cross parameter constraint. Two of the five timed out trials had `nackBackoffMin` > `nackBackoffMax`, creating an inverted backoff window that collapsed the retry interval

and caused the protocol to spin in tight NACK loops. We enforce `nackBackoffMin` < `nackBackoffMax` by swapping the two values whenever the constraint is violated. This preserves the sampled magnitudes while fixing the logical inconsistency.

5.5 Objective Function

Each trial evaluates a candidate parameter vector $\theta \in \Theta$ by running the consensus simulation for 1,800 simulated seconds (30 minutes) and computing a scalar score. The 30 minute window was chosen because the densest traffic periods in our dataset files fall within the final 30 minutes of each recording, so this window captures the most demanding operating conditions.

$$J(\theta) = 100 \cdot r_{\text{success}} - 30 \cdot r_{\text{hold}} - 10 \cdot r_{\text{speed}} - 2 \cdot \bar{n}_{\text{retry}} \tag{5.2}$$

where

- $r_{\text{success}} = n_{\text{admitted}}/n_{\text{initiated}}$ is the admission throughput ratio: the total number of successful sector admissions divided by the number of consensus-initiated entry requests. Because each aircraft may traverse multiple sectors during the 30-minute window, r_{success} can exceed 1.0,
- $r_{\text{hold}} = n_{\text{hold}}/n_{\text{admitted}}$ is the fraction of admissions that required a holding pattern,
- $r_{\text{speed}} = n_{\text{speed}}/n_{\text{admitted}}$ is the fraction of admissions whose speed was modified by the conflict solver,
- $\bar{n}_{\text{retry}} = n_{\text{retries}}/n_{\text{initiated}}$ is the average number of protocol retries per initiated request.

The coefficients were chosen based on intuition and operational reasoning rather than derived from data. The leading term ($\times 100$) makes entry success the dominant objective. Holding patterns receive a larger penalty ($\times 30$) than speed modifications ($\times 10$) because holding consumes more fuel and introduces greater delay. Retries penalize protocol overhead but at a low weight ($\times 2$) since they are invisible to the end user.

Two hard penalties override the continuous score:

$$J(\theta) = \begin{cases} -100 & \text{if } n_{\text{NMAC}} > 0 \\ -50 & \text{if wall clock timeout exceeded} \end{cases} \tag{5.3}$$

The NMAC penalty (-100) is the lowest possible score and ensures that any configuration producing a near mid air collision is dominated by every safe configuration. The timeout penalty (-50) is applied when the consensus solver fails to terminate within the per trial wall clock budget, indicating a pathological parameter combination.

5.6 Acquisition Function and Trial Budget

The surrogate model is a single task Gaussian Process with a constant mean function and a Matérn 5/2 covariance kernel with automatic relevance determination (ARD) [26]:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \prod_{d=1}^{8} k_{\text{Matérn}}^{5/2}\left(\frac{|x_d - x'_d|}{\ell_d}\right) \tag{5.4}$$

The ARD parameterisation assigns a separate lengthscale ℓ_d to each input dimension. After fitting, dimensions with shorter lengthscales exhibit greater influence on the objective. We report normalized importance scores as

$$w_d = \frac{1/\ell_d}{\sum_{j=1}^{8} 1/\ell_j} \tag{5.5}$$

Observed scores are normalized to zero mean and unit variance $(Y - \bar{Y})/\sigma_Y$ before fitting the GP to improve numerical conditioning of the Cholesky decomposition.

The acquisition function is Expected Improvement (EI) [27, 28]:

$$\alpha_{\text{EI}}(\mathbf{x}) = \mathbb{E}\big[\max\big(f(\mathbf{x}) - f^*,\ 0\big)\big] \tag{5.6}$$

where f^* is the best observed score. Under the GP posterior, this expectation has a closed form involving the predictive mean and variance. Maximisation of α_{EI} is performed via L-BFGS-B with 10 random restarts initialised from 256 raw quasi random samples across

the unit hypercube.

The total budget is 50 trials per airport: 10 initial samples followed by 40 GP guided iterations. Early stopping is disabled to ensure full exploration of the search space.

5.7 Initialization and Checkpointing

The 10 initial samples are drawn via Latin Hypercube Sampling (LHS) [29] with a fixed random seed (42) for reproducibility. LHS partitions each dimension into n equal probability strata and places exactly one sample per stratum, providing better coverage of the 8 dimensional space than uniform random sampling with the same budget.

Each trial is executed in a forked child process with a configurable wall clock timeout. The parent process monitors elapsed time and terminates the child if the timeout is exceeded, assigning a score of -50. This mechanism is necessary because the C++ consensus solver can enter long running states on pathological parameter combinations, and Python level signal handlers do not interrupt native code execution.

5.8 Results

The consensus protocol has eight tunable parameters, and the right settings depend on the traffic scenario. The purpose of this section is not to find a single universal configuration but to demonstrate that Bayesian Optimization can automatically find good configurations for any given airport and traffic density, and that the GP surrogate reveals which parameters drive performance under different conditions.

We ran 50 trial campaigns on three airports: LAX with 31 active aircraft, ORD with 47 active aircraft, and DFW with 60 active aircraft. All simulations ran on Linux x86_64 servers using the same protocol code, search bounds, and scoring function. Table 5.2 compares the best configurations found for each airport.

The three airports produce qualitatively different optimal configurations that follow a clear pattern tied to traffic density. LAX and ORD both converge to aggressive strategies with short timeouts (0.5 s) and a single alternate, but they differ in how much time they

Table 5.2: Best configurations found by Bayesian Optimization.

Parameter	**LAX**	**ORD**	**DFW**
Timeout Duration (s)	0.50	0.50	2.93
IR Attempts	10	15	4
Alternates	1	1	2
Phase Delay Factor	0.50	0.50	0.29
NACK Backoff Min (s)	0.50	0.43	0.50
NACK Backoff Max (s)	2.00	0.55	1.20
Start IR Time (s)	1.00	0.001	1.00
Solve Timeout (s)	10.0	1.0	9.1
Best Score	**189.8**	**127.1**	**100.8**
Timeouts	0 / 50	5 / 50	1 / 50

allocate to the solver: LAX uses a high solve timeout (10.0 s) while ORD uses a very low one (1.0 s). With only 31 aircraft, LAX has minimal contention, so each consensus round can afford to let the solver work longer. ORD, at 47 aircraft, benefits from resolving rounds quickly so that other aircraft can proceed. DFW requires the most patient strategy: a longer timeout (2.9 s), a higher solve timeout (9.1 s), and two alternates. With 60 aircraft creating more concurrent consensus rounds competing for sector locks, longer timeouts give the protocol more time to resolve contention before declaring failure, and having backup alternates available helps when the first proposal is rejected.

