Integrating Advanced Visualization and Automated HPC Workflows for GNSS Prediction in Urban Air Mobility

Elena Deckert*
NASA Langley Research Center, Hampton, Virginia, 23666

Julian Gutierrez†
NASA Langley Research Center, Hampton, Virginia, 23666

Russell Gilabert†
NASA Langley Research Center, Hampton, Virginia, 23666

Evan Dill†
NASA Langley Research Center, Hampton, Virginia, 23666

Urban Air Mobility (UAM) brings forth new complex challenges, necessitating the development of advanced technologies to meet the specialized requirements of this emerging sector. Predictive and proactive risk mitigation capabilities will likely be necessary to ensure that these novel aviation operations are safely integrated into the National Airspace System (NAS). This is especially true when considering the operational complexity and low risk tolerance associated with highly autonomous vehicles in urban environments. By analyzing potential hazards and identifying where and when high-risk scenarios might arise, flight planning tools can be leveraged to help reduce exposure to such risks.

Currently, UAM systems rely heavily on Global Navigation Satellite Systems (GNSS) for accurate positioning. However, one significant safety concern is the loss or degradation of this critical navigation system. Particularly in low-altitude flights and urban settings, obstructions like buildings and trees are common and can cause reduced satellite visibility. Accurately predicting satellite visibility at varying altitudes and times is therefore essential to enhance the safety of UAM operations.

This paper introduces an automated High Performance Computing (HPC) workflow backend and a Graphical User Interface (GUI) frontend to support flight planning by producing advanced visualizations to analyze the risk of GNSS performance degradation or loss caused by obstructed satellite visibility. The backend and frontend presented in this paper are designed for NavQ, a GNSS quality prediction service. NavQ allows mission planners to identify safe flight trajectories and operational zones within urban landscapes to minimize possible failure caused by poor navigation data.

I. Introduction

The aviation industry is evolving to incorporate increasingly complex operational models, vehicle designs, and advanced technologies to expand airspace capacity and improve efficiency. Ensuring the safety of these new systems requires early identification and mitigation of potential safety risks before they can lead to serious consequences. To safely integrate these innovations, prediction frameworks are essential to enable proactive identification of high-risk scenarios and reduce exposure to such hazards from an operational perspective. One major risk for Urban Air Mobility (UAM) systems is the potential loss of critical navigation systems. UAM systems rely heavily on Global Navigation Satellite Systems (GNSS) to provide accurate, continuous positioning information necessary for safe urban navigation and obstacle avoidance within complex infrastructures. This is particularly critical for autonomous air vehicles, where high-precision positioning is required to ensure safe, reliable operation without human intervention.

The effectiveness of GNSS technology relies heavily on receiving direct signals from at least four satellites, or space vehicles (SV); their positions in the sky relative to the receiver; and the quality of the received signal. Sources of error

*Research Intern, Safety-Critical Avionics Systems Branch
†Research Engineer, Safety-Critical Avionics Systems Branch

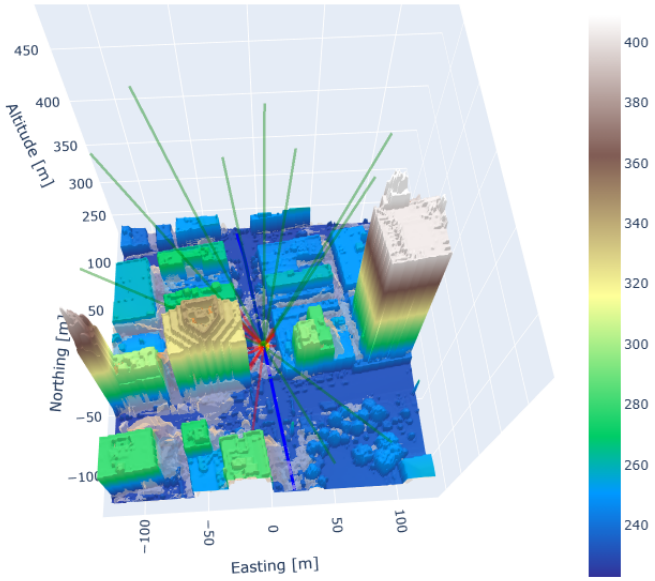


Fig. 1 Example 3D visualization produced by the frontend.

and environmental impacts on GNSS performance have been explored already [1, 2]. An unobstructed view of the sky is ideal to achieve accurate positioning. However, for low-altitude flight operations, such as those envisioned for UAM, maintaining this visibility is challenging due to physical obstructions like buildings and terrain, which often block portions of the sky. Therefore, urban environments can introduce errors in the position reported by GNSS receivers due to the reception of non-line-of-sight (NLOS) satellites and multipath effects, making the modeling process challenging [3, 4]. This can result in degraded or loss of position estimates when well-positioned, direct line-of-sight (LOS) satellites are obstructed [5]. To help mitigate these risks for UAM operations, a High-Performance Computing (HPC) workflow has been developed to visualize GNSS performance quality and availability. Low UAM Maturity Levels (UMLs) [6, 7] have resulted in minimal assurance for urban GNSS receivers, especially due to NLOS/multipath effects. As research progresses and advancement in real-time receiver-side NLOS/multipath mitigation become available, GNSS-assured receivers will become paramount to proper risk management for UAM. In the meantime, mission planners can benefit from risk prediction tools, such as the one we introduce in this paper, to manage potential navigation challenges effectively.

Existing tools for visualizing satellite availability in urban areas can provide valuable mapping capabilities [8, 9]. However, they lack interactive features and user-friendly interfaces, making them less accessible for analysis and adjustments. Additionally, these tools do not support the ability to modify flight routes easily based on the visualized data and the visualizations are not specifically designed to help flight planning.

The GNSS prediction workflow presented in this work can enable increased safety assurance by visualizing the risks of loss in position accuracy. We base our workflow on NASA's NavQ [10], a software that allows operators to maximize the likelihood of maintaining high-integrity position information throughout the duration of a flight operation and filter areas that should be avoided due to loss of navigation data. Figure 1 shows an example of NavQ's output visualized in 3D. This work provides an intuitive interface that can be used for fixed point analysis where an area around a point of interest (POI) is selected and analyzed over a certain period of time. This allows the user to see the movement of satellites over time and determine at which altitude a given area may not receive sufficient coverage at certain times due to the position of the satellites. When used as a path planning support tool, a user can provide a desired flight path and modify the height and time of individual points of interest to achieve sufficient satellite visibility throughout the flight route. These capabilities are enabled by the HPC automation workflow presented in this work, allowing us to generate and retrieve data efficiently.

