tangram in Action

Practical Use Cases for Real-time Open Aviation Research

Xavier Olive*, Yan Lok Cheung†, Junzi Sun‡

*ONERA – DTIS
Université de Toulouse
Toulouse, France

†Department of Mechanical and Aerospace Engineering
The Hong Kong University of Science and Technology
Hong Kong

‡Faculty of Aerospace Engineering
Delft University of Technology
Delft, the Netherlands

***Abstract*—tangram is an open research framework for real-time processing of high-throughput geospatial surveillance streams, with a primary application in ADS-B and Mode S surveillance data. While large-scale historical datasets have motivated extensive aviation research, the transfer of methods and algorithms to live data streams remains less documented. This transfer is significantly more challenging: real-time analyses are difficult to implement, debug, and reproduce. tangram addresses this gap by providing a modular and extensible platform that lowers the technical barriers for researchers, enabling them to integrate their own algorithms without developing a full streaming infrastructure, so that they can focus on addressing their own research questions. This paper introduces the architecture of the platform and demonstrates its flexibility through four representative use cases in air traffic management: integrating weather forecasts, estimating fuel flow, analysing contrail formation, and monitoring airport performance. Together, these examples illustrate how tangram enables reproducible, real-time experimentation and opens new perspectives for operationally relevant research in aviation.**

***Keywords*—open aviation, open data, real-time analyses, ADS-B**

I. Introduction

Aviation research has benefited from the increasing availability of large-scale historical data sources [1] and datasets [2], [3]. These datasets have enabled the development and validation of methods across areas such as air traffic performance [4], safety analysis [5], and environmental impact assessment [6]. However, while historical analyses provide valuable insights, they only capture a static view of the past. Many operational challenges in air traffic management are inherently dynamic and time-critical, requiring methods to be tested and deployed in real-time contexts rather than in a post-operational setting. Examples include monitoring airport capacity as demand evolves, detecting unstable approaches during landing, or predicting contrail formation to prepare climate-conscious operations.

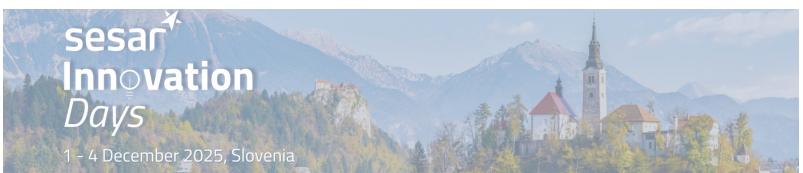
Working with real-time surveillance data introduces new challenges. Data streams are inherently noisy and incomplete, making it difficult to rely on techniques that assume access to the full temporal context, such as filters or smoothers that depend on future observations [7]. Furthermore, unexpected data gaps, latency, and variations in sensor coverage all complicate the design of algorithms meant to operate continuously in live environments. While operational systems in industry address some of these issues, their architectures are typically proprietary, leaving academic researchers without reproducible frameworks to test their methods with open data.

Additionally, access to live air traffic surveillance data further complicates the picture. Initiatives such as the OpenSky Network [1] have had a significant impact in democratizing access to aviation data, yet most openly available services remain restricted to historical archives or delayed streams. While unauthorized API querying (data scraping) from various providers is technically possible, it often violates terms of use and typically yields only minimal subsets of the information required for meaningful analysis (e.g., positions and speeds without richer surveillance metadata). As a result, there remains a gap between research and practice: methods can be validated retrospectively but are not easily transferable to operational settings without significant additional infrastructure.

Bridging this gap requires an integrated platform that supports the entire pipeline of real-time analysis, from raw data ingestion to filtering, transformation, computation, and visualisation. Developing such a platform from scratch is a significant engineering effort, often exceeding the time and resource constraints of typical academic projects. As a result, researchers frequently face a trade-off between advancing methodological innovation and building the technical infrastructure required to test those innovations in realistic environments.

Modularity can help lower this barrier: researchers should not need to master every aspect of distributed systems, streaming architectures, or visualisation frameworks to contribute new analytical methods and live demonstrations. A platform that abstracts these complexities and provides well-defined interfaces for extension enables researchers to focus on their domain expertise while still integrating seamlessly into a real-time pipeline. This approach is embodied in the *tangram platform* [8], an open research framework for ADS-B and Mode S flight surveillance data. The modularity of tangram further facilitates the incorporation of additional, potentially non-open data sources, provided that the necessary interfaces and access mechanisms are available within the hosting infrastructure.

The remainder of this paper is organized as follows. Section II presents the overall architecture of the proposed platform, highlighting the technologies involved and the design principles that support modularity and extensibility. Sections III through VI illustrate the application of the framework through four representative use cases: weather information, fuel flow estimation, contrail formation analysis, and airport performance monitoring. Finally, Section VII concludes and outlines ongoing and future directions.