5.8.1 Convergence

Figure 5.1 shows the convergence behavior of all three campaigns. The first 10 trials in each campaign are Latin Hypercube samples (marked LHS), which provide broad coverage of the search space. The remaining 40 trials are guided by the GP surrogate.

LAX finds a strong score of 159.9 during the LHS phase at trial 3 and steadily improves through the BO phase, reaching its best score of 189.8 at trial 35 with zero timeouts across all 50 trials. The low density of 31 aircraft means that even poorly chosen configurations tend to complete without timing out. ORD finds a score of 106.7 during the LHS phase and improves to 127.1 at trial 40. Five trials timed out, all in the BO phase, indicating that the GP occasionally proposes configurations near the boundary of the feasible region. DFW converges faster, reaching its best score of 100.8 at trial 19, with only one timeout across all

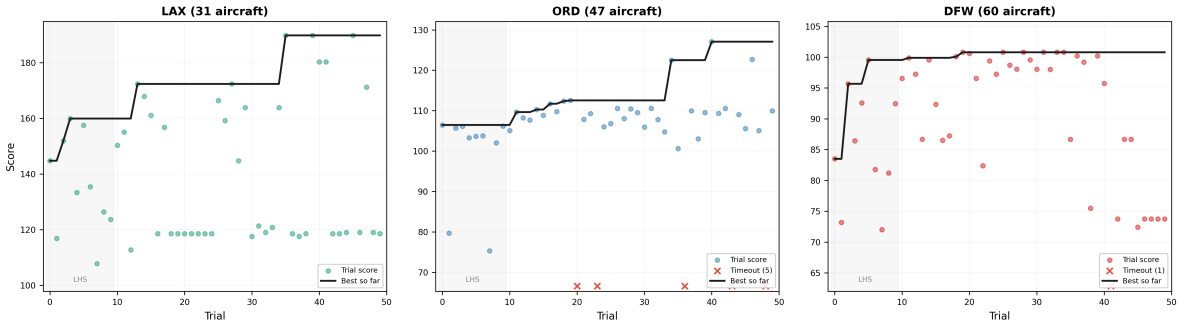


Fig. 5.1 Convergence of Bayesian Optimization for LAX (left), ORD (center), and DFW (right). Colored dots show individual trial scores. The black line tracks the best score observed so far. Red crosses mark trials that timed out (score not shown on the y axis). The shaded region marks the initial LHS sampling phase.

50 trials.

5.8.2 Parameter Importance

Figure 5.2 shows the relative importance of each parameter as computed from the fitted GP lengthscales. A shorter lengthscale means the GP output changes more rapidly with respect to that parameter, indicating higher sensitivity.

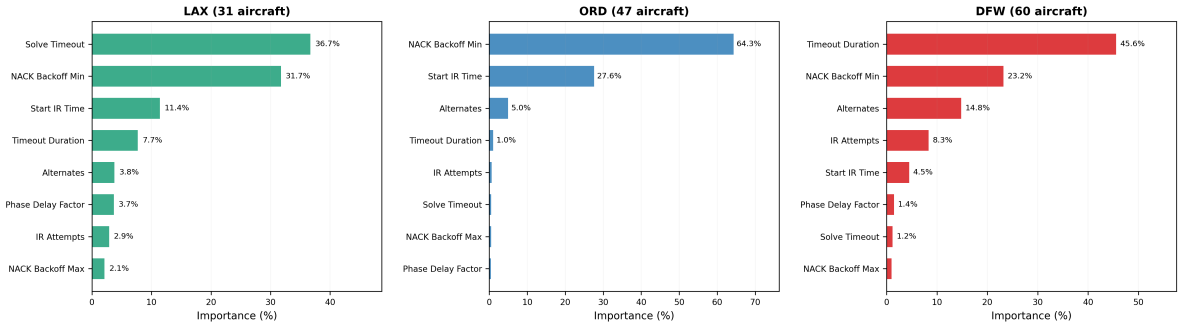


Fig. 5.2 Parameter importance derived from GP lengthscales for LAX (left), ORD (center), and DFW (right). Importance is computed as the normalized inverse lengthscale for each parameter.

At LAX, Solve Timeout leads at 36.7%, followed by NACK Backoff Min at 31.7% and Start IR Time at 11.4%. With only 31 aircraft, contention is low, and the protocol benefits most from giving the solver enough time to find a valid configuration on each round. The remaining parameters have relatively little effect because the low traffic density means most

configurations work reasonably well regardless of timeout or retry settings.

At ORD, NACK Backoff Min dominates with 64.3% importance, followed by Start IR Time at 27.6%. These two parameters together account for over 90% of the objective variation. The backoff minimum controls how quickly a sector retries after receiving a negative acknowledgement, and the start IR time controls when the initial reconfiguration request is sent. At 47 aircraft, contention is moderate, so the retry timing matters more than structural parameters like the number of alternates.

At DFW, the importance distribution is broader. Timeout Duration leads at 45.6%, followed by NACK Backoff Min at 23.2% and Alternates at 14.8%. With 60 aircraft, the protocol faces more concurrent consensus rounds, and the timeout duration determines whether a round completes or is abandoned. The number of alternates also becomes relevant because sectors under higher load benefit from having backup configurations available when the first proposal is rejected.

The fact that the GP learned different importance rankings for the three airports validates the approach. A single hand tuned configuration would not adapt to the different traffic characteristics of each airport. The GP surrogate captures these differences and the acquisition function exploits them to find configurations tailored to each scenario.

5.8.3 Score Interpretation

The score J is an unbounded composite metric, not a percentage. The positive term $100 \times r_{\text{success}}$ rewards throughput while the penalty terms for holdings, speed modifications, and retries subtract from it, so higher scores indicate better overall protocol performance. The scores show a clear trend with traffic density: LAX scores 189.8, ORD scores 127.1, and DFW scores 100.8. As the number of aircraft increases, the protocol faces more contention and the best achievable score decreases, reflecting the increased difficulty of coordinating consensus rounds at higher densities.

5.8.4 Takeaways

These results support three conclusions. First, the consensus protocol is sensitive to its configuration: all three airports converge to qualitatively different parameter settings, so a

single default configuration cannot serve all airports. The trend from aggressive (LAX) to patient (DFW) strategies shows that traffic density systematically changes which protocol mechanisms matter. Second, Bayesian Optimization with a GP surrogate is an effective method for finding good configurations within a practical budget of 50 simulation runs, even as the difficulty scales from 31 to 60 aircraft. Third, the GP lengthscale analysis provides interpretable insight into which protocol mechanisms matter most at each traffic scale, information that would be difficult to obtain through manual experimentation. In a real DAC deployment, this tuning framework could be applied to any new airspace scenario to find a suitable protocol configuration before the system goes live. The three campaigns used real traffic recordings from specific time slots: ORD and DFW from Sunday afternoon (12 PM) and LAX from Saturday night (9 PM). Airline schedules are highly repetitive on a weekly basis, with the same carriers flying roughly the same routes at the same times each week. A configuration tuned for a given airport and time slot should therefore remain effective across weeks without retuning, making the single upfront cost of 50 BO trials a practical investment.