This paper is composed of 5 sections. Section II describes the NavQ backend service used to compute the data for the visualizations. Section III outlines the set up of the frontend implementation for this tool. Section IV provides results on the computational time. Finally, we summarize our findings in Section V.

II. Backend Implementation

Our software is developed as a wrapper over the NavQ tool [11], a GNSS risk forecasting tool that leverages HPC to predict LOS GNSS availability in a city, to estimate the quality of GNSS performance in complex environments [10]. NavQ was implemented in C++ with CUDA [12], Thrust library [13], and OpenMP [14] for fast parallel processing, where the algorithms were designed and optimized for Graphics Processing Units (GPU). NavQ was developed to improve situational awareness of GNSS quality in complex environments and enable the identification of areas that may induce poor or unacceptable navigation system performance for pre-flight and/or in-flight route planning.

A. High-level Design

The deployment is done via Kubernetes [15] and Docker [16] containers, with a Flask Application User Interface (API) [17] to handle requests. This enables efficient resource management and containerization to enhance the processing of large computational workloads efficiently. Additionally, this allows the backend to be easily serviceable and provides flexibility to run in most systems. A GPU and CPU-only version of the dockers were designed to allow faster processing if GPUs are available. The CPU-only version compiles the NavQ software with OpenMP to execute the algorithms, while the GPU version compiles the NavQ software with OpenMP, and Nvidia's Thrust and Compute Unified Device Architecture (CUDA) for the parallelized algorithms.

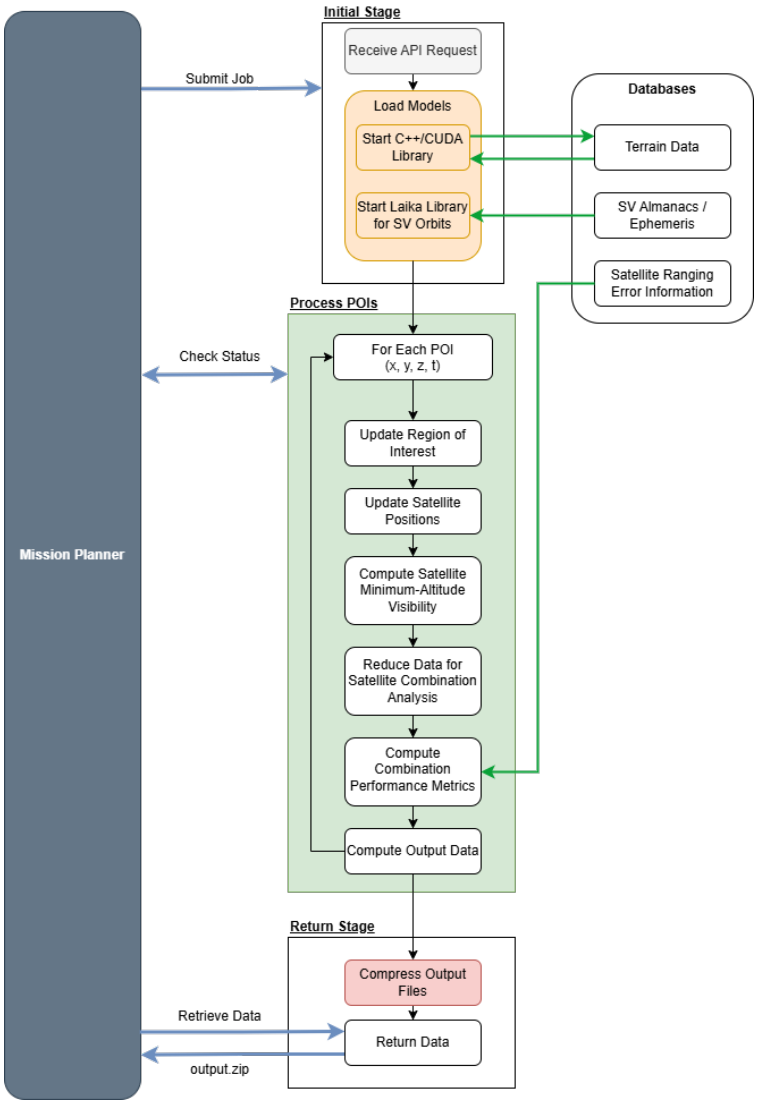


Fig. 2 High-level block diagram of the NavQ backend.

The backend consists of a web page that receives an API request with a specified payload and then allows the user to retrieve the requested data once the job is complete. The backend was designed to wrap around NavQ to provide modularity and easy implementation/modification to the existing NavQ software stack. The backend handles input 3D map manipulation, Laika* [18] instantiation, payload verification, error handling, and the instantiation of NavQ if the job is serviceable. The main routes used in the backend are shown in Figure 2, and are described as follows†:

1) **Submit Job**: This route receives a payload with the parameters required to run the analysis needed by the mission planner. It will return a job ID if the job is submitted successfully.
2) **Check Status**: This route receives a job ID and provides summary information on the status of the job. A general status message is used to indicate whether the job has just started, is currently running or has completed. This route will provide additional information such as current percentage of job completed, as well as general job information such as job start time, IP address of requester, etc. When the job is complete, this same route will provide a summary of all of the POI data as a quick way for mission planners to retrieve summary information without the need to download data.
3) **Retrieve Data**: This route allows the mission planner to download all of the data that was requested as a ZIP file.