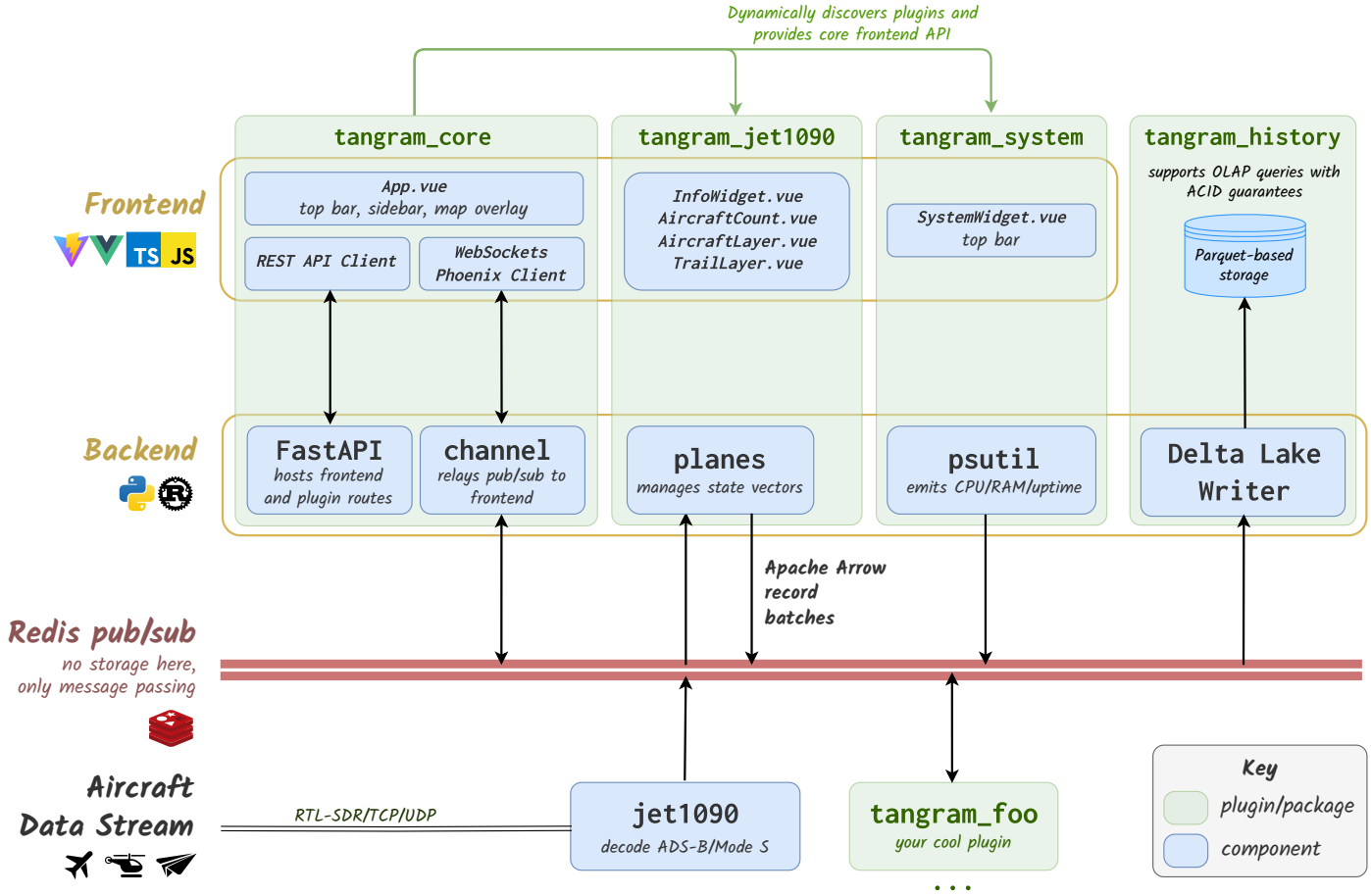


Figure 1. Overview of the tangram platform architecture, consisting of the backend responsible for data processing, a frontend for visualisation, with a high-performance message bus for inter-plugin communication. Plugins are `pip`-installable packages that can be independently developed and tested.

II. Architecture of the Platform

Figure 1 provides an overview of the architecture. The system is organized into three main layers: the backend services, the frontend interface, and the plugin ecosystem. Each component plays a distinct role, but they are tightly integrated to support extensibility, scalability, and ease of use.

A. Structure of the backend core

The backend is designed as a stable and extensible core upon which researchers can build and test their own real-time analyses. It runs as a single, asynchronous Python process initiated by the `tangram serve` command. This core process is intentionally minimal, providing the essential parts for web serving and data streaming but contains no inherent knowledge of aviation data. Its primary role is to discover, load, and orchestrate independent, installable packages known as *plugins* (Section II-B).

All components, including the core services and all active plugins, communicate asynchronously through a Redis [9] publish/subscribe message bus. This data-centric design decouples the modules from one another, allowing them to interact by producing and consuming structured data streams rather than through direct function calls. The `channel` component bridges this internal message bus to the frontend via WebSockets. It acts as a high-performance gateway, translating Redis messages into WebSockets events for the browser and vice-versa. This architecture extends the real-time, event-driven model directly to the user interface.

This bidirectional flow is fundamental to the interactivity of the platform. For data flowing from the backend to the frontend, a service like `planes` publishes updated aircraft state vectors to a client-specific Redis channel. The `channel` component, subscribed to these channels, receives the data and forwards it to the appropriate client over the WebSocket connection. Conversely, the frontend can send information back to the backend. For example, when a user pans or zooms the map, the frontend sends the new map bounding box coordinates along with a unique client identifier through the WebSocket. The `channel` component receives this message and publishes it to a Redis channel. The `planes` channel, listening on this channel, can then update its internal state to filter the aircraft data it sends, ensuring that each client only receives updates for the aircraft visible within their specific viewport.