6 Limitations

6.1 Straight Line Flight Paths

The simulation models en route cruising traffic well. Aircraft fly between resolved NASR and CIFP fix points with assigned headings, speeds, and vertical rates. The conflict detection, consensus protocol, and parametric sweep results all validate under these conditions. However, between fix points the lateral path is always a straight line. This limits coverage of airport terminal regions, where aircraft follow very dynamic, curved, altitude constrained procedures.

Specifically, the simulation does not model:

- Standard Instrument Departures (SIDs), Standard Terminal Arrival Routes (STARs), and instrument approaches with curved segments.
- Continuous heading, speed, and altitude changes between fixes.
- Tighter spacing, converging arrivals, and parallel runway operations.
- Go arounds and missed approaches.

The reason is data availability. The FAA SWIM feed provides position snapshots at discrete intervals, not the curved path an aircraft follows between them. NASR and CIFP fix points were resolved where possible, but the simulation propagates straight lines between those fixes. Full procedure waypoint data exists in the CIFP database, but vector waypoints, naturally, would not be a part of the database. Therefore, it is impossible to model the real world dynamic paths that aircraft take near a terminal region.

The architecture supports this extension. Protocol messages already carry full waypoint sequences, and the pairwise conflict check handles arbitrary waypoint counts. The gap is in the traffic modeling and flight plan generation layers, not in the coordination mechanism.

6.2 Simplified Aircraft Dynamics

Each aircraft is modeled as a point in 3D space that interpolates linearly between waypoints. The simulation gradually moves each aircraft along its route, smoothly transitioning

latitude, longitude, and altitude over the duration of each segment. However, there is no aerodynamic model underneath this motion. There is no lift, drag, thrust, bank angle, or turn radius. When an aircraft reaches one waypoint and begins the next segment, it transitions to the new heading and vertical rate with a linear ramp rather than following the curved path that a real aircraft would fly due to its bank angle and turn dynamics.

This simplification is reasonable for en route cruising, where aircraft fly straight and level for long stretches. It is less accurate at waypoint transitions and during maneuvering phases like holding pattern entries or conflict avoidance turns, where the actual bank angle and turn radius affect the aircraft's path. The consensus protocol and conflict detection still work correctly under this model because they operate on waypoint geometry, not on the aircraft's physical response to commands. A higher fidelity dynamics model would change the trajectories that feed into the protocol, but would not require changes to the protocol itself.

6.3 Terrain and Obstacle Avoidance

The simulation treats airspace as open three dimensional space with no terrain model. Mountains, tall structures, restricted or prohibited airspace zones, and temporary flight restrictions are not represented. In the real world, flight plans must respect minimum safe altitudes, terrain clearance requirements, and obstacle departure procedures, particularly near airports where terrain can constrain available approach and departure corridors. The consensus protocol admits or rejects aircraft based solely on pairwise conflict geometry between flight plans; it does not check whether a proposed trajectory clears terrain or avoids restricted zones. Integrating a terrain and obstacle database (for example, digital elevation data combined with FAA obstacle and airspace boundary files) into the conflict check would extend the protocol to reject plans that are conflict free with other aircraft but unsafe with respect to the ground environment.

6.4 Deterministic Communication

The consensus protocol does model communication delays. Every message is scheduled with a configurable delivery delay, and the NACK backoff mechanism adds a random wait time before retrying after a rejection. However, these delays are fixed and uniform. There is no packet loss, no jitter, and no latency that varies with distance or network load. The following paragraphs address specific concerns that arise from this simplification.

Variable latency. In a real deployment, aircraft would communicate over ADS-B or a dedicated air to air data link with variable latency. Typical ADS-B message latency is on the order of 120 ms. The BO-tuned timeout durations in our experiments range from 0.5 to 5.0 s, one to two orders of magnitude larger than realistic link latency. This margin means the timeout and retry mechanisms (timeout duration, NACK backoff range, and IR attempt count) would absorb realistic jitter without protocol changes. Higher latency would increase the number of retries and lower throughput, but would not violate safety: as noted in §4, Paxos safety holds under arbitrary delays.

Packet loss. Paxos tolerates lost messages by design. A proposer that receives no response simply times out and retries with a new round number. The safety valves in Table 4.2 bound the number of retries and force progress when participants are unreachable. In the worst case, the entering aircraft exhausts its retry budget and is denied entry, which is a safe outcome: the aircraft does not enter the sector.

Clock drift. The simulation uses a global event queue that enforces a single time line across all engines. A real deployment would require synchronized clocks. Aviation already mandates GPS-disciplined time references (UTC via GPS, accuracy $\sim$100 ns), which makes clock drift negligible relative to protocol timeouts on the order of seconds.

Malicious or faulty nodes. The protocol assumes crash stop failures: an aircraft may be slow or unreachable, but it never sends deliberately incorrect messages. Byzantine fault tolerance (e.g., an aircraft spoofing ACKs to force a conflicting admission) is not addressed and would require BFT protocol variants. However, manned aviation already enforces identity verification through Mode S transponder codes, and ADS-B messages are tied to unique ICAO addresses. The crash stop assumption is therefore reasonable for the manned aircraft

domain, though UAS or drone integration may warrant stronger guarantees.

Summary. The current results measure protocol throughput under ideal communication. Variable delays and packet loss would lower BO scores (more timeouts, more retries, more denied entries) but would not create unsafe states. Testing under lossy or variable delay communication remains necessary to quantify this degradation and is discussed as future work in §7.

6.5 Limited Trial Budget

Each Bayesian Optimization campaign was run for 50 trials. The convergence plots in Figure 5.1 show that some campaigns were still improving when the budget ran out, meaning that additional trials could potentially find better configurations. However, each 50 trial campaign took one to two days of wall clock time on a Linux server, depending on the number of aircraft. Under the time constraints of this thesis, running significantly more trials per airport was not practical. A larger budget or access to faster hardware would allow the optimizer to explore more of the search space and potentially converge to stronger configurations.

6.6 Fixed Wing Aircraft Only

The simulation models only fixed wing aircraft. There are no helicopters, VTOLs, or small unmanned aircraft systems in any of the traffic scenarios. Major airports like JFK have significant rotary wing traffic, and the growth of Advanced Air Mobility and eVTOL operations means that mixed vehicle types will become increasingly common in terminal airspace.

There are two reasons for this gap. First, the FAA SWIM feeds that supply our traffic data primarily capture transponder equipped fixed wing aircraft. Rotary wing and small UAS traffic is underrepresented or absent in these feeds, so there was no data to model them against. Second, these vehicle types have fundamentally different flight dynamics. Helicopters can hover, VTOLs transition between vertical and horizontal flight, and small drones operate at low altitudes with sharp turns and stops. The current point mass propa-

gation model does not capture any of these behaviors.