B. NavQ POI Processing

NavQ creates a discretized time-varying 3D volume that computes the estimated quality of GNSS-based position solutions. To identify obstacles that can hinder the visibility of satellites, NavQ requires satellite state information and an accurate surface model of a city's ground structure. A 3D city model is provided as elevation data in a GeoTIFF file containing a raster grid of elevation for the requested region and metadata indicating the map projection and affine transformation coefficients. These models are produced from aerial LiDAR surveys, and rasterized with a 1×1 m resolution, and are only accurate to the day of data acquisition; which means that new buildings or changes in vegetation and mobile geometry are not reflected. A function was implemented in the backend to automatically determine which map NavQ can use from the integrated maps based on the position of the first POI. The simulation iterates through each POI (identified by a unique latitude, longitude, altitude and timestamp) and runs the following stages as described in Figure 2:

1) **Update Region of Interest (ROI)**: Select the area of the map around the current POI.
2) **Update Satellites Positions**: Using the POI as a reference and the timestamp to estimate the positions of the satellites in the sky using the Laika library [18], NavQ determines the azimuth and elevation angle for each satellite.
3) **Compute Satellite Minimum-Altitude Visibility**: For the selected ROI, a GPU-accelerated function is used to determine the minimum-altitude at which each satellite is visible using the algorithms described in [10].
4) **Reduce Data for Satellite Combination Analysis**: A GPU-accelerated function is used to determine all of the unique combinations of satellites found within the ROI.
5) **Compute Combination Performance Metric**: A parallelized CPU function is used to process all of the unique satellite combinations to determine their respective performance metrics (e.g., SV count, GDOP, HDOP, PDOP, covariance matrix).
6) **Compute Output Data**: Based on the parameters provided by the mission planner, each requested output is processed by a GPU function and an array is populated with the correct data accordingly and stored as TIFF or JSON files.

C. Data Retrieval

As mentioned in Section II.B, NavQ can be used to request and retrieve different types of data. For the frontend described in this work, we use the satellite data, horizon, time values, map area data, satellite count, risk [19], and Dilution of Precision (DOP) data. The backend can provide the satellite information (azimuth, elevation, name, and visibility status) when skyplot data is requested. The horizon is also provided if skyplot data is requested. The horizon is a 2D array of the azimuth angles and elevation angles that determine the visibility of the sky based on the building structures surrounding the POI. This data is useful to visualize potential blockages of satellites.

The backend additionally provides the time and the requested data type values for the defined timeframe. The output

*Laika is an open source library used to obtain the satellite ephemeris data from online repositories to estimate the satellites positions in the sky relative to the POI. It allows the user to specify which GNSS constellations to use in the model as well.

†Additional backend routes were developed to support extended functionality; however, these are beyond the scope of the current work and are not discussed in detail here.

data can be computed for different data types where the values are computed for each voxel in the ROI, which is defined by the box size provided by the user. The following data types are supported by the software:

- **Satellite Count**: This metric is the number of LOS satellites.
- **DOP**: These are metrics that quantify the expected accuracy of a GNSS position solution based on the number of satellites within LOS and their geometry in the sky.
- **Risk**: This metric was designed for the path planning algorithm in [19] to estimate the risk associated with GNSS positioning based on the covariance matrix of the positioning error. This metric is a normalized value between 0 and 1, where 0 is no risk associated with GNSS hazards, and 1 means no satellites are visible.
- **Map**: This value simply provides the altitude of the area extracted from the 3D map.

III. Frontend Implementation

The frontend was designed to offer a user-friendly interface for visualizing data produced by the NavQ backend. While the NavQ backend is operated independently, this frontend handles all server communications and data manipulation, providing an easy interface for the user. The user interface design includes a primary control panel that allows users to set initial simulation parameters and modify an uploaded list of POIs. Once these parameters are submitted, a request is sent to the backend with the payload to generate the requested data. A progress bar indicates the status of the job on the server, and when the job is complete, the data is retrieved in the form of a ZIP file. Once that data is processed, a series of visualizations are generated, including 2D and 3D plots that integrate all of the requested data. Additional controls become available to dynamically adjust the plots based on parameters such as time, height, and data type. Both plots are animated to illustrate data progression over a defined time window.

This interface is implemented in Python, with Matplotlib [20] used for 2D visualizations and Plotly [21] for the 3D plot. The Plotly library was selected to facilitate interactive surface plotting capabilities with adjustable opacity. The interface and visualizations are designed on Jupyter notebooks [22] and transformed into an accessible HTML interface using Voila [23], and can be run through any modern browser. The implementation structure of the tool is depicted in Figure 3.

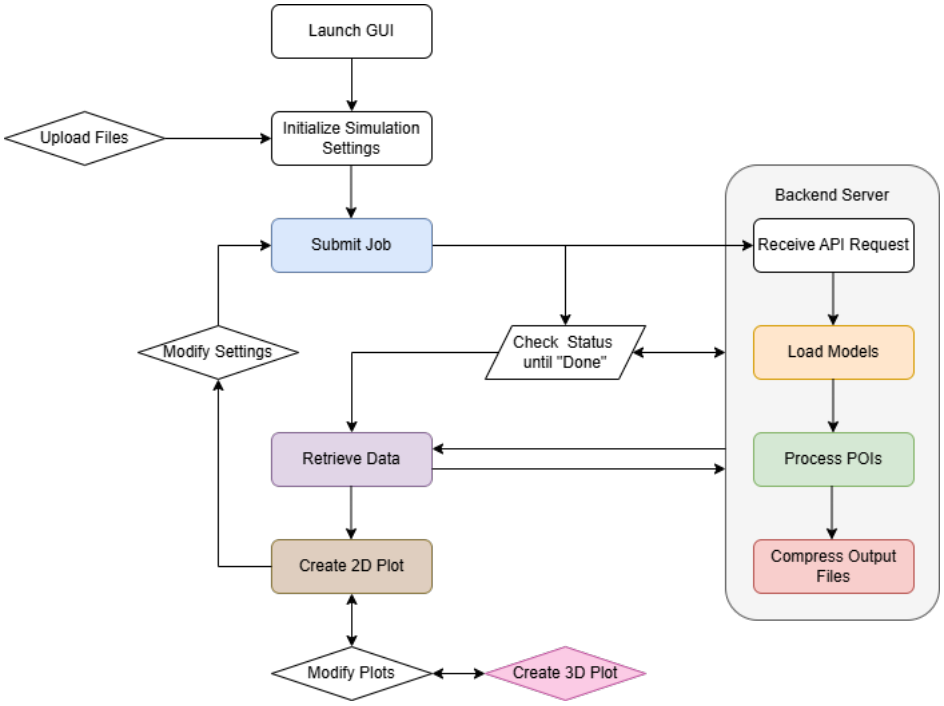


Fig. 3 High-level block diagram of the NavQ frontend.