This model, which decouples components through a shared, data-centric message bus, mirrors the robust node-and-topic architecture of established frameworks like the Robot Operating System (ROS 2). This design was chosen after initial, more monolithic prototypes proved difficult to extend and maintain.

By enforcing communication through the message bus, the system ensures that plugins can be developed, tested, and even fail in isolation.

B. Backend plugins

Plugins provide the primary mechanism for extending tangram beyond its minimal core. The platform is intentionally designed as a lightweight engine, complemented by optional extensions that can be activated on demand. Plugins are self-contained units of code that may exist in separate repositories, with their own dependencies and test suites. This extensibility is realized through distinct mechanisms for the backend and frontend.

A plugin is a standard Python package that the framework discovers at runtime by scanning for an entry point in the package's metadata. This approach enables a "plug-and-play" workflow where new capabilities can be integrated simply by installing the package and enabling it in a configuration file. As depicted in Figure 2, each plugin adheres to a clear contract by providing a central `Plugin` object. This object serves as a manifest, allowing the plugin to register custom REST API endpoints and long-running background services with the core application.

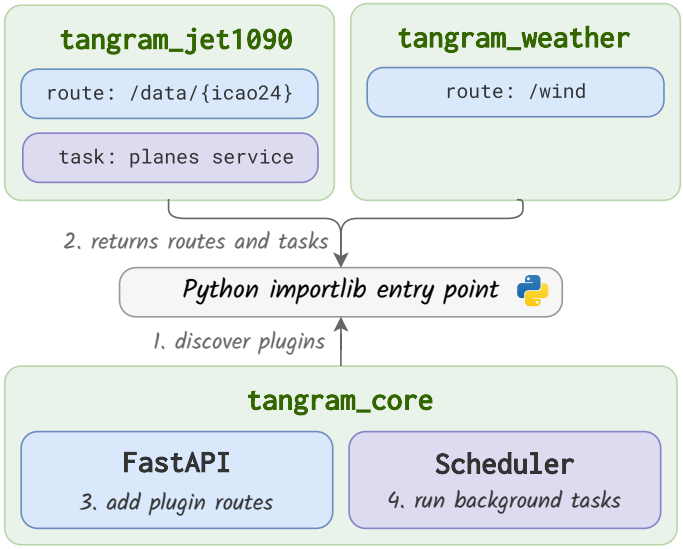


Figure 2. The backend plugin discovery and injection mechanism. The core application discovers installed plugins via Python's entry point system, dynamically integrates into its FastAPI server and asynchronous task scheduler.

To support users in extending the platform, an example-oriented tutorial is available in the documentation (https://mode-s.org/tangram/plugins), which focuses on the technical aspects of plugin implementation. In the following sections, we shift the perspective to applicative considerations and limitations, examining how typical research questions can be addressed when historical data are available, and how their treatment differs in a real-time implementation.

C. Structure of the frontend core

The frontend is built with Vue.js (https://vuejs.org/) and serves as the dynamic interface for real-time visualisation of aviation data. It mirrors the backend philosophy of a minimal core extended by plugins. The core tangram package serves a lightweight web application shell, defined in `App.vue`, which provides the main layout and application scaffolding. This layout consists of a top navigation bar for global controls and status widgets, a central map view powered by MapLibre GL and Deck.gl, and a sidebar area that typically displays detailed information for a selected entity. This core shell however, contains no domain-specific features itself.

D. Frontend plugins

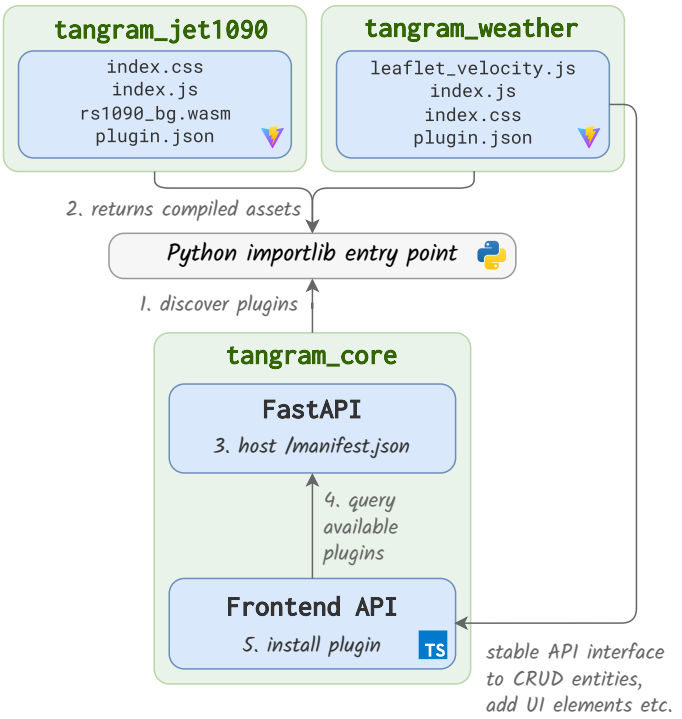


Figure 3. The frontend plugin loading sequence. The backend informs the frontend of the available plugins and registers each plugin through a stable frontend API.

As shown in Figure 3, the frontend discovers plugins through a dynamic loading mechanism. The core web application first queries a `/manifest.json` endpoint served by the backend. This manifest, generated at startup, lists all enabled Python plugins that also provide a frontend component. For each entry in the manifest, the application dynamically imports the JavaScript entry point.

