International Journal of Computer Science and Engineering Innovations
ISSN: 3107-9458 | Volume 2 Issue 3, 1-22, July-September 2026
DOI: https://doi.org/10.64137/31079458/IJCSEI-V2I3P101

Received: 12/06/2026 Revised: 15/06/2026 Accepted: 25/06/2026 Published: 01/07/2026

Original Article

Developing a Scalable SQL-Based Framework for Ingesting and Analyzing Aerospace Simulation Output Data

[1]DWIGHT RANSOM, [2]SIKHA BAGUI
[1,2]Department of Computer Science, University of West Florida, Pensacola, Florida, USA.

ABSTRACT: *Typical aircraft external loads processes were originally developed for smaller models under legacy computational constraints. While these processes have been adapted to handle modern model sizes, they were not designed to scale efficiently with the growing volume and complexity of simulation data. As a result, downstream plotting and critical case selection workflows remain inefficient and lack consistent traceability across aircraft and analysis loops. To address these limitations, a relational SQL-based framework was developed to structure and scale external loads data across millions of records and multiple analysis types into a unified and structured dataset. This framework enables automated post-processing tools for envelope visualization and comparison, along with a reduction methodology that identifies the smallest subset of critical cases required to preserve structural design conditions within a specified tolerance. The database supports efficient extraction of one-dimensional envelope extrema and optional two-dimensional envelope boundaries using convex hull methods. These envelope requirements are used to define a constraint set, which is solved using a greedy set-cover algorithm with a user-defined tolerance (e.g., 3%) to determine the minimal set of representative conditions. SQL enables efficient filtering, aggregation, and envelope extraction over large datasets, while higher-order geometric and combinatorial operations are performed in Python after data reduction. Application of the framework to a representative dataset containing 4,914 flight conditions produced approximately 696 one-dimensional envelope requirements, which were further reduced to a final subset of 82–94 critical cases depending on the inclusion of multidimensional constraints. The reduction process maintained envelope fidelity within the specified tolerance and completed within practical computational time. This approach consolidates large-scale external loads data into a single, structured, and traceable system and enables repeatable, automated workflows for plotting and case reduction. The resulting process significantly reduces manual engineering effort while maintaining alignment with established analysis practices, providing a scalable and reproducible solution for managing large external loads datasets.*

KEYWORDS: *Relational Database, Aerospace Data, Data Ingestion, Data Reduction; Simulated Flight Cas, Structural Load, Flight Conditions, External Loads.*

1. INTRODUCTION

Modern aircraft external loads analyses generate data for thousands to hundreds of thousands of flight conditions. Each of those cases needs to contain all the data for every component of the aircraft, creating large amounts of outputs to organize and analyze. This vast amount of data has been substantially growing as computational power and software development have increased and become capable of handling larger and more detailed models. The downstream recipients of this data, structural engineers, require only a small percentage of the original cases to assess in detail for designing and sizing the aircraft. Current workflows for external loads engineers tend to use company-created internal scripts that were written for simpler models and not optimized to handle current details. As a result, loads teams spend considerable effort navigating old processes that haven't been designed to scale with the increase in output and cannot handle unique situations. Because selection of critical load cases often relies on simplified enveloping and manual judgment, deliberate reduction of the case set is either performed inconsistently or overlooked entirely. Implementing a deterministic algorithm that identifies the smallest subset of cases whose envelopes remain within an acceptable tolerance can significantly reduce structural design cycle time while improving the traceability and repeatability of the selected cases.

The goal of this work is to establish a structured and scalable framework to handle external loads output data and post-processing scripts to automate critical case selection and reduction techniques to increase traceability and efficiency. A relational model allows the data to be normalized across aircraft, analysis types, components, conditions, and revisions, enabling indexed queries over millions of rows that would be infeasible with text-based storage. It also consolidates all external loads outputs into a single structured system rather than scattered files across multiple working directories, eliminating redundancy and simplifying long-term data management. SQL provides schema evolution and referential integrity, both of which are essential as analysis tools and output formats evolve over time. These capabilities ensure that the data structure can accommodate future changes in external solvers such as NASTRAN and integrate additional software into the post-processing workflow while still supporting fast, flexible querying. The reduction algorithm treats critical case selection as a subset

selection problem, identifying the smallest set of cases whose envelopes stay within a specified tolerance (e.g., 3%) of the full dataset. This approach parallels known computational problems such as set cover, convex-hull identification, and ε-approximation, which provide a useful framework for understanding accuracy and performance. In practice, the algorithm applies greedy selection and geometric approximation methods to reduce both 1-D and 2-D envelopes while still preserving the behavior of the original load set.

This work is important in the aircraft design process because it automates and creates traceability for the computationally heavy post-processing of aircraft models, while staying flexible to handle multiple aircraft, models, and software advancements for years to come. The resulting system improves the external loads-to-structural-engineer handoff by reducing the critical cases structural engineers need to analyze to design safe and reliable aircraft. Cutting design cycles down by months and creating fully back-tracing external loads to a single database. More broadly, the project demonstrates how relational database modeling and geometric reduction algorithms can be applied to large, heterogeneous scientific datasets, a problem domain of increasing relevance across engineering disciplines.

The rest of the paper is organized as follows: Section two presents the related works; Section three presents the database architecture and framework; Section four presents the schema architecture and modeling decisions; Section five presents the reduction methodology and algorithm implementation; Section six presents the results and evaluation; Section seven presents the discussions; Section eight presents the limitations of this work; Section nine presents the future works; and finally, Section ten presents the conclusions.

2. RELATED WORKS

Published work describing external loads post-processing tools is limited because most aerospace companies develop these capabilities internally. One of the few publicly available examples is the L-VIS framework described by Ünay et al. (2012) [1], which provides insight into the visualization and filtering techniques commonly used in industry. L-VIS produces 1-D and 2-D load envelopes and supports basic critical-case selection, demonstrating the type of functionality expected in modern loads environments. However, it does not incorporate automated case-reduction techniques, highlighting a gap that motivates the present work, which aims to reduce large case sets while preserving envelope behavior. Additional studies, such as the neural-network-based load-case reduction approach presented by Nazzeri and Titurus (2015) [2], show that reducing the number of load cases sent to structural analysis can materially shorten design cycles. These findings reinforce the importance of systematic and reproducible reduction methods rather than relying solely on manual or heuristic selection.

The use of a relational database for external loads processing is supported by foundational work in data management. Codd's relational model (1970) [3] defines the principles of data independence, logical consistency, and declarative querying, all of which are essential for organizing loads data produced by multiple tools and formats. These principles directly support the adoption of a normalized SQL schema to unify HDF5 outputs, F06 data, trim variables, and related metadata into a consistent structure. As the volume of loads data grows into the millions of records, performance considerations become central. Hellerstein, Stonebraker, and Hamilton (2007) [4] describe how relational database systems achieve scalable performance through indexing, query optimization, buffer management, and parallel execution. These architectural features are especially relevant for workloads involving repeated slicing and filtering of large multidimensional datasets, such as retrieving the cases that define an envelope or evaluating candidate reductions. Long-term data management needs are further emphasized by Babuji et al. (2016)[5], who describe a computational infrastructure designed for reproducible scientific workflows and provenance tracking. Their work demonstrates the importance of structured storage systems for integrating heterogeneous inputs and maintaining consistency as simulation tools and formats evolve, directly aligning with the objectives of creating a durable and extensible loads-database framework.

The algorithmic foundation for load-case reduction draws from several areas of computational theory. Chvátal (1979) [6] demonstrates that greedy heuristics can provide strong approximation guarantees for large set-reduction problems, making them suitable for identifying non-redundant cases in large loads datasets. Feige (1998) [7] further establishes that these greedy approximations are essentially optimal for the set-cover problem under standard complexity assumptions, justifying the use of greedy strategies rather than attempting more computationally expensive global optimizations. Complementing these ideas, Agarwal, Har-Peled, and Varadarajan (2004) [8] introduce coreset-based geometric approximation techniques for selecting small representative subsets that preserve the geometric structure of much larger datasets. This concept is directly relevant when approximating multidimensional load envelopes within a specified tolerance. For two-dimensional envelopes, particularly potato-plot-type analyses, the Quickhull algorithm described by Barber, Dobkin, and Huhdanpaa (1996) [9] provides an efficient method for identifying the boundary points of a convex hull. These boundary points correspond to envelope-defining load cases and can serve as an initial reduced set that may be further refined through greedy or coreset-based techniques.