The consensus protocol itself is vehicle agnostic. It operates on waypoint geometry and does not assume anything about how an aircraft moves between waypoints. Adding new vehicle types would require extending the traffic model and flight plan generation layers to produce appropriate waypoint sequences and dynamics for each vehicle class, but the coordination mechanism would not need to change. This is a realistic and integrable extension once the data and modeling resources are available.

7 Extensions and Future Work

The decentralized consensus protocol presented in Chapter 4 resolves conflicts through a fixed cascade: the coordinator proposes alternates that include holding patterns, speed modifications, or, as a last resort, denied entry declarations. The parametric sweeps in §4.11.5 show that this approach achieves entry success rates above 96% and bounds NMAC counts across a wide range of traffic densities. Yet the protocol's behavior is sensitive to grid granularity in ways that depend on the underlying route structure. The reversal in NMAC rankings between synthetic and JFK traffic (Fig. 4.8) illustrates this clearly. A natural next step is to replace the hand designed resolution cascade with learned policies, allowing each aircraft to adapt its trajectory to local traffic conditions rather than relying on pre computed alternates and fixed priority rules. **Multi Agent Reinforcement Learning (MARL) offers a principled framework for this extension.**

Multi Agent Reinforcement Learning formulation. The DAC airspace maps directly onto a cooperative MARL problem. Each aircraft becomes an *agent* that observes its local traffic environment and selects a continuous control action at every decision step. The observation vector o_i for aircraft i contains the relative positions, velocities, and headings of the N_a^* nearest neighbours, plus sector level state: the current conflict set ϕ, remaining fuel, and distance to the sector exit boundary. The action space is continuous: heading adjustment $\Delta\psi \in [-30°, +30°]$, speed modification $\Delta v \in [-10, +10]\,\%$, and vertical rate $\Delta\dot{h}$. This replaces the discrete holding/speed/denial cascade with graduated trajectory shaping. An aircraft can make a small heading correction early instead of committing to a full racetrack holding pattern late. The reward signal is a weighted composite of four terms: a large penalty for any separation violation (NMAC proximity), a fuel consumption cost proportional to path deviation, a time penalty to encourage efficient transit, and a completion bonus for successful sector traversal without conflict.

This formulation maps naturally onto *Multi Agent Deep Deterministic Policy Gradient* (MADDPG) [30], an actor critic algorithm that extends DDPG [31] to the multi agent setting. MADDPG follows a *centralized training, decentralized execution* (CTDE) approach: during training a single centralized critic $Q(s, a_1, \ldots, a_{N_a}; w)$ observes the global state and

the joint action vector, providing each agent's actor $\mu(o_i; \theta_i)$ with a stable learning signal. At deployment the critic is discarded and every aircraft runs its local actor network using only its own observation o_i. This is the same operational model as the current per aircraft consensus engine, but with neural networks replacing the Paxos state machine. Lamenza [32] recently demonstrated the MADDPG framework for decentralized UAV swarm coordination in sensor harvesting scenarios. With parameter sharing across homogeneous agents and a shared reward signal, collaborative behavior took place without explicit coordination rules: agents learned to partition the collection area among themselves and avoid redundant coverage, consistently outperforming a greedy nearest sensor heuristic as the swarm scaled from 2 to 8 UAVs.

Centralized training, decentralized execution. The CTDE approach fits the DAC architecture well. The current system already assumes that each aircraft runs its own consensus engine with local sector state; inter aircraft communication occurs only through protocol messages. In the MARL extension, the centralized critic is used exclusively during offline training in a simulator that mirrors the DAC environment: sectorized airspace, multi hop traversals, geometric conflict detection. Because the critic sees every agent's observation and action, it can learn the joint dynamics that a single agent's local view cannot capture: for example, that two aircraft approaching the same sector boundary from opposite directions should yield in complementary ways. After training, each aircraft carries a lightweight actor network ($\sim 10^4$ parameters for a 'two hidden layer' MLP) that runs inference in microseconds, well within the decision cadence of the consensus protocol's timeout intervals. This mirrors the progression observed in Lamenza's [32] experiments, where the centralized critic enabled stable multi agent training even when decentralized heuristic baselines struggled with coordination.

Advantages over the rules based protocol. Three properties make MARL attractive as a successor to the current system. First, *adaptivity*: a learned policy can discover traffic aware separation strategies that a uniform grid cannot. The JFK sweep results show that the optimal grid size for minimizing NMACs depends on route geometry (coarse grids win

when routes converge; fine grids win when traffic is uniform), but the current protocol has no mechanism to exploit this structure. A trained actor network could implicitly learn to adjust its behavior based on local traffic patterns, effectively performing route aware conflict resolution. Second, *smooth control*: the continuous action space allows graduated corrections. In the current protocol, an aircraft that cannot find a conflict free trajectory through its sector must commit to a full racetrack holding pattern: 14 waypoints, multiple laps, significant fuel expenditure. A learned policy could instead apply a slight heading change or a 5% speed reduction seconds earlier, resolving the same conflict with far less deviation. Third, *emergent coordination*: Lamenza's results demonstrate that MADDPG agents learn to partition space and develop implicit right of way conventions through the shared reward signal alone, without any explicit role assignment or priority rules.

Challenges and open problems. Despite its promise, applying MARL to safety critical airspace management introduces significant challenges. The most fundamental is *safety certification*: aviation regulators require deterministic worst case separation guarantees, whereas RL policies are probabilistic by nature. A near miss that occurs in even 0.1% of episodes is unacceptable in operational airspace. Constrained RL formulations [33] and runtime safety shielding are active research directions that could close this gap. In a shielded system, a verified fallback controller overrides the learned policy whenever a separation constraint is about to be violated.

A second challenge is *variable agent counts*. The centralized critic in MADDPG requires a fixed input dimension determined by the number of agents. Aircraft continuously enter and exit sectors, changing the population at every decision step. Lamenza [32] explored zero padding and phantom agents to handle variable counts but found that performance degrades significantly when the actual count deviates from the training distribution. Attention based critic architectures and graph neural networks, which accept variable size inputs by design, are a promising direction for the DAC setting where sector populations can range from 1 to 20+ aircraft.

A third challenge is *training cost*. Lamenza reports around 7 days of GPU time on an NVIDIA RTX 3090 for scenarios with up to 8 agents over 50 million training steps. DAC

scenarios involve 10 to 160 aircraft across dozens of sectors; the state and action spaces are orders of magnitude larger. Curriculum training (starting with a handful of aircraft and progressively scaling up), transfer learning from low density to high density scenarios, and asynchronous training pipelines could help manage the computational burden.

Finally, the *"sim to real"* gap affects any learning based approach. The DAC simulation uses point mass aircraft dynamics, deterministic communication (fixed delays, no packet loss), and deterministic conflict detection. Policies trained in this environment may not transfer directly to operational ATC settings with sensor noise, communication latency, wind, and weather deviations. Domain randomization during training and fine tuning on higher fidelity simulators are common mitigations.