A. Simulation Controls

The simulation parameters can be specified in the primary control panel as shown in Figure 4. The user can provide inputs such as figure configurations, satellite constellations, data types, output types to be visualized, and threshold values to build the analysis on. Users can set figure parameters directly or upload configuration files to recreate prior simulations. Additionally, a building file can be uploaded to include surface structures not present in the maps integrated into NavQ. The selected output types define the data requested for computation and correspond to the plots available for visualization. Thresholds are divided into warning and failure levels, allowing visualization of varying risk levels.

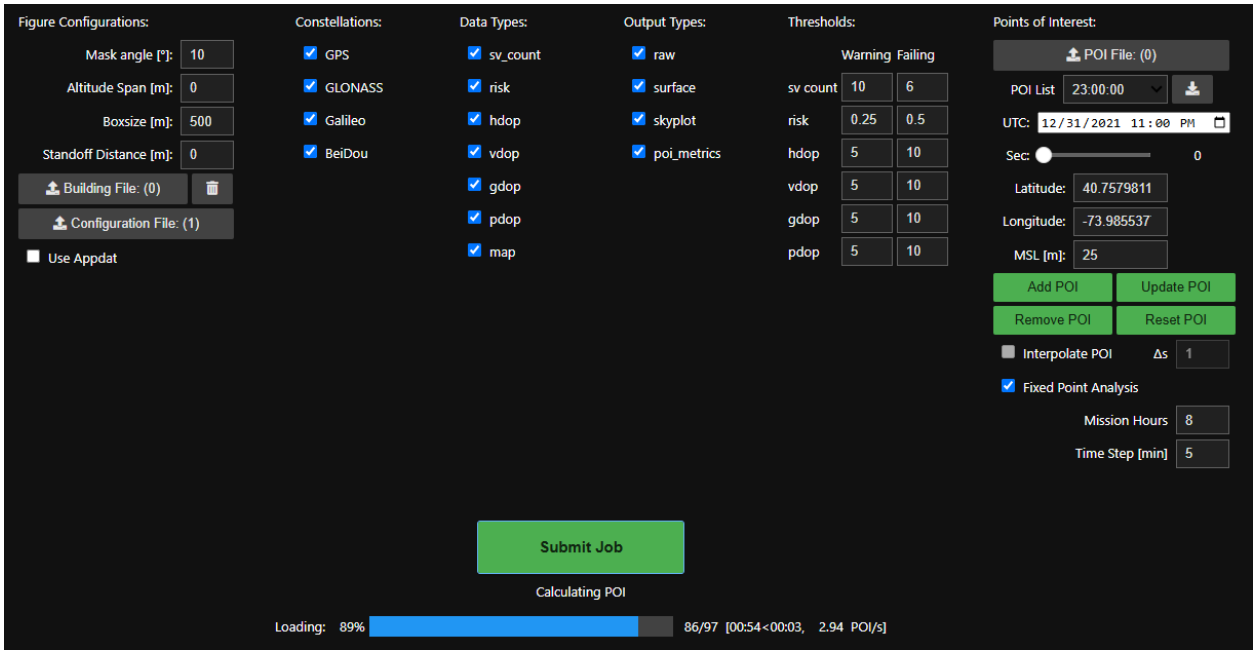


Fig. 4 User controls to initialize or modify simulation settings.

The tool also supports the ability to create a list of POIs to analyze. Each POI is identified by its coordinates (longitude, latitude, altitude) and a UTC timestamp. Users can upload a POI list and modify it by adding and removing points, or updating existing points. For single-point input, a dedicated fixed-point analysis can be conducted, analyzing the given coordinates over a specified time interval determined by the users input. For multiple points, the tool provides an option to interpolate between the given points with a chosen interval. Once the user has completed selecting the input parameters and submitted the job, the frontend sends a payload request to the backend based on the current simulation controls. After the job is processed, it retrieves the requested data, which is then utilized for generating the plots.

B. Output Display

1. Job Summary

The job summary displays an overview of the computed data. It gives insights on the computation times for each step in the backend and the size of the output data. Moreover, it provides the overall percentage of POIs that fall above or below the warning and failing thresholds, which allows a quick analysis of a proposed POI or flightpath. Figure 5 shows an example of this section of the outputs.

2. 2D Visualization

The dynamic visualizations, generated through multiple plots, illustrate the satellite coverage for specific areas around a POI and highlight potential blind spots. The two-dimensional plots include a skyplot, a data plot, and a time-series plot. Figure 6 shows a skyplot example at the left, a data plot of the satellite count in the middle, and the time series values for the satellite count on the right. NavQ provides the interface with the data requested through the data types input, which then generates the desired plots accordingly.

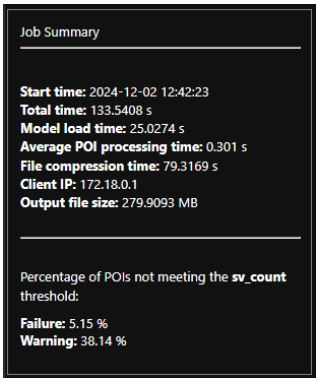


Fig. 5 Job summary providing insight on computed data.

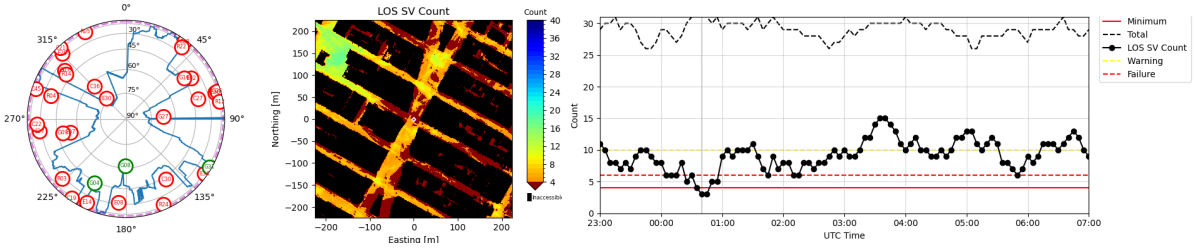


Fig. 6 Two-dimensional plots for a POI at latitude: 40.757981, longitude: -73.985538, and 25 m MSL in New York City, analyzing the SV visibility at a fixed point on 2024-10-02 00:40:00 UTC over a timeframe of 8 hours.