3. DATABASE ARCHITECTURE AND DATA FRAMEWORK

3.1. SYSTEM REQUIREMENTS

External loads analysis in the aerospace industry has experienced substantial growth in data volume over the past two decades due to increasing model fidelity, solver advancements, and expanded computational capacity. As a result, post-processing and data management have become primary constraints in the external loads workflow. In addition to scale, modern analyses exhibit variability in output structure and format, with different load loops and configurations generated by multiple software tools and solver versions; a standardized database is required. Older, flat-file workflows that were designed for smaller, more uniform datasets do not enforce structural consistency across revisions or analysis types and rely heavily on ad-hoc scripting. A structured data management system is therefore required to support scalable, consistent, and reproducible post-processing across diverse analysis outputs.

3.1.1. FUNCTIONAL REQUIREMENTS

The system must support the following operations:

- Retrieve and aggregate load data across multiple loops and conditions using user-defined grouping and filtering criteria for comparative visualization.
- Support multi-dimensional envelope extraction across user-selected degrees of freedom, components, and analysis configurations.
- Identify envelope-defining conditions while maintaining traceability to the originating aircraft-state and control-surface parameters.
- Compute a reduced subset of conditions such that the resulting envelope remains within a user-defined tolerance of the full dataset.
- Support correlation analysis between load responses and associated aircraft state variables.
- Provide complete traceability from each load measurement to its originating condition and time step.

3.1.2. NON-FUNCTIONAL REQUIREMENTS

In addition to functional capabilities, the system must satisfy the following properties:

- Serve as an authoritative and centralized repository for external loads outputs independent of solver type or version.
- Enforce version control and update governance to ensure that users query the correct and most current dataset without requiring prior revision knowledge.
- Maintain acceptable query latency suitable for interactive envelope extraction and multi-dimensional plotting.
- Enforce data integrity through structured relationships and controlled updates.
- Provide schema extensibility to accommodate future analysis types and additional state variables.

3.2. PERFORMANCE CONSTRAINTS

If query latency or usability does not meet practical engineering needs, users are likely to bypass the system and develop independent post-processing scripts. This would result in data scattered across multiple tools with no reliable way to trace results back to the original condition, duplicated workflows, and reduced reproducibility in critical-case selection. Therefore, performance and usability are essential requirements rather than optional enhancements.

4. SCHEMA ARCHITECTURE AND MODELING DECISIONS

The database schema was designed to enforce an authoritative, traceable representation of external loads data while supporting multiple analysis types with differing structural characteristics. The modeling approach separates high-volume numerical load measurements from contextual metadata to reduce redundancy, maintain referential integrity, and enable scalable query performance across large datasets.

4.1. DIMENSIONAL STRUCTURE AND DATA SEPARATION

The schema follows a dimensional modeling approach in which load measurements are stored in fact tables and descriptive attributes are stored in dimension tables. Fact tables contain the high-cardinality numerical load data, while dimension tables store contextual metadata required to interpret each load value, including aircraft configuration, flight condition parameters, spatial point definitions, measurement direction, and time step information.

This separation prevents duplication of descriptive metadata across millions of load rows. For example, aircraft-level attributes or condition parameters are stored once in their respective dimension tables and referenced through foreign keys. Without this separation, metadata such as Mach number, altitude, or aircraft mass properties would be repeated for every load record, increasing storage requirements and introducing potential inconsistency during updates.

Shared dimension tables include:

- Dim_Aircraft – aircraft-level metadata applicable across analyses.
- Dim_Component – component-level definitions used for grouping and filtering.

- Dim_Point – monitor and stress point metadata, including coordinate system identifiers.
- Dim_Measure – one row per degree of freedom (Fx, Fy, Fz, Mx, My, Mz).
- Dim_Time – time-step definitions for dynamic analyses.
- Dim_Condition – flight-state, mass property, and control-surface parameters describing each operating condition.

These dimension tables are designed to be reusable across multiple fact tables and analysis types.

4.1.1. FACT TABLE GRAIN AND ANALYSIS-SPECIFIC SEPARATION

Separate fact tables were implemented for static flight loads, dynamic flight loads, and static ground loads. At the schema level, these tables are structurally identical: each row represents a single degree-of-freedom load value at a specific spatial point for a specific condition and time step.

For static analyses, the time dimension is included but assigned a default value (time = 0). Dynamic analyses populate the time dimension with solver-provided time step values. As a result, the grain definition is consistent across fact tables, and condition identity is determined by the combination of condition parameters and time.

Although the table structures are identical, separate fact tables were intentionally maintained for workflow and operational reasons rather than structural necessity. In practice, each analysis type is post-processed independently, often by different engineers. Individual analyses may generate smaller, temporary databases during post-processing before validation and integration into the centralized repository. Maintaining separate fact tables allows these workflows to remain decoupled while preserving a consistent schema.

This separation also improves query clarity and performance isolation. Many queries target a single analysis type (e.g., static flight loads only), and explicit table segmentation allows direct filtering without relying solely on an analysis type attribute within a single unified table. Cross-analysis comparisons remain possible when required, but the majority of engineering workflows operate within a single analysis domain. Only final envelope aggregation typically requires combining data across all analysis types.

By structuring the schema this way, the design balances structural uniformity with practical workflow segmentation, enabling modular ingestion and targeted querying without introducing unnecessary complexity into condition modeling or measurement storage. A simplified version of the schema is shown in **Error! Reference source not found.**. The diagram illustrates the relationship between fact tables and dimension tables, highlighting how load values are linked to conditions, spatial points, and measurement definitions through foreign key relationships. It also emphasizes the consistent grain across fact tables and the role of shared dimension tables in maintaining a unified structure despite workflow segmentation.

4.1.2. MEASUREMENT REPRESENTATION (LONG FORMAT MODELING)

Load components are stored in a long-format representation, with each row representing a single degree-of-freedom load value defined by a point, condition, and, for dynamic analyses, a time step. The degree of freedom is specified through the Dim_Measure table, which contains six entries corresponding to Fx, Fy, Fz, Mx, My, and Mz.

This design enables uniform query patterns across all load components. Filtering by degree of freedom is performed through a single predicate on MeasureID, allowing consistent envelope extraction, aggregation, and reduction logic without measure-specific column handling. It also centralizes measurement metadata (naming, units, conventions) in a single dimension table.

The primary tradeoff of this approach is increased row count relative to a wide-table representation containing six load columns per row. However, the long-format model improves extensibility, simplifies querying logic, and maintains consistent dimensional relationships across all analyses.