Toward a hybrid architecture. The most practical path forward may combine both approaches. Aircraft would follow learned MARL policies during normal operations, benefiting from adaptive conflict resolution. When the actor network's output falls outside a verified safe envelope, or when the traffic scenario deviates from the training distribution, the system would fall back to the Bayesian optimized Paxos consensus protocol for deterministic safety. This hybrid design keeps the adaptivity of learning for the common case and the hard guarantees of the rules based protocol for the edge cases that matter most.

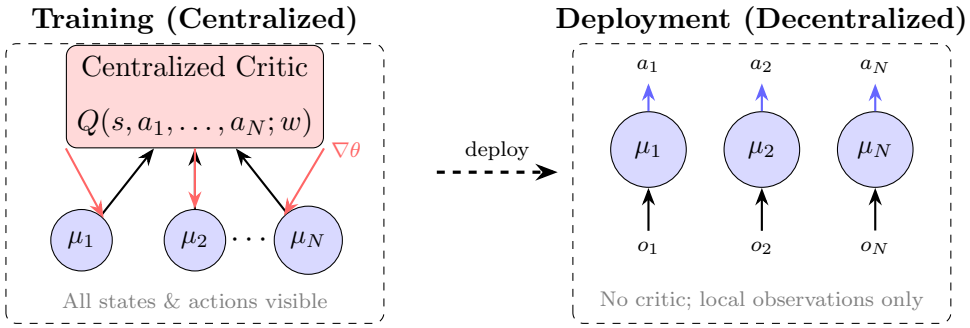


Fig. 7.1 Centralized training, decentralized execution (CTDE) for MARL-based conflict resolution. During training (left), the centralized critic observes all agents' states and actions. At deployment (right), each aircraft runs only its local actor network μ_i.

Adaptive sector sizing. The current pipeline predicts a grid once and holds it for the entire simulation. In practice, traffic patterns shift as aircraft enter and leave the airspace.

At cruise speeds of 450 to 500 knots, an aircraft covers roughly 15 nautical miles in two minutes and 23 nautical miles in three minutes across the 100 nautical mile study area. That is enough to meaningfully change the density, proximity, and flow direction features that drive the XGBoost prediction. A natural extension is to rerun the predictor every two to three minutes on updated traffic features and check whether the grid should change. The prediction itself takes milliseconds, so the computational cost is negligible.

The challenge is the transition. When the grid changes from, say, 3×3 to 4×4, every sector boundary moves and every aircraft needs to determine which new sector it belongs to. A practical approach would use hysteresis and a scheduled cutover. The system would require the new grid prediction to be stable across two or three consecutive checks before triggering a change, preventing oscillation when traffic sits near the boundary between two grid sizes. Once confirmed, the system would announce the new grid with a future activation time, allow all in-progress consensus rounds to complete during a short freeze window, and then have every aircraft adopt the new sector map and discover its new neighbors through the existing INIT_REQ discovery process. This mirrors how real air traffic control handles sector consolidation and splitting during shift changes. The detailed design of this transition protocol remains open work.

Inter-sector boundary coordination. The current protocol treats each sector as a fully independent consensus zone. This is by design: it maps cleanly onto the single decree Paxos structure and keeps the protocol simple, since each sector's occupants only need to agree among themselves. For most traffic scenarios this works well. Entry and exit consensus already catch the majority of boundary conflicts because an aircraft cannot enter a new sector without being checked against everyone inside, and the 60 second boundary lookahead triggers that process before the aircraft physically crosses. The sectorization scoring function also penalizes grid configurations that split dense traffic corridors, which reduces the likelihood of heavy traffic along shared boundaries in the first place.

That said, there is a gap. Two aircraft that are already admitted to adjacent sectors and flying parallel to the shared boundary could be within NMAC range of each other, and neither sector's protocol would detect it. Neither aircraft is crossing the boundary, so neither

triggers entry or exit consensus, and neither sector has visibility into the other's traffic. This scenario is narrow but real, and it becomes more likely as traffic density increases near sector boundaries.

A practical extension would be a lightweight boundary monitoring layer. Aircraft within a configurable distance of a sector boundary would periodically broadcast their position to members of the adjacent sector using ADS-B. Those broadcasts would trigger pairwise conflict checks against the neighboring sector's admitted plans, without requiring a full consensus round. If a conflict is detected, the affected aircraft could initiate a speed adjustment or a preemptive sector transition. This would add cross sector situational awareness without changing the core consensus protocol.

Learned objective weights. The weights in the BO objective function (100, 30, 10, 2) were set by intuition. An alternative is to learn them from data. One approach is bilevel optimization [34] where the outer loop adjusts the weights and the inner loop runs BO with those weights, then evaluates the resulting configurations against a ground truth ranking of what good protocol behavior looks like. Another option is inverse reinforcement learning [35], where the model observes configurations that domain experts consider good and infers a reward function that reproduces those preferences. Either approach would replace hand tuned weights with data driven ones and could reveal that the relative importance of success rate, holding, speed modifications, and retries differs from what intuition suggests.

Communication model extensions. The simulation currently uses fixed, uniform message delays with no packet loss. A natural extension is to test the consensus protocol under a *partially synchronous* network model: message delays drawn from empirical ADS-B latency distributions, configurable packet loss rates, and occasional network partitions. As discussed in §4, Paxos safety holds under these conditions, so the protocol would not produce conflicting admissions. The open question is how much throughput degrades as communication quality decreases. Degradation curves (success rate and BO score as a function of loss rate and delay variance) would quantify the margin between the current ideal-case results and realistic operating conditions. The Bayesian Optimization framework can then be reused

to retune protocol parameters for each communication profile, just as it currently tunes for each airport's traffic density. Longer term, integration of unmanned systems may require Byzantine fault tolerant protocol variants, since UAS lack the transponder-based identity guarantees of manned aviation.

Why latency alone is not the bottleneck. The BO tuned timeout window ranges from 0.5 to 5.0 seconds. Typical ADS-B message latency is on the order of 120 milliseconds. That is one to two orders of magnitude of margin. Introducing realistic latency with jitter would not meaningfully stress the protocol because rounds would still complete well within the timeout window. The more valuable experiment is packet loss. If 5 percent of messages are dropped, do retry counts spike? Does the entry success rate hold above 90 percent? Does the BO tuned backoff window still stagger proposals effectively, or do collisions increase? We have not yet measured how throughput degrades as a function of loss rate. Running the same BO campaigns under lossy conditions would produce degradation curves that quantify the gap between ideal and realistic performance.

Ground server coordination. The current protocol is purely air to air. A practical extension is to introduce ground servers as coordination points. Multiple ground stations distributed across a region could assist with heavier computation such as conflict resolution or state aggregation, reducing the burden on individual aircraft and allowing faster consensus rounds in dense traffic.