The skyplot presents the satellite positions and horizon line as determined by the surrounding infrastructure on a polar plot at a current point in time. The frontend configures the polar plot and translates the horizon data. Given as a 2D array of the azimuth angles and elevation angles, it is adapted to the polar plot to be plotted as a continuous line that closes upon itself. The satellite data retrieved as a list of azimuth, elevation, and visibility status values is similarly converted into coordinates which are plotted on the polar plot in red or green reflecting the visibility of each satellite.

The data plot can be adjusted to display one of the selected data types, surfaces, and thresholds. The data is retrieved as a 2D array for each data type reflecting the corresponding value for each map cell. The frontend creates a map grid and assigns a color to each cell based on the values in the data array and a colormap selected to represent the according data type. In "raw plot" mode, the data type determines whether it displays the number of visible satellites, the normalized risk, or the DOP magnitude surrounding the POI. In "surface" mode, it shows the minimum altitude required to exceed the failing or warning thresholds around the POI for the selected data type. The displayed map is centered around the current POI.

In combination, the visualization of the satellite visibility and data offers a top-down and bottom-up perspective that clarifies the area layout with respect to the data type and satellite coverage. This combined view aids in identifying factors affecting visibility, such as nearby buildings, elevation, or vegetation, allowing for decisions about possible adjustments, such as shifting the POI location or increasing its elevation.

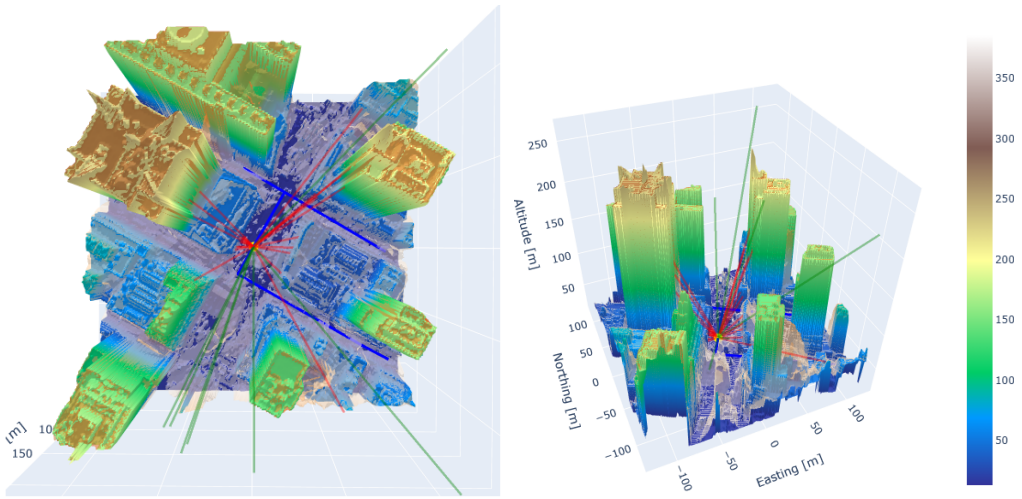
The time-series plot illustrates how the number of visible satellites, normalized risk, or DOP magnitude changes for each POI or at a fixed point over time. It provides a temporal overview as a line graph, showing when visibility falls below warning or failure thresholds, thus indicating which POIs or timestamps are at potential risk of being affected by positional degradation or signal loss. A vertical timer indicates the current position in time that is currently reflected in the skyplot and data plot.

3. 3D Visualization

The 3D data visualization consolidates all the retrieved data in one plot, integrating the 2D plot information into a spatial representation of elevation and surface features in combination with the satellite information. The visualization retrieves the data using the map data type to display the area's surface sourced from the NavQ-integrated 3D map for the specified area around the current POI. This surface illustrates the urban infrastructure, including elevation, buildings, or

trees enhanced by a color gradient that indicates altitude. The second surface layer in light orange is draped over the map data, representing the minimum height needed at each map cell to achieve adequate satellite visibility, based on the selected data type and threshold. Its reduced opacity allows the user to discern the height difference between the map data and the necessary elevation for a receiver to produce sufficient location accuracy (Figure 7). A 3D plot is essential in allowing the representation of both surfaces simultaneously, a feature not possible in the 2D plots.

Additionally, the tool can display the satellite rays directed towards the POI as depicted in Figure 7. The rays are tracing paths from the POI based on the azimuth and elevation until they intersect with the surface or exceed the plot limits. This illustrates potential obstructions by buildings or other structures and enables users to identify specific objects that impede satellite visibility. The current POI is plotted as a sphere and the coordinates of the list of all POIs are plotted as a line graph representing the flight path of a system.



(a) (b)

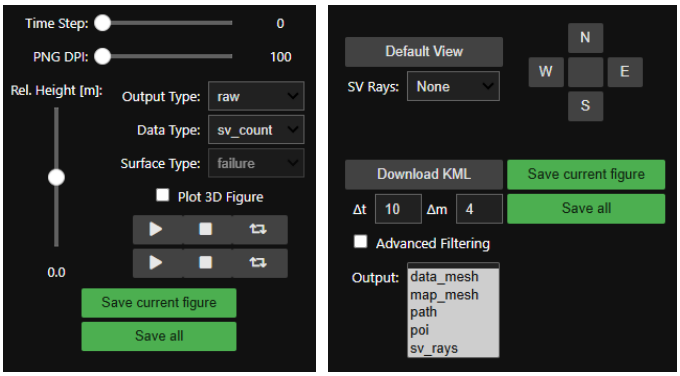
Fig. 7 The satellite rays and a flight path are plotted in the 3-dimensional city model around Times Square in New York City, NY visualized from a top view (a), and from the south/west (b).