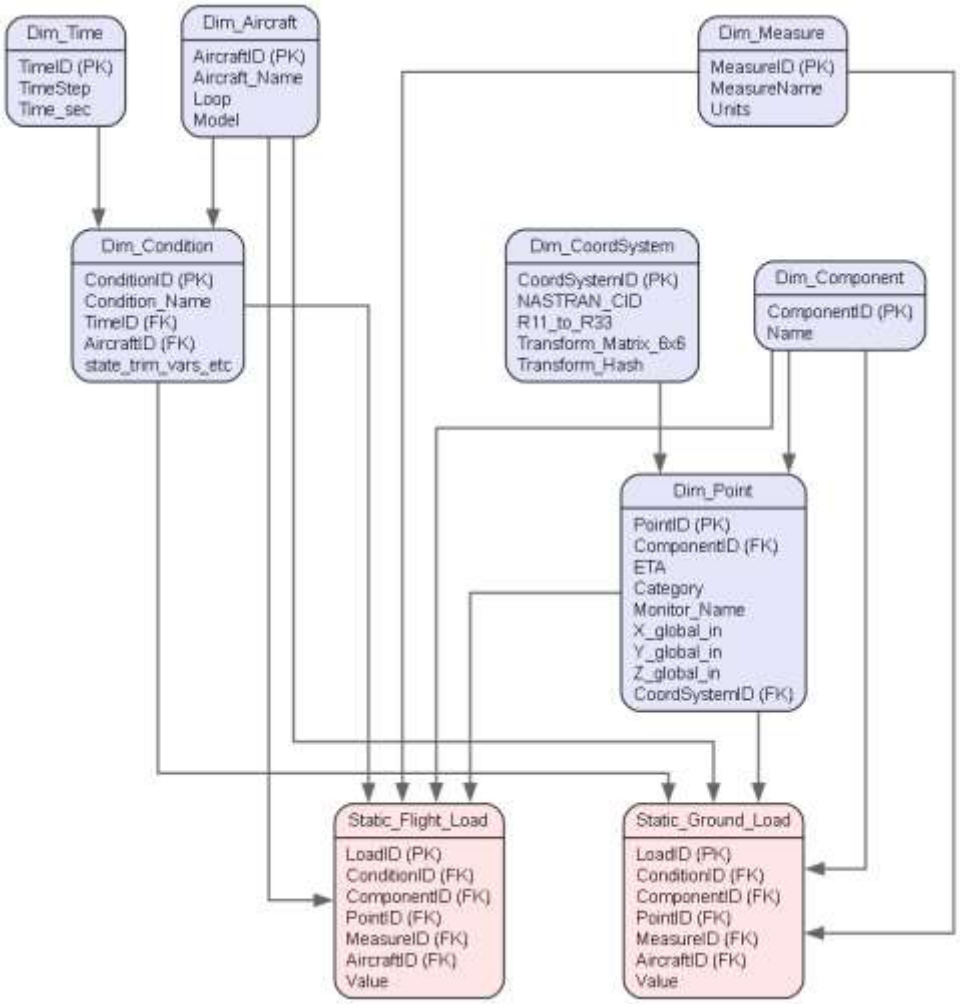

FIGURE 1 Database Schema

4.1.3. CONDITION MODELING AND CANONICAL CORRELATION KEY

The Dim_Condition table serves as the canonical correlation key for all load records. A condition represents a unique operating case defined by the combination of condition name, time step, and the aircraft context (AircraftID). Each distinct condition instance is assigned a UUID (ConditionID), which is referenced by all fact tables.

Time is embedded within condition identity rather than modeled as a separate foreign key in the fact tables. As a result, fact tables reference only ConditionID, maintaining a consistent grain definition across static and dynamic analyses while avoiding composite joins on condition and time during envelope extraction and reduction queries. This was done to solve issues with dynamic analyses; each solver time step is treated as a distinct condition instance. If time were a foreign key, correlated data would not be traceable.

Each condition record is uniquely defined by a composite constraint on ConditionName, TimeID, and AircraftID. This guarantees that duplicate condition-time entries cannot exist within the same aircraft context. A UUID is used as a stable surrogate key for joins and indexing, while the composite constraint enforces domain-level identity, ensuring deterministic traceability from every load record to a single, well-defined operating state.

4.1.4. REFERENTIAL INTEGRITY AND TRACEABILITY

Traceability is enforced at the database level through foreign key constraints (Bagui and Earp, 2023) [10] defined on all fact tables, including static flight loads, dynamic flight loads, and static ground loads. Each load record references valid entries in Dim_Aircraft, Dim_Component, Dim_Condition, Dim_Point, Dim_Measure, and Dim_Time. These foreign key columns are defined as NOT NULL, ensuring that no load record can exist without complete contextual metadata.

This design guarantees referential integrity: a load value cannot reference a non-existent aircraft, condition, spatial point, measurement definition, or time entry, and orphaned measurement records are structurally prevented. As a result, every load measurement can be deterministically traced to:

- The aircraft configuration and analysis context
- The originating condition definition
- The spatial point and associated coordinate system metadata
- The associated degree of freedom (Fx, Fy, Fz, Mx, My, Mz)
- The time step associated with the condition (for static analyses, this is time = 0)

By enforcing these relationships within the database engine rather than relying solely on ingestion discipline, the schema establishes an authoritative and reproducible foundation for envelope extraction, correlation analysis, and reduction workflows.

4.1.5. COORDINATE SYSTEM METADATA

Coordinate system metadata is modeled as a dedicated dimension table (Dim_CoordSystem) rather than embedded directly within point records. The Dim_Point table stores global spatial location and references a coordinate system through a foreign key. This separation distinguishes physical location from load orientation.

The Dim_CoordSystem table stores the solver-defined coordinate system identifier, associated rotation matrix, and any transformation matrix required to represent loads in the external reporting convention. Each coordinate system entry is uniquely defined to prevent duplication and ensure consistent interpretation across multiple points. This structure enables reuse of coordinate definitions, preserves referential integrity, and supports deterministic interpretation of load direction without duplicating orientation metadata across millions of fact rows.

4.1.6. DESIGN TRADEOFFS

Several modeling tradeoffs were evaluated during schema development:

- A single unified fact table versus separate fact tables per analysis type.
- Wide-table measurement storage versus long-format modeling.
- Subtyped condition tables versus a universal condition dimension.

The final design prioritizes clear grain definition, traceability, and workflow flexibility over strict normalization minimalism. This structure supports both current analyses and future extensions without requiring structural redesign of the database.

4.1.7. KEY DESIGN DECISIONS

When designing the database, plotting with specific filters (aircraft, component, degree of freedom, etc.) was a primary consideration. Once plotting routines were developed, performance limitations became clear.

The original min/max envelope queries were executed for every component and degree of freedom requested. In a dataset with one aircraft and one loop, the Static_Flight_Load table contained approximately 2.5 million rows. For an aircraft with 20 components and six degrees of freedom, this results in 20 × 6 max queries and 20 × 6 min queries. Without indexing, each query required scanning a large portion of the fact table before grouping by point and computing the extreme values. Plotting times were typically 10–15 minutes and would scale poorly as the database grew. To address this, a composite index was introduced on the primary filtering columns:

$$CREATE\ INDEX\ IF\ NOT\ EXISTS\ idx\_fact\_load\_lookup$$
$$ON\ Static\_Flight\_Load(AircraftID, ComponentID, MeasureID, PointID);$$

This allows the database to directly locate the relevant subset of rows instead of scanning the entire table for each query. Similar indexes were created for the dynamic and ground load tables.

After indexing, plotting time decreased from approximately 10–15 minutes to 10–20 seconds for the same workload. This improvement was repeatable. Without indexing, the database would not meet practical performance expectations, and engineers would likely revert to separate filtering workflows, undermining traceability and centralized data control.

4.2. COORDINATE SYSTEM HANDLING

4.2.1. SEPARATION OF SPATIAL LOCATION AND LOAD ORIENTATION

External load interpretation requires explicit and traceable management of coordinate systems. If coordinate system definitions are not preserved, downstream users may assume incorrect load directions, leading to misinterpretation of bending and torsional behavior. Because external loads reporting heavily relies on comparison, particularly between symmetric components such as left and right wings, consistent orientation is critical. External loads are typically reported using a convention where $F_z$is defined normal to the surface and $F_y$is aligned along the wing following the load reference line. The $F_x$direction is then defined pointing aft. Under this convention, the right wing forms a right-hand-rule coordinate system, while the left wing does not. This approach simplifies visual comparison and sanity checking across symmetric components, even though it does not strictly enforce a right-hand-rule definition for each local coordinate system.

To prevent ambiguity, spatial location and load orientation are modeled separately. The Dim_Point table stores the global location of each monitor or stress point (X_global_in, Y_global_in, Z_global_in) along with plotting metadata such as ETA and category. These coordinates represent physical location only and are not used to define load orientation or perform moment transfers. External load orientation is instead defined through a foreign key reference to the Dim_CoordSystem table.

4.2.2. COORDINATE SYSTEM DEFINITION AND UNIQUENESS

The Dim_CoordSystem table stores the complete definition required to interpret load direction. This includes the NASTRAN coordinate system identifier (NASTRAN_CID), the origin of the coordinate system in global coordinates (A1–A3), and the rotation matrix components (R11–R33) defining the local-to-global transformation, and a multiplication matrix if the external load is not reported in a typical right-hand-rule coordinate system.

In addition to the solver-defined coordinate system, an optional 6×6 transformation matrix is stored to represent any multiplication matrix applied to convert solver-native orientation into the standardized external loads reporting convention. This transformation allows, for example, consistent normal force. $\boldsymbol{F_z}$reporting for both left and right wings even when the underlying solver coordinate systems differ.