The key advantage of the current decentralized design is that it does not depend on ground infrastructure. Over oceans, polar routes, or remote regions where no ground station exists, aircraft use the pure Paxos protocol and coordinate among themselves directly. If a ground server fails, the protocol does not break. Aircraft continue with the same consensus mechanism as if no ground support existed. The ground server is an optimization, not a requirement for correctness or safety.

8 Conclusion

This thesis presented a four stage pipeline for adaptive airspace sectorization and decentralized admission control that operates without requiring per sector human controllers.

The first contribution is a replay simulation that ingests real FAA SWIM trajectories and evaluates candidate sector partitions on safety, efficiency, and density balance, producing labeled datasets for supervised learning. The second is a two stage XGBoost predictor that determines whether sectorization is needed and selects the optimal grid configuration from traffic features alone, without rerunning the simulation at inference time. The third is a reimplementation of the Paxos based DATC consensus protocol as a standalone per aircraft engine architecture with fuel aware planning, random backoff for livelock prevention, and bounded retry safety valves. The fourth is a Bayesian Optimization loop with a Gaussian Process surrogate that automatically tunes the protocol for each operating environment, removing the need for manual parameter selection.

The system has real limitations: straight line flight paths between fix points, point mass dynamics, deterministic communication, no terrain modeling, and single airport scenarios. These constrain the traffic modeling layer, not the coordination mechanism. The protocol and optimizer generalize to arbitrary waypoint geometries and would handle curved procedures or multi facility coordination without modification.

The main contribution is the framework itself: a testbed that connects real traffic replay, machine learned sectorization, consensus based coordination, and automated tuning into a closed loop. This provides a baseline for future extensions, including the multi agent reinforcement learning formulation described in Section 7, and a step toward reducing controller workload through decentralized coordination while maintaining separation safety.

Deploying a system like this in real airspace would follow the same incremental path the FAA uses for any new technology. This thesis sits at the simulation and testing stage. The next steps would be operational testing in low risk airspace such as oceanic routes or UAS corridors, then advisory mode where the system suggests and a human decides, then supervised autonomy where the system acts and a human monitors. NASA's Advanced Air Mobility initiative may accelerate this timeline because the projected density of drones and

air taxis will exceed what human controllers can manage today. For constrained domains, meaningful operational use could be realistic within 10 to 15 years. Dense terminal airspace with passengers would take longer. The architecture is designed so that each component can be independently improved and validated as the technology matures.

REFERENCES

[1] U.S. Department of Transportation, Office of Inspector General, "FAA faces controller staffing challenges as air traffic operations return to pre-pandemic levels at critical facilities," DOT OIG, Tech. Rep. AV2023035, Jun. 2023. Accessed: Sep. 7, 2025. [Online]. Available: https://www.oig.dot.gov/library-item/37874.

[2] Federal Aviation Administration, "Air traffic controller workforce plan, FY 2025–FY 2028," FAA, Tech. Rep., 2024. Accessed: Sep. 7, 2025. [Online]. Available: https://www.faa.gov/about/plans_reports/congress/air-traffic-controller-workforce-plan-2025-2028.

[3] EUROCONTROL, "Network operations report 2023," EUROCONTROL, Tech. Rep., 2024. Accessed: Sep. 7, 2025. [Online]. Available: https://www.eurocontrol.int/publication/annual-network-operations-report-2023.

[4] National Transportation Safety Board, "Runway incursion at Austin-Bergstrom international airport: Investigative docket materials," NTSB, Tech. Rep., 2023. Accessed: Sep. 7, 2025. [Online]. Available: https://data.ntsb.gov/Docket?ProjectID=106680.

[5] *Air traffic control (JO 7110.65z), change 2*, Federal Aviation Administration, May 2022. Accessed: Sep. 18, 2025. [Online]. Available: https://www.faa.gov/air_traffic/publications/atpubs/atc_html/.

[6] I. V. Laudeman, S. G. Shelden, R. Branstrom, and C. R. Brasil, "Dynamic density: An air traffic management metric," NASA Ames Research Center, NASA Technical Memorandum NASA TM-1998-112226, 1998. Accessed: Sep. 18, 2025. [Online]. Available: https://ntrs.nasa.gov/citations/19980210764.

[7] EUROCONTROL Experimental Centre, "Cognitive complexity in air traffic control: A literature review," EUROCONTROL, Tech. Rep. EEC Note 04/03, 2003. Accessed: Sep. 18, 2025. [Online]. Available: https://www.eurocontrol.int/publication/cognitive-complexity-air-traffic-control-literature-review.

[8] P. Flener and J. Pearson, "Automatic airspace sectorisation: A survey," *Knowl. Eng. Rev.*, vol. 28, no. 3, pp. 293–314, 2013. DOI: 10.1017/S0269888913000061. Accessed: Sep. 9, 2025.

[9] P. Kopardekar, K. Bilimoria, and B. Sridhar, "Initial concepts for dynamic airspace configuration," in *AIAA Aviation Technol., Integr., Oper. Conf. (ATIO)*, AIAA, 2007. DOI: 10.2514/6.2007-7778. Accessed: Sep. 18, 2025. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2007-7778.

[10] Federal Aviation Administration. "System wide information management (SWIM) — overview," Accessed: Oct. 5, 2025. [Online]. Available: https://www.faa.gov/air_traffic/technology/swim.

[11] FIXM Community. "Flight information exchange model (FIXM) — specifications," Accessed: Oct. 5, 2025. [Online]. Available: https://fixm.aero/.

[12] Federal Aviation Administration. "NASR subscriber file — aeronautical data," Accessed: Oct. 5, 2025. [Online]. Available: https://www.faa.gov/air_traffic/flight_info/aeronav/aero_data/NASR_Subscription.

[13] Federal Aviation Administration. "Coded instrument flight procedures FAACIFP18," Accessed: Oct. 5, 2025. [Online]. Available: https://www.faa.gov/air_traffic/flight_info/aeronav/digital_products/cifp.

[14] Federal Aviation Administration. "Digital products — aeronautical information services (CIFP/FAACIFP18)," Accessed: Oct. 5, 2025. [Online]. Available: https://www.faa.gov/air_traffic/flight_info/aeronav/digital_products.

[15] S. Paul, S. Patterson, and C. A. Varela, "Conflict-aware flight planning for avoiding near mid-air collisions," in *38th AIAA/IEEE Digit. Avionics Syst. Conf. (DASC)*, San Diego, CA, 2019. Accessed: Oct. 20, 2025. [Online]. Available: http://wcl.cs.rpi.edu/papers/DASC2019_paul.pdf.

[16] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" In *Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 507–520.