This 3D model merges the perspectives of the top-down and bottom-up 2D plots into an interactive cube, viewable from any angle. As an interactive plot, it allows the user to select different times and POIs allowing the simulation of flight paths according to the list of POIs or the analysis of the satellite visibility at a fixed point over a certain period of time. The 3D visualization, particularly useful in urban environments, helps demonstrate the impact of buildings and other structures on satellite visibility in an intuitive manner.

C. User Interaction

Users can modify their simulation settings and resubmit jobs as needed. Once a job is completed, interactive controls enable further user interaction (Figure 8). These controls allow the frontend to retrieve selected data types, times, and altitude levels, dynamically adjusting the plots in response.

For instance, interactive sliders allow users to adjust height and time, providing insights into how POI configurations can be optimized to meet GNSS accuracy requirements. Depending on the user's objective, they can modify data types, select surface options, and set thresholds to a warning or failing value. The 3D plot can be set to predefined camera angles, allowing consistent perspectives during analysis. This flexibility enables detailed exploration of flight paths and environmental factors around a POI over time. Additionally, users can simulate time progression or altitude changes using a play button, which animates the data over the specified time frame or altitude range adding an additional time dimension. For reporting or presentation purposes, users can save current plot snapshots or create GIFs that represent data across the entire time window. Finally the frontend allows users to download the 3D data as a KML file with the option of applying advanced filtering to downsize the data. These features enable comprehensive analysis, fostering a clearer understanding of how changes in parameters impact satellite visibility and positioning accuracy.



(a) (b)

Fig. 8 User controls to modify the generated plots (a) and change the view of the 3D plot (b).

IV. Performance Analysis

We evaluated the performance of the backend and frontend using flight path simulations under typical analysis control settings for a short and long path. The short flight path consisted of 66 POIs, and the long flight path included 222 POIs, with analysis conducted at box sizes of 150 m and 300 m. The execution times shown in this section are averaged across 10 runs. The computational durations of the different steps were assessed and compared between a local GPU-based server and a remote CPU-based server hosted on APPDAT. APPDAT is a platform hosted at NASA Johnson Space Center (JSC) for developing cloud-native applications [24]. The local GPU-based server has a 64-thread Intel Xeon Silver 4216 CPU with an Nvidia A6000 GPU and 128 GB of RAM. The CPU-based server on APPDAT was a n2-highscpu-32 Google Cloud compute system with 32 cores and 32 GB of RAM.

A. Local GPU-based Server Performance

We first evaluated the performance of the local GPU-based server and provide a breakdown of the total execution time for a standard short request. Figure 9 shows the breakdown of all of the different stages we described in previous sections.

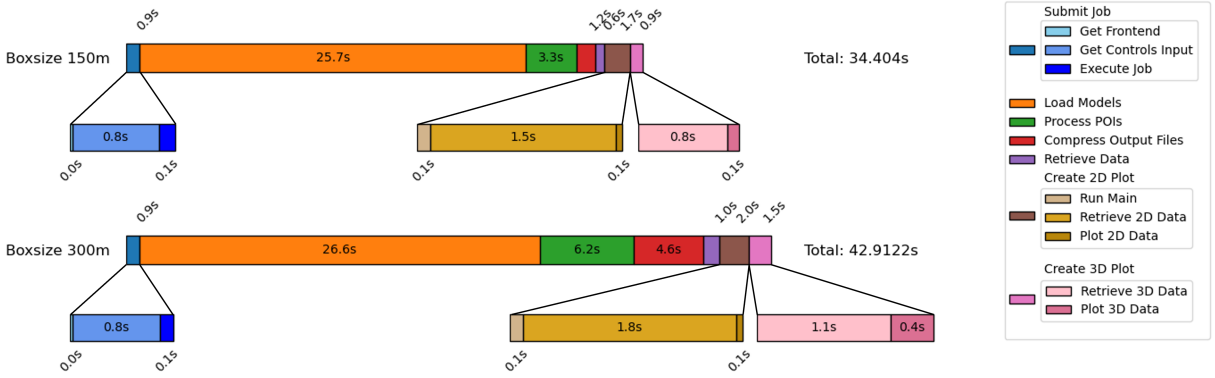


Fig. 9 Computation time on local GPU-based server for a short simulated path with 66 POIs.

The submit job segment is composed of 3 steps: 1) retrieving the frontend, 2) getting the input from the control widgets, and 3) executing the job by sending the payload to the server. This step's execution time remains consistent

even when increasing the box size of the request. The majority of the time is spent getting the control inputs which is affected by the way widgets are loaded and their information retrieved in Jupyter.

The majority of the computation time is spent on the backend server. In the local GPU server, most of the time is spent loading the models as shown in Figure 9. Overall, the time to load the models is consistent for different types of payloads. The time to start the C++/CUDA library can be slightly impacted by uploading a terrain file, which is done in the short path but not the long path, though the majority of the time is attributed to Laika's start up process to retrieve the satellite information from online servers based on the constellations selected by the user.

Once the backend has loaded the models, it is able to efficiently process the POIs with an average processing time of less than 0.1 s per POI. Increasing the size of the box has a negative effect on the POI processing time and compression of the output file. A larger box size generates more data, as it encompasses a broader area around each POI over time. Consequently, the computation and compression times increase by at least a factor of 2. In this case, doubling the box side length (4x increase in area), results in a 3x increase in the size of the compressed file: 150 m produces 5.585 MB while 300 m yields 16.485 MB.

Most of the time is spent retrieving plot data in the plot generation stage. This process is optimized when plots are later updated via the controls. The creation of 2D and 3D plots becomes significantly faster as plot data for both is retrieved simultaneously, avoiding redundant retrievals. Individual plot elements are updated on the basis of the changed data, minimizing the need to redraw static elements. This noticeably improves the plotting time as updating the figures can be done up to 10 times faster than populating the initial plot depending on the data type that is modified. Plotting only the specified data types reduces the computation time as it not only reduces the number of plots but also the amount of data that needs to be retrieved and processed.