Each coordinate system entry is uniquely identified by the combination of NASTRAN_CID and a hash of the transformation matrix (Transform_Hash). This ensures that any change in orientation logic results in a distinct coordinate system definition. Multiple points may reference the same coordinate system entry, preventing duplication and maintaining referential integrity.

4.2.3. TRANSFORMATION CHAIN AND EXTERNAL LOADS CONVENTION

The transformation logic is explicit and reversible. External loads originate in a NASTRAN-defined local coordinate system. A multiplication matrix may then be applied to convert those loads into the external reporting convention used for comparison and visualization. To recover global orientation, the inverse of the multiplication matrix can be applied, followed by the rotation matrix stored in Dim_CoordSystem.

No transformations are performed within SQL. The database stores load magnitudes exactly as reported in the external loads convention, along with all metadata required to interpret or transform them deterministically outside the database. Moment transfer operations (e.g., $\mathbf{r} \times \mathbf{F}$) are not performed within the database and are handled separately during ingestion or post-processing when required. This separation ensures that the database preserves orientation traceability without embedding engineering transformations into query logic.

4.2.4. ENGINEERING RISK MITIGATION AND DATA INTEGRITY

Improper handling of coordinate systems can directly affect structural interpretation. If load directions are assumed incorrectly, bending moments may be interpreted as torsional effects or vice versa. In symmetric components such as wings, inconsistent orientation between left and right sides can obscure model sanity checks or lead to incorrect structural sizing decisions. In extreme cases, such misinterpretation could contribute to undersizing and structural failure.

By storing global point location separately from orientation, and by modeling coordinate systems as a dedicated dimension with explicit transformation metadata, the database ensures that load direction is fully re-constructible without reopening the original NASTRAN model. This design preserves engineering intent, while also supporting consistent plotting and reporting conventions, and maintains deterministic traceability of load orientation across all analyses.

4.3. DATA INGESTION AND NORMALIZATION PIPELINE

4.3.1. MULTI-SOURCE INGESTION REQUIREMENT (.f06 + .h5)

A key design driver for ingestion is that the full set of information required for traceable external loads post-processing is not contained in a single solver output file. NASTRAN's .h5 results output is structured and consistent, but it does not include several pieces of context that are required for engineering usability. NASTRAN does not inherently care about human-readable naming conventions; it organizes results around subcase numbers (e.g., "SUBCASE 1000") rather than condition names that carry meaning to downstream users. The .f06 output includes additional printed context (such as condition naming and weight configuration) that is intentionally added for engineering interpretation and is commonly required in the aerospace workflow, even if it may not be relevant in other industries that use NASTRAN.

Although much of the same information in the .h5 may exist somewhere in the .f06, it is a large text-based output and typically requires pattern-based extraction. That approach is more fragile across solver versions because small formatting differences (spacing, wording, line breaks) can break text parsing logic. The .h5 file provides a more stable and structured source for numeric results and solver-organized datasets, so the ingestion pipeline combines both sources. .f06 is used for engineering-context metadata, and .h5 is used for structured numeric results and solver-native organization.

4.3.2. CONFIGURATION-DRIVEN MAPPING FILES

The ingestion pipeline uses configuration mapping files (.csv) to translate solver-oriented naming into an engineer organization and to define what should be extracted, ignored, or renamed for downstream use. These mappings include point catalogs, condition-summary/trim variable definitions, and stress-point generation metadata (interpolation definitions and multiplication matrices). In practical terms, these mapping files act as the contract between solver output and the relational schema. They define how raw solver data becomes consistent dimension and fact table entries.

These mappings are maintained as .csv files rather than being hard-coded because they are specific to the aircraft and loop. Different aircraft and loop configurations may use different trim variables, different stress points, or different point naming conventions. Maintaining mappings outside of the code allows the ingestion logic to remain stable while enabling updates to aircraft/loop-specific definitions without modifying the ingestion scripts.

4.3.3. PAIRING LOGIC AND EXTRACTION STRATEGY

Ingestion is performed on one paired .f06 and .h5 file set at a time. Pairing is based on shared filenames with different extensions, ensuring that subcase numbering aligns within that run. Within a paired dataset, the ingestion process identifies the NASTRAN subcase number and uses it as the primary link between the .f06 and .h5 contents. The .f06 parsing extracts the metadata associated with each subcase (including engineer-provided condition naming and other printed context), while the .h5 parsing extracts structured results for the corresponding subcase.

This approach avoids ambiguity introduced by subcase number reuse across separate NASTRAN runs. Subcase “1000” is only meaningful within a single analysis output set, so the pipeline processes one .f06/.h5 pair at a time and then moves to the next paired set.

At the database level, aircraft/loop identity is represented through Dim_Aircraft, and all extracted per-case metadata from .csv and .f06 is stored in Dim_Condition. Numeric load data, TRIM/state variables, coordinate systems, and solver domain mappings are extracted from the .h5 and mapped into the corresponding schema tables.

4.3.4. NORMALIZATION AND POST-PROCESSING TRANSFORMATIONS

The ingestion pipeline performs normalization steps required to convert solver output into the database representation used for plotting and reduction workflows. A primary example is increment superposition. In external loads workflows, a full maneuver may be represented by combining multiple NASTRAN cases (increments). For example, a 1.0g trim case may be combined with a 1.5g maneuver increment to form a total 2.5g condition. This approach reflects how trim strategy differs between baseline and maneuver increments (e.g., stabilizer carrying trim in the 1.0g condition while elevator carries increment response), and the ingestion step combines these increments to produce a consistent “total maneuver” load and state definition. This same concept applies to both loads and relevant state/control variables when those values are stored in Dim_Condition.

Stress points are also generated as an optional transformation step. Monitor points are the primary points provided directly by NASTRAN and are typically selected to align with model boundaries and structural interfaces. Stress points are computed during ingestion by interpolating from monitor points and applying multiplication matrices when required. Stress points are stored as separate entries in Dim_Point and are labeled by category (e.g., “Stress” vs “Monitor”), even if the physical location is similar, so that downstream users can immediately distinguish interpolated values from raw solver values. A “monitor-only” mode is supported for model debugging and quick validation, where interpolation is skipped to reduce processing time and focus only on raw NASTRAN outputs.

4.3.5. AIRCRAFT METADATA CAPTURE AND INITIALIZATION

The ingestion system begins with explicit user-provided metadata to define the analysis context. At launch, the user is prompted for:

- Aircraft Name
- Aircraft Loop
- Aircraft Model
- Load Type (flight or ground)
- Monitor-points-only flag

These values populate Dim_Aircraft and define the analysis domain before any solver data is parsed. By explicitly collecting this metadata at runtime, the ingestion process ensures that each dataset is uniquely identified at the aircraft/loop level prior to inserting condition or load records. This prevents ambiguity when subcase numbering is reused across separate NASTRAN runs and supports controlled integration into the broader database.

4.3.6. BATCH INSERTION AND MEMORY MANAGEMENT

Fact insertion is performed using batch operations within explicit transactions. Database initialization (schema and shared dimension tables) occurs once at the start of ingestion, and the database connection remains open for the duration of the run.

Processing then occurs per paired .f06/.h5 file set. Each file pair is parsed, transformed, and inserted independently. Fact rows for that pair are accumulated in memory, inserted using executemany(), and committed before moving to the next pair. This limits peak memory usage and prevents large dynamic datasets from exhausting available RAM.

INSERT OR IGNORE is used to allow safe re-execution without duplicate key violations. After each file pair is committed, intermediate objects are explicitly released, and garbage collection is invoked to maintain stable memory usage.

This approach preserves transactional integrity at the file-pair level while maintaining performance through batched inserts.

4.4. SCALABILITY AND STORAGE GROWTH

Transitioning from flat text-based workflows to a relational database increases storage requirements. The original flat-file process (text files and CSVs) occupied approximately 250 MB, whereas the structured relational database grew to approximately 1.47 GB after ingestion and indexing. This increase is expected and results from deliberate modeling decisions rather than inefficiency.

Several factors contribute to storage growth. First, the use of UUID surrogate keys increases row width relative to using solver-native identifiers directly. Second, the long-format modeling approach stores one row per degree of freedom rather than six load components in a single wide row. While this increases row count, it simplifies querying, filtering, and extensibility. Third, indexing adds additional B-tree structures to accelerate lookup performance. Finally, dynamic analyses significantly increase data volume because each time step is represented as a distinct condition instance. For dynamic cases, row counts may scale on the order of hundreds of times larger than static flight data (e.g., 360× if 360 time steps are present per maneuver). Growth behavior is predictable. Static flight datasets scale approximately linearly with additional aircraft or loops. Dynamic datasets scale multiplicatively with the number of time steps per condition. This behavior is understood and consistent with the grain definition established in Schema Architecture and Modeling Decisions.