[17] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[18] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[19] S. Paul, S. Patterson, and C. A. Varela, "Collaborative situational awareness for conflict-aware flight planning," in *2020 AIAA/IEEE 39th Digit. Avionics Syst. Conf. (DASC)*, San Antonio, TX, USA, 2020, pp. 1–10. DOI: 10.1109/DASC50938.2020.9256620.

[20] S. Paul et al., "Formal verification of safety-critical aerospace systems," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 38, no. 5, pp. 72–88, 2023. DOI: 10.1109/MAES.2023.3238378.

[21] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998. DOI: 10.1145/279227.279229.

[22] S. Paul, G. A. Agha, S. Patterson, and C. A. Varela, "Eventual consensus in synod: Verification using a failure-aware actor model," *Innov. Syst. Softw. Eng.*, 2022. DOI: 10.1007/s11334-022-00463-5. Accessed: Nov. 12, 2025. [Online]. Available: http://wcl.cs.rpi.edu/papers/paul-ISSE-2022.pdf.

[23] S. Paul, C. McCarthy, S. Patterson, and C. Varela, "Formal verification of timely knowledge propagation in airborne networks," *Sci. Comput. Program.*, vol. 239, 2025. DOI: 10.1016/j.scico.2024.103184. Accessed: Nov. 12, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642324001072.

[24] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[25] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001. DOI: 10.1162/106365601750190398.

[26] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[27] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.

[28] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Global Optim.*, vol. 13, no. 4, pp. 455–492, 1998.

[29] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[30] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6379–6390.

[31] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. 4th Int. Conf. Learn. Representations (ICLR)*, 2016.

[32] T. d. S. Lamenza, "A deep reinforcement learning approach for sensor data collection by a swarm of UAVs," M.S. thesis, Pontifícia Univ. Católica do Rio de Janeiro (PUC-Rio), Apr. 2025.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd. MIT Press, 2018.

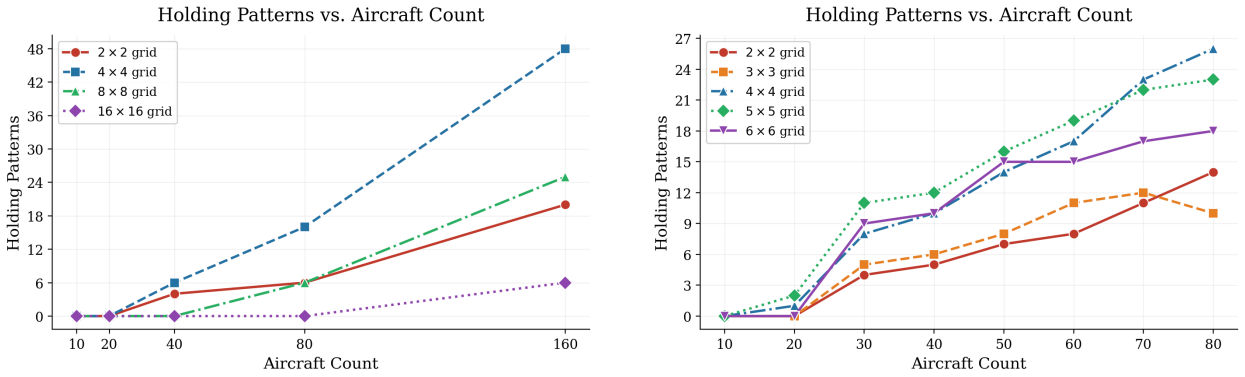
[34] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Ann. Oper. Res.*, vol. 153, pp. 235–256, 2007. DOI: 10.1007/s10479-007-0176-2.

[35] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. 17th Int. Conf. Mach. Learn. (ICML)*, 2000, pp. 663–670.

9 Appendices

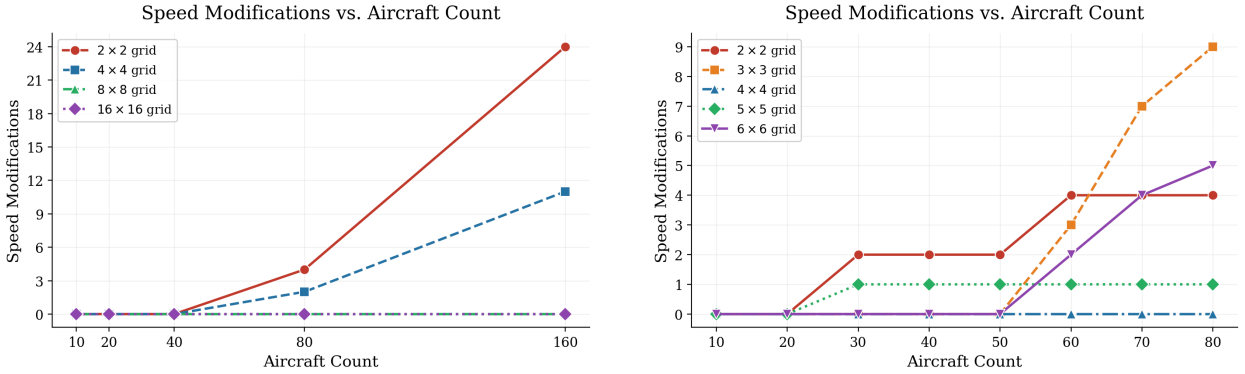
9.1 Additional Sweep Results

The following figures supplement the main results presented in §4.11.5. Each figure pairs the synthetic sweep (left) with the JFK real world sweep (right) for a single metric.



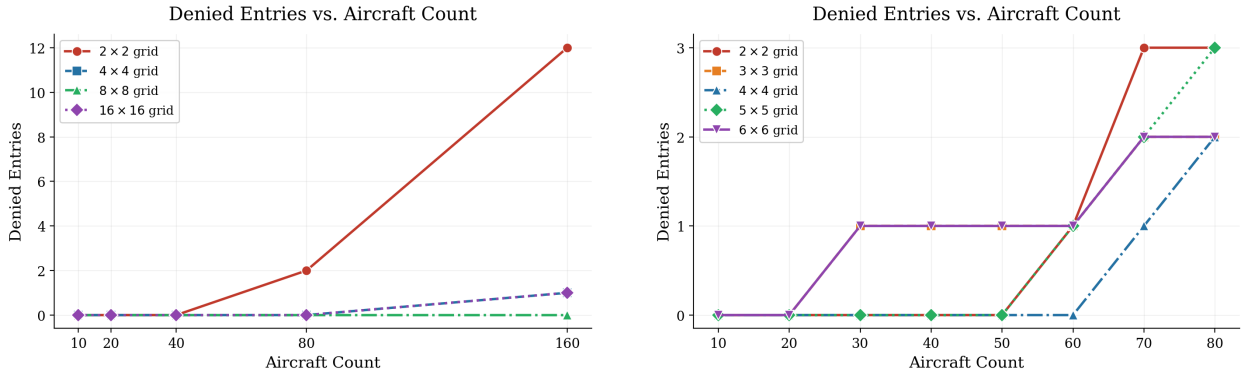
(a) Synthetic sweep **(b) JFK real world sweep**

Fig. 9.1 Holding patterns assigned vs. aircraft count.