This entry point executes an `install` function, which receives a `TangramApi` object. The API acts as the formal contract for frontend extensions, providing methods to register their Vue components as widgets in predefined UI locations, such as the `TopBar` or `Sidebar`. The core application's template then dynamically renders these registered components in the appropriate slots. This decoupled design allows researchers to share custom UI components as part of their installable packages, ensuring that both backend logic and frontend visualisation are bundled and deployed together.

E. System monitoring plugin

The power of this architecture is best illustrated by examining the optional plugins distributed with the platform. As a straightforward example, the `tangram_system` plugin demonstrates the basic pattern of a backend service driving a frontend widget. It registers a continuous background task that monitors the health of the host system, collecting metrics such as CPU load, memory usage, and uptime. These metrics are periodically published to a dedicated Redis channel, providing users with immediate feedback on the platform's operational status via a simple component in the top navigation bar.

F. Aircraft tracking plugin

A more comprehensive example is the `tangram_jet1090` plugin, which encapsulates all aviation surveillance-specific functionality. The data pipeline begins with an external `jet1090` process (see https://mode-s.org/jet1090), an independent, high-performance decoder for ADS-B and Extended Mode S messages. As a modern successor to the widely used `dump1090`, `jet1090` aggregates data from local RTL-SDR devices or network streams and publishes raw, decoded messages to a Redis channel. The `tangram_jet1090` plugin subscribes to this stream to derive higher-order state information. This plugin encapsulates the aviation-specific logic, isolating it from the generic core.

This plugin orchestrates the transition from stateless raw messages to stateful aircraft tracks. It consists of a high-performance Rust service, `planes`, exposed to Python, which manages three critical tasks: live state maintenance, client-specific streaming, and an optional historical data production.

1) Live State and Eviction: The `planes` service maintains an in-memory table of aircraft state vectors. As raw messages arrive, they are reconciled with existing entities; late or out-of-order packets update the state if they provide newer timestamps, ensuring eventual consistency. A periodic cleanup task enforces a retention policy defined by a user-configurable expiration parameter, evicting aircraft that have ceased broadcasting to ensure that the live view remains strictly real-time.

2) View-Dependent Streaming: To optimize frontend performance, the service implements server-side filtering. The frontend periodically transmits its viewport coordinates to the backend. The `planes` service iterates through the active state vectors, performs a containment check and publishes a filtered subset of aircraft to a unique, client-specific Redis channel. This ensures clients only consume bandwidth for visible entities.

3) Persistence via Delta Lake: For historical analysis, `tangram_jet1090` acts as a producer for the `tangram_history` plugin. To handle high-throughput surveillance data without blocking the real-time event loop, the system employs a decoupled ingestion strategy. State vectors are serialized into Apache Arrow record batches and pushed to a Redis Stream. A separate consumer service within the `tangram_history` plugin reads these batches asynchronously and persists them to partitioned Delta Lake tables. This architecture ensures backpressure protection: if the persistence layer I/O latency, the Redis Stream buffers incoming data, preserving the integrity of the live tracking service.

Historical data is accessed via a REST API endpoint registered by the `tangram_jet1090` plugin. This endpoint uses the Polars dataframe library to query the Delta Lake tables managed by the history service, allowing for efficient retrieval of specific trajectories.

Historical data is accessed not through a separate service, but via a REST API endpoint registered by the plugin, which directly queries the Delta Lake Table populated by the `tangram_history` plugin.

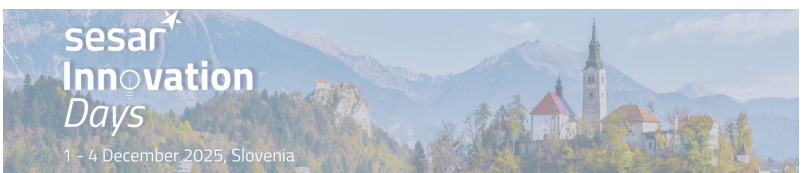
III. Use case 1: Weather information

Weather is a critical factor in air traffic management and trajectory analysis. While surveillance data provides accurate information on aircraft position and their derivatives, it does not include the surrounding meteorological conditions that strongly influence performance, efficiency, and environmental impact. For instance, ADS-B messages broadcast only ground speed and track angle; Enhanced Mode S may report additional parameters such as true airspeed and heading, but only in regions within range of Secondary Surveillance Radars parameterized to query such information. Therefore, exogenous meteorological data are necessary to understand and model aircraft behaviour in its operational context.

Wind estimation has long been a research focus, with several studies proposing to infer wind vectors directly from ADS-B [10] or Mode S data [11]. However, these methods typically rely on statistical inference and are limited to areas with sufficient traffic density. An alternative approach is to integrate external meteorological predictions, such as numerical weather models, into the analysis. This approach is particularly suited to real-time pipelines, where predictive weather products (e.g., Météo France ARPEGE forecasts) are one of the practical data sources available. By contrast, most historical studies rely on reanalysis products such as ERA5, which are not suitable for real-time operational contexts.