At the current scale (millions of fact rows per aircraft/loop), SQLite remains appropriate as a single-user analytical database. The ingestion and indexing strategies described above in Key Design Decisions and Data Ingestion and Normalization Pipeline maintain acceptable query latency for envelope extraction and plotting. However, the long-term architecture anticipates the possibility of consolidating multiple aircraft- and loop-specific databases into a larger centralized repository. At that scale—particularly if concurrent access or tens of millions of dynamic rows become standard—a migration to a client-server relational system may be appropriate.

The current design therefore balances immediate usability and simplicity with a clear path toward future scaling. Storage growth is an intentional tradeoff in exchange for traceability, normalized structure, and deterministic query performance.

5. REDUCTION METHODOLOGY AND ALGORITHM IMPLEMENTATION

5.1. PROBLEM DEFINITION AND REDUCTION OBJECTIVE

External loads analysis produces thousands of flight conditions, each containing load values across multiple components, spatial points, and six degrees of freedom (Fx, Fy, Fz, Mx, My, Mz). Downstream structural engineers do not evaluate every condition individually. Instead, they rely on envelope behavior — the maximum and minimum load values at each (Component, Point, DOF) location — to size and validate the structure. The computational problem addressed in this work is the following: Identify the smallest subset of flight conditions whose load envelope remains within a user-defined tolerance of the envelope produced by the full dataset.

5.2. REDUCTION DOMAIN AND DATASET SCOPING

The database schema is designed to store data for multiple aircraft and multiple load loops within a single system. However, envelope extraction and reduction are only meaningful when performed within a clearly defined dataset scope.

Reduction is therefore executed within a specific Aircraft and Loop selection. These attributes act as required partition keys for all envelope and reduction queries. Conditions from different aircraft cannot be mixed, and cross-aircraft reduction is not allowed. Even within a single aircraft, mixing loops without explicit intent may produce incorrect results. Different loops may correspond to different model revisions, updated component definitions, modified stress points, or changes in load extraction methodology.

If Aircraft and Loop filters are not applied, condition names may be duplicated across datasets, component and point catalogs may differ, and envelope-defining cases may be drawn from unintended analyses. This can result in outputs that appear numerically valid but reference incorrect source data, breaking traceability and invalidating reduction results.

Once the Aircraft/Loop domain is fixed, the workflow determines the available components and points in that dataset. The user may then optionally apply additional filters (for example, restricting to specific components, spanwise stations, or degrees of freedom). All envelope requirements and subsequent reduction steps are generated strictly within this scoped dataset.

Special engineering cases may require comparing multiple loops (e.g., using a newer loop to show bounding relative to an earlier certified loop). Such cross-loop validation requires explicit documentation and is treated as out of scope for this project. The reduction framework presented here assumes a single Aircraft/Loop domain per execution to preserve deterministic correctness and traceability.

5.3. BASELINE ENVELOPE AND REQUIREMENT DEFINITION

Envelope behavior is represented through a set of reduction requirements derived from the baseline dataset. The most common requirements correspond to the maximum and minimum load values at each (Component, Point, DOF) location. Each location therefore produces two independent extrema: maximum value and minimum value. If there are P spatial points and six degrees of freedom per point, the baseline envelope produces $2 \times P \times 6$ independent one-dimensional requirements. These extrema are anchored to the original full dataset and remain fixed throughout the reduction process.

Reduction correctness is evaluated relative to these baseline extremes. The reduced maximum must not fall below the baseline maximum by more than the specified tolerance, and the reduced minimum must not exceed the baseline minimum by more than the specified tolerance. Tolerance is therefore applied directionally relative to the baseline extreme, and the baseline envelope does not shift as conditions are removed.

In addition to one-dimensional extrema, envelope requirements may also be derived from multi-dimensional load relationships. In external loads analysis this commonly appears as two-degree-of-freedom envelope plots, often referred to as potato plots, where combinations of loads define the envelope boundary. When requested by the user, these relationships are analyzed to identify the conditions that form the convex hull of the dataset in the selected two-DOF space. The hull-defining conditions become additional envelope requirements that must also be preserved during reduction.

In external loads workflows, a tolerance of approximately 3% is typically acceptable. Differences of this magnitude commonly arise from rounding effects, solver linearization differences, and conservative assumptions embedded within the model. Because these variations are expected in computational analysis, the tolerance threshold is user-configurable so that engineering judgment can determine the appropriate acceptable deviation.

5.4. COMPUTATIONAL FORMULATION

Removing the aerospace context, the problem becomes selecting the minimal subset of items that preserves the maximum and minimum envelope values across multiple dimensions within a bounded tolerance. In the external loads setting, each flight condition can be interpreted as covering a subset of envelope requirements if its load values satisfy the tolerance-adjusted thresholds or geometric envelope constraints associated with those requirements. The reduction objective is therefore to select the smallest subset of conditions such that all envelope requirements remain satisfied.

This formulation aligns with the classical set cover problem, which is NP-complete. Because the goal of this work is to produce a practical engineering tool capable of operating on large datasets, the solution strategy does not attempt exact global optimization. Instead, a deterministic greedy approximation approach is used to balance computational efficiency with engineering sufficiency.

The contribution of this work is not the introduction of a new combinatorial algorithm. Rather, it lies in integrating a relational database architecture with a tolerance-aware reduction framework. The system combines a database schema specifically designed for aerospace external loads data, efficient SQL-based envelope extraction over indexed fact tables, and a deterministic greedy set-cover approach that operates directly on engineering-defined envelope requirements. Because the reduction algorithm operates on requirements generated through relational queries and geometric envelope analysis, database schema design, indexing performance, and envelope extraction procedures become integral components of the reduction process rather than independent implementation details.

5.5. REDUCTION FRAMEWORK OVERVIEW

The reduction workflow is executed through a user-generated configuration file that defines the dataset scope, envelope definition, and optional geometric envelope analysis. This configuration drives the full reduction process and allows engineers to control which portions of the dataset participate in envelope construction and case selection.

The reducer is configured using a structured input file (reducer_input.txt, Appendix A) containing several input blocks. The Basic Input Block defines the primary dataset scope and system configuration, including the database path, aircraft identifier, load loop, analysis type (static flight, static ground, dynamic flight, or combinations), and the envelope tolerances used during reduction. This block establishes the global context for all subsequent queries and computations.

The Condition Block allows the user to restrict the set of flight conditions considered during reduction. Conditions are filtered using user-defined string matches applied to the Condition_Name field in the database. In practice, this allows engineers to isolate groups of conditions corresponding to payload states, fuel configurations, airspeeds, or other naming conventions embedded in the simulation output.

The Components Block specifies which structural components participate in the reduction and which degrees of freedom (Fx, Fy, Fz, Mx, My, Mz) should be considered for each component. Multiple components may be defined simultaneously, and each component can include a different subset of degrees of freedom depending on the structural behavior being evaluated.

The ETA Block allows optional point reduction within selected components. In some analyses, only a subset of spatial locations is required for envelope verification, and this block provides a mechanism for restricting the reduction domain to those points.

Finally, the Potato Plot Block defines optional two-degree-of-freedom envelope comparisons. Each entry specifies a component, a spatial point, and a pair of degrees of freedom that should be analyzed together to form a combined load envelope. These specifications trigger additional geometric envelope analysis beyond the standard one-dimensional envelope extraction.

These filtering stages significantly reduce the working dataset before envelope analysis begins. As the database grows, the size of the underlying tables will increase, while the final query outputs used for envelope generation remain largely determined by the user-specified configuration. In practice, the envelope extraction queries return only a few hundred envelope locations regardless of the total database size.

The first analytical step is the extraction of the one-dimensional baseline envelope. This step is executed entirely within SQL. Using grouped queries on the filtered dataset, the database identifies the maximum and minimum load values for each (Component, Point, DOF) combination. The query then joins back to the underlying data to identify the flight condition responsible for each extreme value. The resulting set of conditions forms the witness set for the one-dimensional envelope and provides the baseline requirements used during reduction.