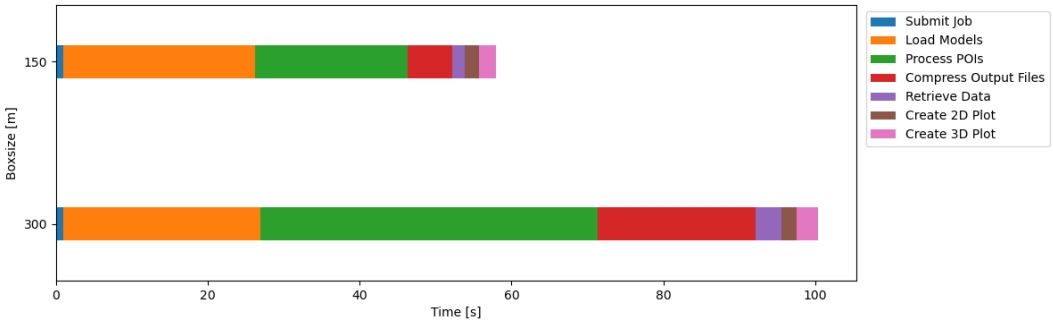


Fig. 10 Computation time on local GPU-based server for a long simulated path with 222 POIs.

When considering a longer path the job execution time is impacted due to the larger payload being passed to the server as shown in Figure 10. A significant increase in POI processing time and output file compression is observed with the longer path. The output data size increases linearly with the increase in POIs being processed. As a result, the increase from 66 to 222 POIs results in a 5.4x increase in the output zip file size for a 150 m box size, and 5.7x increase in the output zip file size for a 300 m box size. The average processing time for each POI is slightly higher than for the short path. This is caused by the difference in terrain that is being considered around the POI.

The number of POIs also affect the time required for data retrieval and plotting in the 3D visualization, though the 2D plot remains relatively unaffected. Data retrieval is influenced more by the increased path length than by the box size, as map data for all POIs is precomputed to establish consistent 3D plot limits based on the maximum values across all POIs. In contrast, plotting time is more strongly impacted by the box size than the path length, as only a single POI is plotted at a time and the box size affects the amount of data around that point that has to be plotted.

B. Remote CPU-based Server Performance

Figure 11 shows the execution time for the requests sent to the remote CPU-based server. It is clear that the majority of the time is spent on the backend processing the POIs. There is also a clear execution time increase when using the larger box size of 300 m. The CPU-based server uses the CPU-only version of the NavQ software which has been parallelized with OpenMP. Despite this, the lower number of usable cores on this server severely limits the CPU's ability to process the large areas in parallel, and results in severe performance limitations when compared to the GPU-based server.

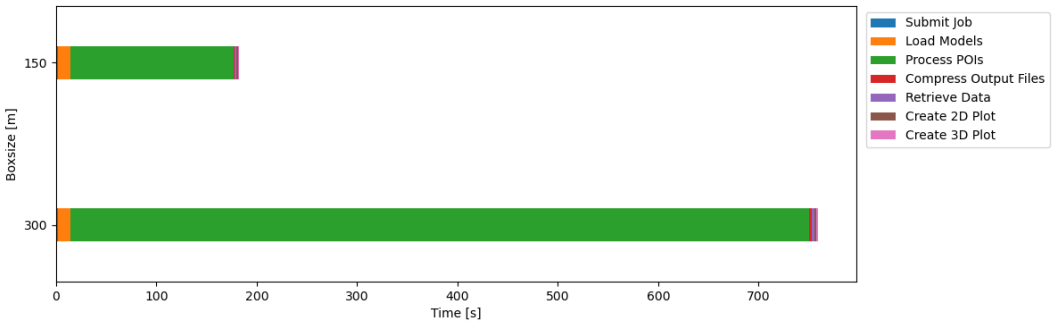


Fig. 11 Computation time on remote CPU-based server for a short simulated path with 66 POIs.

C. Performance Comparison

Figure 12 shows the comparison of the execution time between the local GPU-based server and the remote CPU-based server. The load model time is shorter for the CPU-based server which could be a result of a better connection to the servers Laika accesses and faster frequency of the CPU. The most substantial difference lies in the POI processing time, which increases drastically on the CPU-based server. Processing times rise from less than 0.1 s on the GPU to 2.47 seconds for a box size of 150 m and 11.18 seconds for a box size of 300 m. The file compression time is shorter for the simulation run on the CPU-based server. This can be caused by a difference in frequency which the core is running on. All frontend related stages, including plotting times, remain largely unaffected by which version of the backend is used.

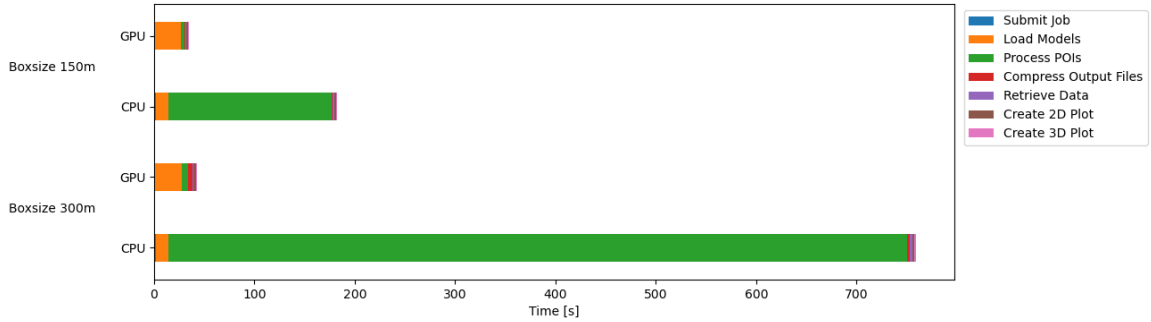


Fig. 12 Computation time comparison between the local GPU-based server and the remote CPU-based server for a short simulated path with 66 POIs.

V. Conclusions

In this paper we introduce a backend and frontend that integrates an automated HPC workflow to produce advanced visualizations to aid in GNSS risk mitigation analysis for UAM. The frontend is designed to interface with the backend, eliminating the need to manually communicate with the backend server. It allows mission planners to visualize flight paths or areas of interest over a period of time as an intuitive tool for data manipulation and analysis. We were able to design an efficient backend and frontend to create and update visualizations, producing a tool that helps users understand the large amounts of data generated by NavQ and interact with them.