In tangram, weather information is integrated via a plugin architecture that couples backend access to meteorological forecasts with frontend visualisation tools. On the backend, the plugin retrieves GRIB files from the ARPEGE model provided by Météo France through the `data.gouv.fr` initiative, which publishes new forecasts every six hours for a three-day horizon. The plugin exposes an API endpoint that serves wind vector fields at user-specified isobaric levels, using efficient data handling via `xarray` (loading GRIB file) and `orjson` (encoding the data for Web transfer). A caching mechanism ensures that previously downloaded files are reused when possible, falling back to earlier forecasts if newer products are not yet available.

On the frontend, the plugin provides a dedicated Vue component that enables interactive visualisation of wind fields. The interface includes a slider to select the desired isobaric level, mapped to a corresponding flight level, and displays the associated wind vectors as a velocity field overlaid on the map. The visualisation leverages the `weatherlayers-gl` library, adapted to integrate with the Vue-based frontend and the Deck.gl map context. Figure 4 illustrates an example of this functionality, where predicted winds are displayed alongside real-time trajectories. This combined view allows researchers to



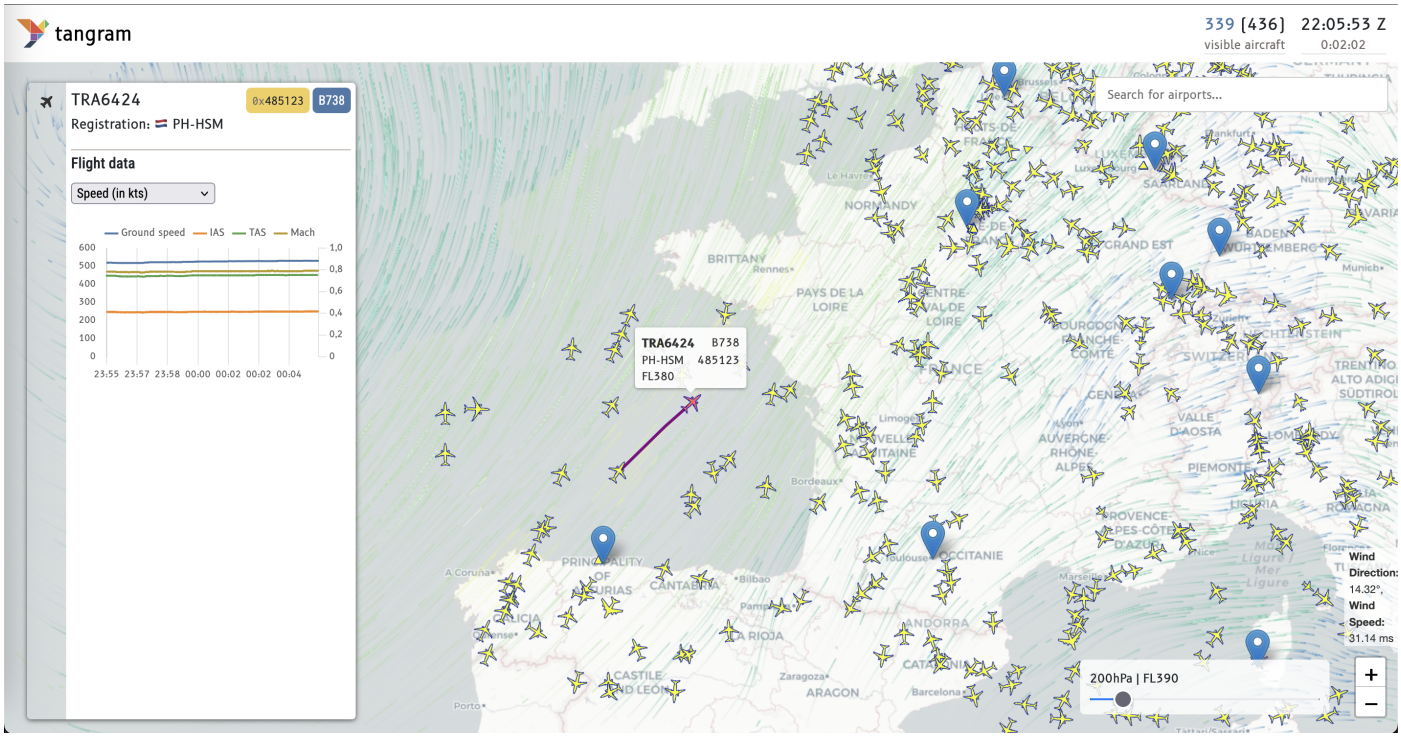


Figure 4. Visualisation of a forecast wind field overlaid with real-time aircraft trajectories. Note that the wind field that is displayed for TRA6424 is consistent with the ground speed and true airspeed measured by the aircraft.

relate observed aircraft performance to forecasted atmospheric conditions, and provides a foundation for further analyses such as fuel-flow estimation (Section IV) and contrail prediction (Section V).

Note that the wind field that is displayed for TRA6424 is consistent with the ground speed and true airspeed measured by the aircraft: 30 m/s (displayed in the lower right corner of the map) roughly corresponds to 60 kts, which is compatible with the delta in the speed values in the plot. Also, a higher ground speed value is observed when the aircraft is flying with a strong tailwind.

IV. Use Case 2: Fuel Flow Estimation

Fuel flow is a key metric for evaluating both aircraft performance and environmental impact. Its estimation relies on many external parameters, including aircraft type and engine characteristics, take-off mass, altitude, airspeed, and flight phase. ADS-B broadcasts ground speed and track angle information, while Enhanced Mode S can include true airspeed and heading when available. A reliable estimation of fuel consumption requires integrating the true airspeed with aircraft performance models: in this respect, meteorological information (Section III) is of great value, as true airspeed can be approximated by correcting ground speed with wind forecasts when Enhanced Mode S is unavailable. Simplified models may also estimate the fuel flow by assuming no wind, using ground speed in place of the true airspeed.