Two-dimensional envelope requirements are generated only when potato plot specifications are present in the configuration file. Because SQLite does not provide native convex-hull functionality, the database first returns all load values for the specified (Component, Point, DOF_1, DOF_2) combinations. These values are then processed in Python using scipy.spatial.ConvexHull to determine the convex hull of the dataset in the selected two-degree-of-freedom space. The hull vertices identify the conditions that form the boundary of the potato plot envelope.

Hull vertices may contain redundant points that lie along nearly collinear segments of the boundary. To prevent unnecessary growth in the reduction problem, a simplification step is applied to the hull before requirements are generated. This simplifier removes boundary points that fall within user-defined angular deviation and directional projection tolerances while preserving the overall envelope shape. The remaining vertices define the two-dimensional boundary requirements that must be preserved during reduction.

The full set of reduction requirements is then formed by combining the one-dimensional envelope extrema with any additional two-dimensional boundary requirements generated from potato plot analysis. The number of requirements therefore depends on the configuration and the dataset being analyzed. One-dimensional envelope extraction typically produces several hundred requirements, while two-dimensional envelope requests may add additional requirements depending on the number of potato plots specified and the complexity of the resulting convex hull boundaries.

Once the requirement set is constructed, the reduction algorithm selects the minimal subset of flight conditions needed to satisfy all requirements. The greedy set-cover algorithm used for this process is described in the Greedy Reduction Algorithm, followed by a reverse-delete refinement stage that removes redundant conditions while maintaining envelope coverage.

After reduction is complete, the results are verified against the original baseline envelope. For one-dimensional envelopes, the reduced dataset is compared directly to the baseline maxima and minima to confirm that all values remain within the specified tolerance. For two-dimensional envelopes, verification is performed using a directional projection check that evaluates the

reduced envelope against the baseline envelope at regular angular increments. In the current implementation, projections are evaluated every five degrees to confirm that the reduced envelope remains within tolerance across the entire load space.

Together, these steps form a deterministic reduction framework that integrates database-driven envelope extraction, geometric envelope analysis, and combinatorial case selection to produce a reduced set of flight conditions while preserving the engineering envelope behavior of the full dataset.

5.6. REQUIREMENT GENERATION

The reduction algorithm operates on a set of requirements derived from the baseline envelope of the dataset. A requirement states that the load value produced by the reduced dataset must remain within a specified tolerance of the corresponding baseline envelope value.

Requirements are generated from two sources: one-dimensional envelope extrema and optional two-dimensional envelope boundaries. The one-dimensional extrema define the baseline envelope for each degree of freedom independently, while two-dimensional envelope boundaries capture combined load behavior using potato plot analysis. Both requirement types are constructed prior to reduction and combined to form the constraint set that the reduced condition set must satisfy.

5.7. ONE-DIMENSIONAL ENVELOPE REQUIREMENTS

The baseline envelope is first computed for each user-defined component, spatial point, and degree of freedom. For every (Component, Point, DOF) combination, the system identifies the maximum and minimum load values across all conditions within the scoped dataset. These extrema are extracted directly from the database using grouped SQL queries, which also return the conditions responsible for producing each extreme value. These conditions are referred to as the baseline conditions for the envelope.

Each envelope location therefore produces two independent requirements, a maximum requirement and a minimum requirement. If there are P spatial locations, and the user specifies a set of degrees of freedom for each component, the number of one-dimensional requirements becomes:

$$2 \times (\text{Component, Point, DOF})$$

For example, a user specified dataset containing 348 envelope values (component, point, DOF), produces 696 one-dimensional requirements.

Each requirement is defined relative to the baseline extreme value and a user-specified tolerance. The tolerance is computed using the following piecewise functions:

$$T_{max} = \begin{cases} B_{max}(1-\tau), B_{max} \geq 0 \\ B_{max}(1+\tau), B_{max} < 0 \end{cases}$$
$$T_{min} = \begin{cases} B_{min}(1-\tau), B_{min} \leq 0 \\ B_{min}(1+\tau), B_{min} > 0 \end{cases}$$

where $B_{\max}$ and $B_{\min}$ are the baseline maximum and minimum values, and τ is the user-specified tolerance.

The reduced dataset must then satisfy:

$$R_{\max} \geq T_{\max}$$
$$R_{\min} \leq T_{\min}$$

The piecewise function is used to handle edge cases where a maximum value is negative, or a minimum value is positive. Due to the database performing the envelope aggregation directly, the Python reduction code receives only the reduced set of baseline extrema and the corresponding baseline conditions, rather than the full raw dataset.

5.8. TWO-DIMENSIONAL ENVELOPE REQUIREMENTS

While one-dimensional envelopes capture independent load extrema, structural design often depends on combined loading behavior. To capture these interactions, the system supports optional two-degree-of-freedom envelope analysis using potato plots.

Each potato plot specification defines a (Component, Point) location and two degrees of freedom that should be evaluated together. For each specification, the database retrieves all load pairs.

$$(DOF_1, DOF_2)$$

For every condition in the filtered dataset. Because SQLite does not provide built-in convex hull functionality, the geometric envelope is computed within Python using the scipy.spatial.ConvexHull algorithm.

The convex hull identifies the boundary conditions that define the outer envelope of the load space. However, convex hulls often contain redundant vertices that lie along nearly collinear segments of the boundary. Retaining all such points would unnecessarily increase the number of reduction requirements without improving envelope fidelity.

To address this, a hull simplification step is applied before requirement generation. The simplification algorithm iteratively evaluates triplets of adjacent hull vertices $A \rightarrow B \rightarrow C$. If the angular deviation between the vectors $A \rightarrow B$ and $A \rightarrow C$ falls within a user-defined angular tolerance, vertex B is removed. The algorithm then advances along the boundary and continues evaluating subsequent triplets.

If two vertices are removed consecutively, an additional verification step is performed to ensure that the simplified hull still remains within tolerance. In this case, the algorithm reevaluates the relationship between the previous retained vertex and the second consecutively removed vertex before confirming the removal. For example, if B and C are out of tolerance from the local angular deviation check, another check is done checking $A \rightarrow C \rightarrow D$. If the angular deviation between the vectors $A \rightarrow C$ and $A \rightarrow D$ falls outside a user-defined angular tolerance, vertex C is added back into the simplified hull.

To prevent one degree of freedom from dominating the angular calculation, the load data is normalized prior to simplification using range scaling (min–max normalization):

$$\text{normalized} = \frac{value - min}{max - min}$$

This normalization ensures that angular deviation reflects geometric envelope shape rather than differences in load magnitude. After angular simplification, a directional projection verification step is performed. The simplified hull is compared against the original hull by projecting the envelope at regular angular increments. In the current implementation, projections are evaluated every five degrees. If any removed hull vertex lies outside the user-defined projection tolerance when compared to the simplified hull, that vertex is restored.

The remaining vertices define the simplified geometric envelope boundary. Each retained hull vertex becomes a two-dimensional boundary requirement. Instead of enforcing exact reproduction of the hull vertex, the requirement is defined using a normalized Euclidean tolerance region around the vertex:

$$d = \sqrt{\left(\frac{\Delta DOF_1}{DOF_{1,vertex}}\right)^2 + \left(\frac{\Delta DOF_2}{DOF_{2,vertex}}\right)^2}$$

A reduced condition satisfies the requirement if it lies within this tolerance region of the baseline vertex. This normalized formulation avoids scale bias and prevents one degree of freedom from dominating the distance calculation. Because nearby vertices may share overlapping tolerance regions, a single reduced condition may satisfy multiple two-dimensional requirements.

5.9. COMBINED REQUIREMENT SET

The final requirement set used by the reduction algorithm is the union of the one-dimensional and two-dimensional requirement sets:

$$R = R_{1D} \cup R_{2D}$$

The number of one-dimensional requirements depends primarily on the number of components, spatial points, and degrees of freedom selected by the user. Two-dimensional requirements depend on the number of potato plot specifications and the complexity of the resulting convex hull boundaries.

For example, a typical dataset may produce approximately 696 one-dimensional envelope requirements. A single potato plot envelope might produce an initial convex hull containing more than twenty vertices, which after simplification may be reduced to roughly eight boundary requirements. Additional potato plot specifications would increase the number of two-dimensional requirements proportionally.