This work also highlights the significance of leveraging HPC capabilities, specifically those provided by GPUs. Our initial performance analysis demonstrates the substantial improvements that can be achieved through GPU-accelerated computations. By harnessing the computational capacity of GPUs, we have shown that the software's performance can be significantly enhanced, underscoring the importance of this technology in supporting real-time GNSS prediction and flight path analysis for UAM applications. This work showcases the potential benefits of HPC adoption and serves as a stepping stone for future investigations into optimizing GPU utilization to maximize performance and efficiency.

The conclusions drawn from this research highlight the potential of integrating advanced visualization and automated

HPC workflows for GNSS prediction in UAM applications. However, there are opportunities for further improvement to enhance the overall efficiency and effectiveness of the developed system. Future work will focus on optimizing the Laika library to reduce loading times, thereby facilitating faster data generation and flight path analysis. Additionally, implementing real-time communication capabilities between the frontend and backend systems will enable seamless integration with operational environments, supporting real-time requests and decision-making processes. By addressing these limitations, we aim to further advance the adoption of HPC-aided GNSS prediction, enabling safer and more efficient flight operations.

References

[1] Amin, M. G., Closas, P., Broumandan, A., and Volakis, J. L., "Vulnerabilities, threats, and authentication in satellite-based navigation systems [scanning the issue]," *Proceedings of the IEEE*, Vol. 104, No. 6, 2016, pp. 1169–1173.

[2] Pelc-Mieczkowska, R., Tomaszewski, D., and Bednarczyk, M., "GNSS obstacle mapping as a data preprocessing tool for positioning in a multipath environment," *Measurement Science and Technology*, Vol. 31, No. 1, 2019, p. 015017.

[3] Betz, J. W., *Engineering satellite-based navigation and timing: global navigation satellite systems, signals, and receivers.*, John Wiley & Sons, 2015.

[4] Appleget, A., and Bartone, C., "A consolidated GNSS multipath analysis considering modern GNSS signals, antenna, installation, and boundary conditions," *In Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation*, ION GNSS+ 2019, 2019, p. 2855–2869.

[5] Morton, Y. J., van Diggelen, F., Spilker Jr, J. J., Parkinson, B. W., Lo, S., and Gao, G., *Position, Navigation, and Timing Technologies in the 21st Century, Volumes 1 and 2: Integrated Satellite Navigation, Sensor Systems, and Civil Applications, Set*, John Wiley & Sons, 2020.

[6] Goodrich, K. H., and Theodore, C. R., "Description of the NASA Urban Air Mobility Maturity Level (UML) Scale," *AIAA SciTech 2021 forum*, 2021, p. 1627.

[7] Patterson, M., Isaacson, D., Mendonca, N., Neogi, N., Goodrich, K., Metcalfe, M., Bastedo, B., Metts, C., Hill, B., DeCarme, D., Griffin, C., and Wiggins, S., "An Initial Concept for Intermediate-State, Passenger-Carrying Urban Air Mobility Operations," *AIAA SciTech 2021 forum*, 2021.

[8] Zhao, Y., Wu, B., Wu, J., Shu, S., Liang, H., Liu, M., Badenko, V., Fedotov, A., Yao, S., and Yu, B., "Mapping 3D visibility in an urban street environment from mobile LiDAR point clouds," *GIScience & Remote Sensing*, Vol. 57, No. 6, 2020, pp. 797–812.

[9] Lu, Y.-H., and Han, J.-Y., "Gnss Satellite Visibility Analysis Based on 3d Spatial Information in Urban Areas," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. 43, 2020, pp. 123–128.

[10] Gutierrez, J., Young, S., Moore, A., Gilabert, R., Dill, E., Bates, E., Peretic, M., Schmitt, K., and Scholz, A., "A High-Performance Computing Predictive GNSS Performance Monitor for Autonomous Air Vehicles in Urban Environments," *NASA Technical Memorandum*, 2024.

[11] Dill, E., Gutierrez, J., Young, S., Moore, A., Scholz, A., Bates, E., Schmitt, K., and Doughty, J., "A Predictive GNSS Performance Monitor for Autonomous Air Vehicles in Urban Environments," *Proceedings of the 34th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2021)*, 2021, pp. 125–137.

[12] NVIDIA Corporation, "NVIDIA CUDA C Programming Guide," , 2021. Version 11.2.

[13] Bell, N., and Hoberock, J., "Thrust: A productivity-oriented library for CUDA," *GPU computing gems Jade edition*, Elsevier, 2012, pp. 359–371.

[14] Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., and McDonald, J., *Parallel programming in OpenMP*, Morgan kaufmann, 2001.

[15] "Kubernetes." GitHub.com. `https://github.com/kubernetes/kubernetes`, (accessed: 2024-10).

[16] Merkel, D., "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, Vol. 2014, No. 239, 2014, p. 2.

[17] Grinberg, M., *Flask web development: developing web applications with python*, " O'Reilly Media, Inc.", 2018.

[18] Schafer, H., Santana, E., Haden, A., and Biasini, R., "A commute in data: The comma2k19 dataset," *arXiv preprint arXiv:1812.05752*, 2018.

[19] Gutierrez, J., Neogi, N. A., Kaeli, D., and Dill, E. T., "A High-Performance Computing GNSS-aware Path Planning Algorithm for Safe Urban Flight Operations," *AIAA AVIATION 2022 Forum*, 2022, p. 3461.

[20] Hunter, J. D., "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, Vol. 9, No. 3, 2007, pp. 90–95. https://doi.org/10.1109/MCSE.2007.55.

[21] Inc., P. T., "Collaborative data science," , 2015. URL https://plot.ly.

[22] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C., "Jupyter Notebooks – a publishing format for reproducible computational workflows," *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides and B. Schmidt, IOS Press, 2016, pp. 87 – 90.

[23] "Voila-dashbords." GitHub.com. https://github.com/voiladashboards/voila, (accessed: 2024-10).

[24] Shoemaker, P., and Estes, C., "Accessible Telemetry Streams using a Zero Trust Architecture for the Flight Operations Directorate," 2021.