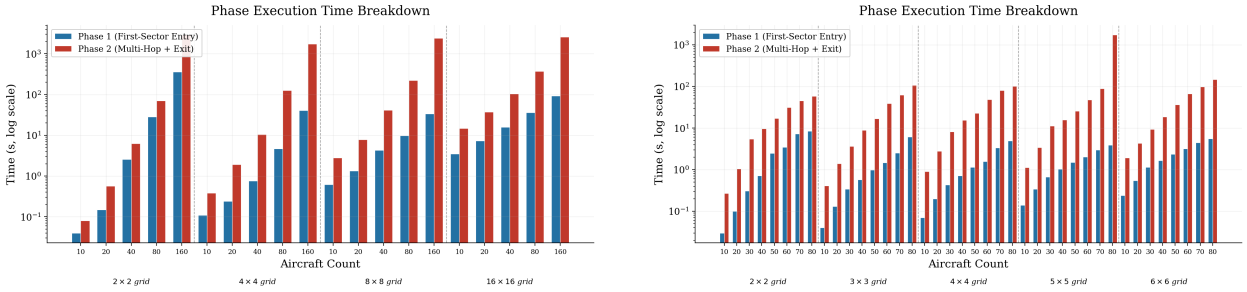
(a) Synthetic sweep **(b) JFK real world sweep**

Fig. 9.2 Speed modifications vs. aircraft count.



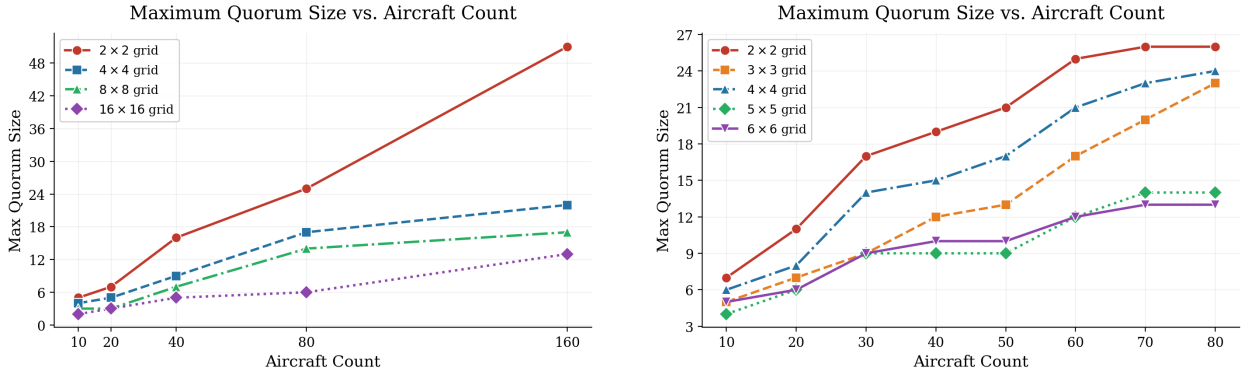
(a) Synthetic sweep (b) JFK real world sweep

Fig. 9.3 Denied entries vs. aircraft count.



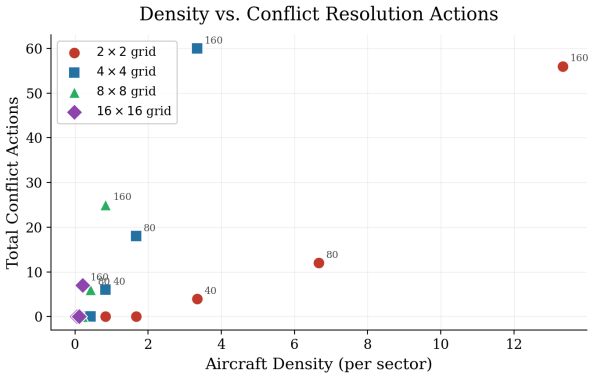
(a) Synthetic sweep (b) JFK real world sweep

Fig. 9.4 Phase 1 vs. Phase 2 execution time breakdown.

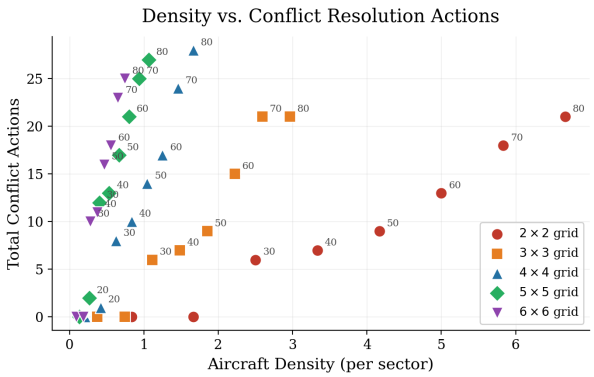


(a) Synthetic sweep (b) JFK real world sweep

Fig. 9.5 Maximum quorum size vs. aircraft count.

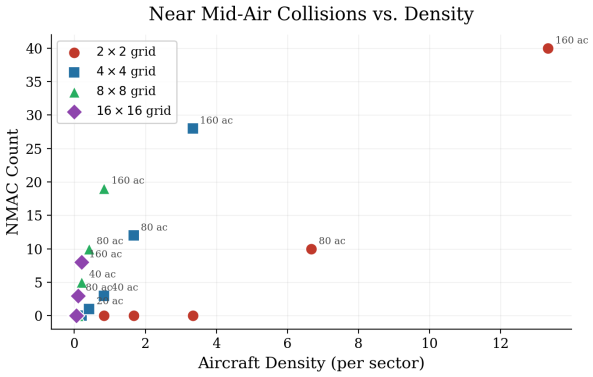


(a) Synthetic sweep

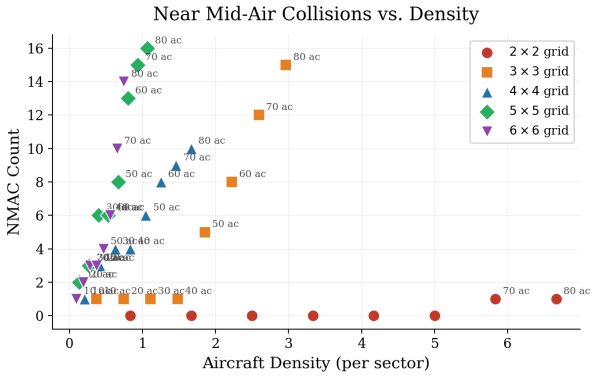


(b) JFK real world sweep

Fig. 9.6 Aircraft density (per sector) vs. total conflict resolution actions.



(a) Synthetic sweep



(b) JFK real world sweep

Fig. 9.7 Near mid air collisions vs. aircraft density (per sector).