Several modelling tools are available for estimating the fuel flow. The Open Aircraft Performance Database (OpenAP) [12] offers modular implementations for aerodynamic drag, engine thrust, and fuel flow models, encompassing a diverse range of commercial aircraft types. It builds upon both data-driven models and calibration against reference datasets. Complementary tools such as Acropole [13] provide high-fidelity models for specific aircraft types, which OpenAP generalizes to a broader fleet. These models enable researchers to compute instantaneous fuel flow based on mass, true airspeed, altitude, and rate of climb or descent, and can further be linked to emission models for environmental assessments.

When working with historical datasets, integration is relatively straightforward. Libraries such as `traffic` [14] provide direct interfaces to OpenAP, allowing researchers to reconstruct flight phases, filter trajectories, and apply suitable performance models. Prior to this step, preprocessing is often required: common filtering techniques [7] rely on rolling windows, which exploit both past and future observations to smooth the measurements. These methods are efficient in retrospective analyses but cannot be directly applied to real-time data streams. In online applications, the problem becomes more challenging. Kalman filters are well-suited for real-time estimation, as they only depend on past and current states, whereas smoothing techniques are no longer applicable. An intermediate solution is to operate in a *delayed real-time* mode: estimates are produced continuously but can also be revised retroactively as new observations are received, improving the accuracy of the most recent segment of the trajectory.

The architecture of a fuel flow estimation plugin in tangram depends on the intended use:

- A *backend implementation* can subscribe to Redis channels carrying aircraft state vectors, apply preprocessing filters,

OpenAP's estimation functions in real time, and maintain a history of per-aircraft fuel consumption. Current estimates can either be broadcast to all clients (for continuous monitoring) or served on demand via the REST API when a user selects a specific aircraft. By leveraging the `tangram_history` plugin's architecture, computed fuel flow records can be appended to the Delta Lake tables alongside trajectory data, allowing for efficient querying of historical fuel consumption via the REST API. The frontend is only used for visualisation thanks to another Vue component.

- A *full frontend implementation* is also possible: this approach reduces backend load but increases client-side complexity. This requires some upstream work (in progress) to port filtering methods and OpenAP functionality to code that is interpretable by the browser: options include vanilla Javascript (or Typescript) implementations, transpilation of existing Python code to WebAssembly (PyScript) or porting Python code to a third-party language that can easily be wrapped for both Python and WebAssembly: Rust is the most natural candidate for this last option.

tangram's modular design supports both options, with plugins that can be implemented at both the backend and frontend levels. Both strategies illustrate how tangram's modular design supports different trade-offs between scalability and accuracy. In either case, integrating fuel flow estimation into real-time visualisation provides immediate insight into the operational efficiency of flights and creates the basis for higher-level analyses, such as emissions monitoring or contrail prediction.

V. Use Case 3: Real-time Contrail Formation Estimation

Aircraft-induced contrails are a major contributor to aviation's climate impact, potentially comparable to the warming effect of carbon emissions. Mitigating this impact requires identifying and avoiding atmospheric regions where persistent contrails are likely to form. Since these regions are often thin and geographically localized, minor and fuel-efficient adjustments to flight altitude can often prevent contrail formation. This makes real-time prediction an essential component of any climate-optimized flight operations strategy.

Retrospective studies typically analyse contrail formation by combining historical flight data with high-fidelity reanalysis weather datasets like ERA5. An operational tool, however, must integrate live aircraft surveillance data with predictive weather models. The core of such a use case is the Schmidt–Appleman criterion (SAC), a physical model that determines if ambient atmospheric conditions are suitable for contrail formation, given an aircraft's emissions and engine performance. A contrail, once formed, will persist and spread if the aircraft is flying through an ice-supersaturated region (ISSR), which is also determined from the forecast weather data.

We developed a preliminary contrail tangram plugin to perform this analysis. The current prototype provides an on-demand historical analysis for a selected aircraft. The implementation consists of:

- A backend service that exposes a REST API endpoint. When a user selects an aircraft, this service queries the Delta Lake storage managed by `tangram_history` to retrieve the full trajectory. It then correlates these spacetime points with weather data using the *fastmeteo* library. The backend service computes the segments where both the SAC and ISSR conditions were met.
- A frontend component that calls this endpoint for the selected aircraft and visualizes the resulting contrail-forming segments as red segments overlaid on the aircraft's flight path. The component automatically refreshes the data periodically to update the visualisation as the flight progresses. An example is shown in Figure 5

Figure 5. Estimated contrail formation based on the live trajectory data for one example trajectory.

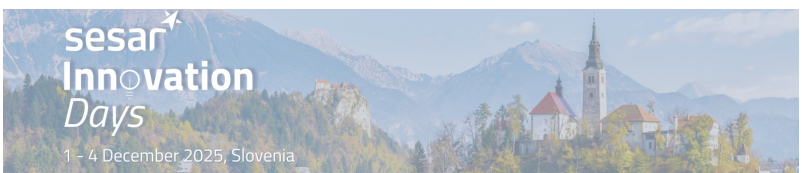
The final implementation will provide a real-time, airspace-wide implementation. In this configuration, the backend plugin will not be on-demand but will continuously:

- Subscribe to the live state vector stream for all aircraft.
- Integrate with the real-time ARPEGE weather forecast plugin (Section III) to obtain ambient conditions.
- Estimate aircraft engine parameters, leveraging models like OpenAP as discussed in the fuel flow use case (Section IV).
- Compute contrail formation status for each position update and publish the results to a dedicated Redis channel.