The combined requirement set forms the constraint system used by the reduction algorithm. The next section describes how these requirements are mapped to flight conditions and solved using a greedy set-cover formulation to identify the minimal subset of conditions that preserves the envelope behavior of the full dataset.

5.10. REQUIREMENT COVERAGE MAPPING

Once the requirement set has been generated, the next step is determining which flight conditions satisfy those requirements. Each requirement corresponds to a constraint derived from the baseline envelope, and each flight condition represents a potential candidate for reproducing that envelope behavior. The reduction process therefore requires identifying the relationships between conditions and the requirements they satisfy.

These relationships form the input to the reduction algorithm. For every condition in the filtered dataset, the system evaluates which requirements remain satisfied within the user-defined tolerance. This produces a coverage relationship between conditions and requirements that defines the search space for the reduction process.

5.11. REQUIREMENT SATISFACTION

Two types of requirements exist in this framework: One-dimensional envelope requirements verify that the load value produced by a condition remains within the tolerance threshold of the baseline envelope extreme. The second is Two-dimensional requirements derived from potato plot analysis, which verify that the load pair produced by a condition lies within the normalized Euclidean tolerance region surrounding a baseline hull vertex. Because the tolerance region is circular, the position of the vertex on the envelope does not bias the requirement in any particular direction.

Because of the large number of flight conditions present in the dataset, a single condition may satisfy multiple envelope requirements simultaneously. In some cases, a condition may represent both a maximum and minimum envelope location across different components or degrees of freedom. In other situations, a condition may satisfy only one requirement, while another condition may satisfy that same requirement along with several others. This overlap between conditions and requirements is what allows the reduction process to remove redundant cases while still preserving the envelope behavior of the full dataset.

5.12. COVERAGE MAP CONSTRUCTION

To represent these relationships, the system constructs a coverage map that records which conditions satisfy which requirements. Conceptually, this structure forms a bipartite relationship between two sets: the set of candidate conditions and the set of envelope requirements. An edge exists between a condition and a requirement when that condition satisfies the requirement within the defined tolerance.

The coverage map is constructed before the reduction algorithm begins. Performing this step up front allows SQL to perform the heavy work of identifying the relevant data after the dataset has been filtered by aircraft, loop, analysis type, and user-specified constraints. Once the coverage relationships are determined, they remain fixed, eliminating the need for the reduction algorithm to dynamically recompute requirement satisfaction during execution.

5.13. SPARSE COVERAGE REPRESENTATION

Although the coverage relationship can be represented conceptually as a matrix of conditions versus requirements, most condition–requirement pairs do not represent valid coverage relationships. Storing the entire matrix would require allocating space for a large number of irrelevant zero entries. Instead, the implementation stores only the pairs where coverage exists in the form

$$(\text{condition\_id}, \text{requirement\_id})$$

This sparse representation avoids storing unnecessary data and reduces the amount of information that must be processed during reduction. The coverage map is then converted into a bitmask representation so that requirement coverage can be evaluated using bitwise operations. Bitmask operations are both memory efficient and computationally fast, allowing the algorithm to determine how many currently uncovered requirements a condition satisfies using simple CPU instructions.

5.14. BASELINE CONDITION PRESERVATION

In addition to the coverage relationships, the baseline conditions responsible for producing the original envelope extrema are preserved as part of the candidate condition set. These baseline cases represent the physically identified worst-case results from the full dataset. Although other conditions may reproduce the envelope behavior within tolerance, preserving the baseline cases ensures traceability between the reduced dataset and the original envelope results and allows the reduction process to favor those conditions during tie-breaking when equivalent coverage solutions exist.

5.15. COVERAGE GRAPH INTERPRETATION

After user-defined filtering, a typical dataset contains several thousand candidate conditions and several hundred envelope requirements. While this size does not make the problem computationally intractable, it is large enough that naive search approaches become inefficient. Using established approximation strategies allows the system to efficiently identify a small subset of representative conditions without wasting computational effort.

Once the coverage map is constructed, the reduction problem becomes selecting conditions in an order that maximizes requirement coverage while minimizing the number of conditions retained. The following section will describe how this selection process is performed using a greedy set-cover reduction algorithm.

5.16. GREEDY REDUCTION ALGORITHM

Once the requirement set and coverage relationships have been established, the reduction problem becomes selecting a subset of flight conditions that collectively satisfy all envelope requirements. This problem can be formulated as a set-cover problem, where the objective is to identify the smallest collection of elements whose combined coverage satisfies a predefined set of constraints. Greedy approximations to the set-cover problem are widely used because they provide efficient solutions for large datasets while maintaining strong approximation guarantees (Chvátal, 1979; Feige, 1998). [6, 7] For the scale of external loads datasets, where thousands of conditions may exist but only a small subset ultimately defines the envelope behavior, a greedy approach provides an effective and computationally practical solution.

5.17. ALGORITHM INITIALIZATION

The reduction algorithm operates on the requirement set and the precomputed coverage map generated in the previous sections. Each requirement is assigned a unique bit index, and each condition is represented using a bitmask indicating which requirements it satisfies. This representation allows requirement coverage to be evaluated using fast bitwise operations.

The algorithm begins with the baseline envelope conditions identified during requirement generation. These baseline conditions correspond to the flight cases that originally produced the envelope extrema in the full dataset and remain part of the candidate pool to preserve traceability to the original analysis results. An uncovered requirement mask is initialized to represent the full set of requirements that must be satisfied. As the reduction progresses, this mask is updated to track which requirements remain unresolved.

5.18. GREEDY CONDITION SELECTION

At each iteration, the algorithm evaluates every candidate condition to determine how many currently uncovered requirements it can satisfy. For a given condition, the set of newly satisfied requirements is computed using a bitwise intersection between the condition coverage mask and the mask representing the remaining uncovered requirements:

$$new\_bits = mask \,\&\, uncovered$$

If this intersection is empty, the condition does not contribute additional coverage and is ignored during that iteration. Candidate conditions are then scored according to the requirements they now satisfy. In the simplest configuration, the score is the number of newly covered requirements. The default implementation applies a rarity-weighted scoring scheme in which each requirement is assigned a weight based on how many conditions satisfy it within the dataset. Requirements satisfied by fewer conditions are considered rarer and therefore receive higher weights. The score of a condition is computed as the sum of the weights associated with the requirements it newly satisfies.

This weighting encourages the algorithm to satisfy rare requirements earlier in the reduction process, reducing the likelihood that later iterations become constrained by difficult-to-satisfy requirements. During each iteration, all candidate conditions are evaluated and assigned a score. The condition with the highest score is selected and added to the reduced condition set. If multiple conditions produce identical scores, the first condition encountered is selected, ensuring deterministic execution.

5.19. ITERATIVE REDUCTION PROCESS

Once a condition has been selected, the uncovered requirement mask is updated by clearing the bits corresponding to the requirements satisfied by that condition. The algorithm then proceeds to the next iteration and recomputes condition scores based on the updated uncovered set.

The candidate condition list itself is not modified during execution. Conditions that no longer provide new coverage are naturally filtered out because their intersection with the uncovered requirement mask produces no new bits. The greedy loop continues until all requirements have been satisfied. An additional verification check ensures that every requirement is covered by at least one selected condition. If the algorithm encounters a requirement that cannot be satisfied by any remaining condition, an error is raised, indicating an inconsistency in the requirement generation or coverage mapping stages.

Although the underlying database may contain millions of individual load records, the reduction algorithm operates only on the requirement set and the associated coverage map. Typical reductions involve several thousand candidate conditions and several hundred to a few thousand envelope requirements depending on the number of components, spatial points, and potato plots requested by the user. The use of bitmask representations allows coverage to be evaluated extremely efficiently even at this scale.

The greedy reduction process produces the final reduced set of flight conditions that collectively satisfies all envelope requirements within the specified tolerance. This reduced condition set forms the primary output of the algorithm and represents the subset of cases that must be retained to preserve the envelope behavior of the original dataset. Additional outputs and verification checks used to evaluate the quality of the reduction are described in the next section.

5.20. ENVELOPE VERIFICATION

After the greedy reduction process is completed, the reduced condition set is automatically verified to ensure that the envelope behavior of the original dataset has been preserved within the user-specified tolerance. Verification is performed within the reducer workflow and is executed every time the reduction algorithm runs.