The frontend would then subscribe to this channel, enabling a live, dynamic visualisation of contrail formation across all flights in view. This transforms the system from a historical analysis tool into a live situational awareness platform for monitoring aviation's real-time climate impact.

VI. Use Case 4: Airport Monitoring

Airport movement analysis has long been a subject of research, and several of our previous studies have focused on the derivation of ADS-B-based milestones [15] and the design of performance metrics. In particular, we developed methods for filtering trajectories to match taxiway networks [7], as well as metrics to evaluate the operational performance of runways and specific taxiway segments [16].



In the present experiment, we illustrate how tangram can support airport monitoring by feeding real-time traffic data into a separate performance assessment dashboard. The prototype dashboard was implemented on the Observable platform, which is a web-based environment for creating interactive notebooks that combine JavaScript code, live data streams, and visualisation libraries in a reactive programming model. Observable automatically re-executes dependent cells whenever the underlying variables are updated, making it particularly well suited for real-time data applications.

A WebSocket interface was established between the dashboard and the tangram `channel` component. This setup reproduces the behaviour of a standard browser: a WebSocket connection is created, a bounding box is shared, and only aircraft located within that bounding box are streamed downstream. On the client side (Observable), additional filtering is applied in JavaScript to retain only aircraft within the airport footprint (as defined by OpenStreetMap) and at altitudes close to the ground. Each time new surveillance messages are received, these filters are applied, aggregated information is appended to a dataset, and the corresponding plots are automatically refreshed by the Observable runtime.

Using a receiver located at our facilities with line-of-sight to ground movements, we performed simple aggregations to monitor the number of aircraft taxiing or parked, as well as ground vehicles (e.g., follow-me, fire, police and customs, runway inspection), all of which broadcast messages with Downlink Format 18 (Figure 6). Such a setup provides a lightweight yet powerful means to generate live situational awareness dashboards for airport operations. Looking ahead, the same approach could be scaled to multiple airports or combined with A-CDM processes, enabling comparative studies of airport performance and supporting real-time decision making in collaborative environments.

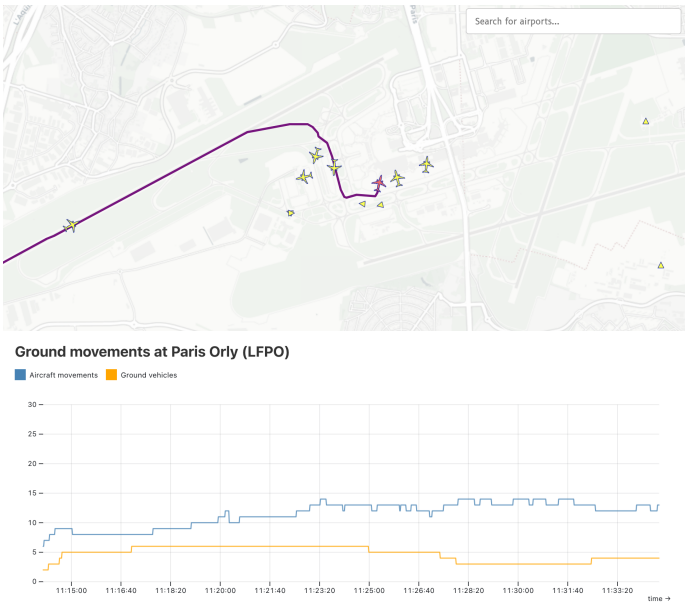


Figure 6. Live airport surface monitoring at Paris Orly (LFPO). (Top) Real-time positions of aircraft and ground vehicles; (Bottom) Aggregated occupancy counts.

VII. Conclusion and Future Work

This work presents tangram, an open research framework designed to bridge the gap between historical data analysis and real-time operational contexts in aviation. Through a series of representative use cases, from integrating weather forecasts to estimating contrail formation, we have illustrated how analytical methods can be adapted for live implementation. The key contribution of this work is a versatile platform that facilitates the exploration, refinement, and validation of algorithms on streaming surveillance data, thereby enabling more operationally relevant research.

Future development will proceed along two primary strategic directions. The first involves broadening the scope of demonstrable applications. The existing plugin architecture provides a robust foundation for integrating a wider range of analyses, including but not limited to: real-time separation monitoring [17], unstable approach detection [18], live airport performance analysis [4], and GNSS jamming detection [19], among others. These extensions will serve to further validate the platform's utility across diverse research domains.

The second direction focuses on extending the framework's application to simulated environments. By leveraging the Redis publish/subscribe interface, tangram can serve as a unifying visualisation frontend for various air traffic simulators, such as BlueSky [20] or AirTrafficSim [21]. This capability allows for the seamless application of existing real-time analysis modules to simulated traffic, enabling detailed what-if scenarios, and extend to further applications such as data filtering, metric computation, and airspace monitoring, to be applied seamlessly to simulated traffic. This line of work naturally extends to the emerging domain of Urban Air Mobility (UAM), where the platform can link strategic multi-agent mobility models and operational trajectory simulators. Future work in this area will prioritize the migration towards 3D rendering to adequately represent complex, low-altitude urban airspace.

Overall, tangram provides an open-access versatile platform for visualizing air traffic data alongside auxiliary information that can be retrieved or computed by the user, such as weather, fuel flow, contrail, or performance metrics. This capability offers researchers a unique perspective: methods and algorithms developed on historical datasets can now be tested, explored and refined in a real-time context, bridging the gap between retrospective analysis and live operational environments; for example, evaluating fuel flow or contrail formation as flights progress. Moreover, the platform's modular architecture allows components to be added or repositioned quickly, enabling experimentation and prototyping that would be challenging when relying on conventional data providers with limited access or constrained by institutional IT policies.

Reproducibility Statement

The code for the tangram platform is openly shared at:
https://github.com/open-aviation/tangram
Documentation is available at https://mode-s.org/tangram

REFERENCES

[1] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm, "Bringing up OpenSky: A large-scale ADS-B sensor network for research," in *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, pp. 83–94, 2014.

[2] M. Strohmeier, X. Olive, J. Lübbe, M. Schäfer, and V. Lenders, "Crowdsourced air traffic data from the OpenSky Network 2019–2020," *Earth System Science Data*, vol. 13, pp. 357–366, Feb. 2021.

[3] X. Olive, L. Basora, and J. Sun, "Arrival trajectories at five major European airports," 4TU.ResearchData, Aug. 2022.

[4] M. Schultz, S. Reitmann, and S. Alam, "Predictive classification and understanding of weather impact on airport performance through machine learning," *Transportation Research Part C: Emerging Technologies*, vol. 131, p. 103119, Oct. 2021.

[5] X. Olive and L. Basora, "Detection and identification of significant events in historical aircraft trajectory data," *Transportation Research Part C: Emerging Technologies*, vol. 119, p. 102737, Oct. 2020.

[6] K. Seymour, M. Held, G. Georges, and K. Boulouchos, "Fuel Estimation in Air Transportation: Modeling global fuel consumption for commercial aviation," *Transportation Research Part D: Transport and Environment*, vol. 88, p. 102528, Nov. 2020.

[7] X. Olive, J. Krummer, B. Figuet, and R. Alligier, "Filtering Techniques for ADS-B Trajectory Preprocessing," *Journal of Open Aviation Science*, vol. 2, Mar. 2025.

[8] X. Olive, J. Sun, X. Huang, and M. Khalaf, "tangram, an open platform for modular, real-time air traffic management research," *Journal of Open Source Software*, vol. 10, Nov. 2025.

[9] D. Eddelbuettel, "A brief introduction to Redis," 2022.

[10] C. Hurter, R. Alligier, D. Gianazza, S. Puechmorel, G. Andrienko, and N. Andrienko, "Wind parameters extraction from aircraft trajectories," *Computers, Environment and Urban Systems*, vol. 47, pp. 28–43, Sept. 2014.

[11] J. Sun, H. Vu, J. Ellerbroek, and J. Hoekstra, "Ground-based Wind Field Construction from Mode-S and ADS-B Data with a Novel Gas Particle Model," p. 9, 2017.

[12] J. Sun, J. Hoekstra, and J. Ellerbroek, "Openap: An open-source aircraft performance model for air transportation studies and simulations," *Aerospace*, vol. 7, no. 8, p. 104, 2020.

[13] G. Jarry, D. Delahaye, and C. Hurter, "Towards aircraft generic quick access recorder fuel flow regression models for ads-b data," in *International Conference on Research in Air Transportation*, 2024.

[14] X. Olive, "traffic, a toolbox for processing and analysing air traffic data," *Journal of Open Source Software*, vol. 4, p. 1518, 2019.

[15] M. Schultz, J. Rosenow, and X. Olive, "Data-driven airport management enabled by operational milestones derived from ADS-B messages," *Journal of Air Transport Management*, vol. 99, p. 102164, Mar. 2022.

[16] X. Olive, M. Waltert, and M. Schultz, "Assessing Airport Surface Traffic Performance from Open Sources of Aviation Data," in *Proceedings of the 1st Air Transportation Research and Development Symposium*, (Prague, Czech Republic), June 2025.

[17] M. Ribeiro, J. Ellerbroek, and J. Hoekstra, "Review of conflict resolution methods for manned and unmanned aviation," *Aerospace*, vol. 7, no. 6, 2020.

[18] X. Olive and P. Bieber, "Quantitative Assessments of Runway Excursion Precursors using Mode S data," in *Proceedings of the 8th International Conference for Research in Air Transportation*, (Barcelona, Spain), 2018.

[19] M. Felux, P. Fol, B. Figuet, M. Waltert, and X. Olive, "Impacts of Global Navigation Satellite System Jamming on Aviation," *Navigation*, vol. 71, June 2024.

[20] J. Hoekstra and J. Ellerbroek, "Bluesky atc simulator project: an open data and open source approach," in *Proceedings of the 6th International Conference for Research in Air Transportation*, (Philadelphia, PA), 2016.

[21] K. Y. Hui, C. H. Nguyen, G. N. Lui, and R. P. Liem, "Airtrafficsim: An open-source web-based air traffic simulation platform.," *Journal of Open Source Software*, vol. 8, no. 86, p. 4916, 2023.