The verification process compares the envelopes produced by the reduced condition set against the baseline envelope. This comparison is performed independently for one-dimensional envelope extrema and for two-dimensional potato plot envelopes. The verification step does not re-evaluate the requirement coverage directly; instead, it recomputes the envelope values using the reduced condition set and checks that they remain within the specified tolerance relative to the baseline envelope.

5.21. VERIFICATION OUTPUTS

The verification process produces a pass or fail result for the overall reduction. If any envelope location fails the tolerance check, the reducer generates output identifying the component, point, degree of freedom, or potato plot vertex responsible for the failure.

In addition to the automated verification check, the reducer generates several output files that allow engineers to manually review the results of the reduction process. These include baseline and reduced envelopes, envelope comparison files, and potato plot visualizations for both the baseline and reduced condition sets.

Because potato plot evaluation has traditionally involved an engineer-in-the-loop workflow, these outputs provide transparency and allow engineers to confirm that the simplification and reduction processes have not removed cases that may still be important for engineering judgment or certification considerations.

6. RESULTS AND EVALUATION

6.1. DATASET DESCRIPTION

The database used in this study was constructed using external loads data generated from an aircraft loads analysis workflow. To avoid disclosure of proprietary information, aircraft identifiers and certain metadata fields have been generalized. However, the size and structure of the dataset reflect a real engineering loads environment, including the number of flight conditions, components, monitor points, and load values used in a current aircraft development program.

TABLE provides an overview of the size of the database and requirements for the reduction algorithm.

TABLE 1 Dataset Size and Requirement Counts used in Reduction Experiments

Metric	Value
Total flight conditions	4914
Total components	30
Total stress points	77
Total static flight fact table rows	2,270,268
Envelope requirements (1D only)	696
Potato plot configurations evaluated	0-15

Each condition represents a simulated flight case containing structural loads across all aircraft components and stress points. These loads were stored in the relational database described previously in Database Architecture and Data Framework and queried during the reduction process.

6.2. BASELINE ENVELOPE EXTRACTION

The first step in the reduction workflow is extraction of the baseline envelope conditions. These conditions represent the maximum and minimum loads observed across all components, stress points, and degrees of freedom. For the dataset used in this study, the baseline envelope extraction produced results shown in TABLE.

TABLE 2 Envelope Extraction Results

Metric	Value
Full dataset conditions	4914
Baseline envelope witness conditions	196

These witness conditions represent the baseline set of flight cases that produce the extreme loads across the aircraft structure. While the baseline envelope already reduces the total dataset significantly, many of these conditions only represent the extreme load at a single location and may not be required to represent the envelope within an acceptable tolerance. The greedy reduction algorithm described previously in Reduction Methodology and Algorithm Implementation is applied to further reduce this set while preserving envelope behavior.

6.3. REDUCTION PERFORMANCE UNDER INCREASING CONSTRAINT COMPLEXITY

The reduction algorithm was evaluated using several configurations with increasing envelope constraint complexity. The reducer operates on a set of requirements derived from envelope extrema and optional potato plot convex hull vertices. These requirements define the load conditions that must be preserved by the reduced condition set within a specified tolerance. The reduction parameters used in all experiments are shown in Table 3.

TABLE 3 Reduction Parameters Used For All Configurations

Parameter	Value
Envelope tolerance	3%
Potato tolerance	3%
Potato angular deviation	3%

To evaluate how the algorithm scales with increasing envelope complexity, four configurations were tested:

- Envelope constraints only (1D extrema)
- Envelope constraints with 1 potato plot
- Envelope constraints with 2 potato plots
- Envelope constraints with 15 potato plots

Potato plots introduce multidimensional constraints that represent combined loading behavior between two degrees of freedom. These convex hull vertices add additional envelope requirements beyond the one-dimensional extrema. The potato plot configurations were cumulative. Potato plots introduced in earlier configurations remained active in subsequent configurations. For example, the potato plot used in the single-potato configuration was also included in the two-potato and fifteen-potato configurations. TABLE shows the envelope-only configuration reduced the baseline witness set from 196 conditions to 82 cases while preserving all envelope constraints within the specified tolerance. This represents a reduction of approximately 58.2% of the baseline envelope cases.

The baseline envelope extraction itself reduced the full dataset of 4914 flight conditions to 196 witness cases. The reducer algorithm then further removed redundant cases while maintaining envelope fidelity, producing a final reduced condition set suitable for downstream structural analysis. As additional potato plot constraints were introduced, the number of envelope requirements increased from 696 to 817. Despite this increase in constraint complexity, the number of retained conditions increased only modestly, from 82 to 94 cases. For the most complex configuration evaluated, 817 envelope requirements were satisfied using 94 retained conditions, corresponding to an average of approximately 8.7 requirements per retained case. This indicates that many flight conditions simultaneously satisfy multiple envelope and multidimensional load constraints. Because structural loads are correlated across the aircraft, a single flight condition often produces significant loads at multiple locations and degrees of freedom, allowing the greedy algorithm to efficiently select a small subset of representative cases.

Runtime increased as additional constraints were added, ranging from approximately 23 seconds for envelope-only reduction to 72 seconds for the configuration containing fifteen potato plots. Even for the most complex configuration evaluated, the reduction process completed in under two minutes.

TABLE 4 Reduction Results For Increasing Potato Plot Constraints

Configuration	Requirements	Baseline Witness Cases	Reduced Cases	Reduction vs Baseline	Runtime
Envelope only	696	196	82	58.2%	23.13 s
Envelope + 1 potato	705	207	82	60.4%	26.53 s
Envelope + 2 potato	715	208	83	60.1%	29.06 s
Envelope + 15 potato	817	285	94	67.0%	71.91 s

FIGURE compares the baseline and reduced envelopes for the right-wing maximum shear load distribution. The two curves are nearly indistinguishable, indicating that the reduced condition set accurately reproduces the baseline envelope shape. The

percent difference between the baseline and reduced envelopes is also shown on the secondary axis. All differences remain within the specified 3% tolerance, confirming that the reduction process preserves envelope fidelity across the span.

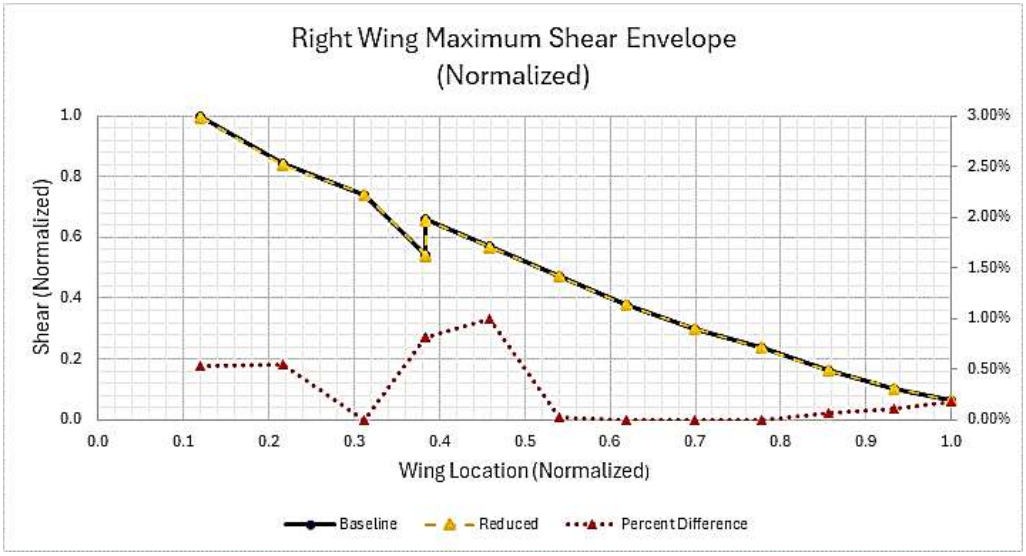

FIGURE 2 Baseline Envelope Vs Reduced Envelope Comparison

Figure 3 compares the baseline potato plot for wing bending and wing torsion at station 70% span on the wing. This data shows how the data can cluster in certain corners of the graph, and the simplification plus reducer algorithm helps eliminate redundant cases while still representing the potato plot envelope as a whole.

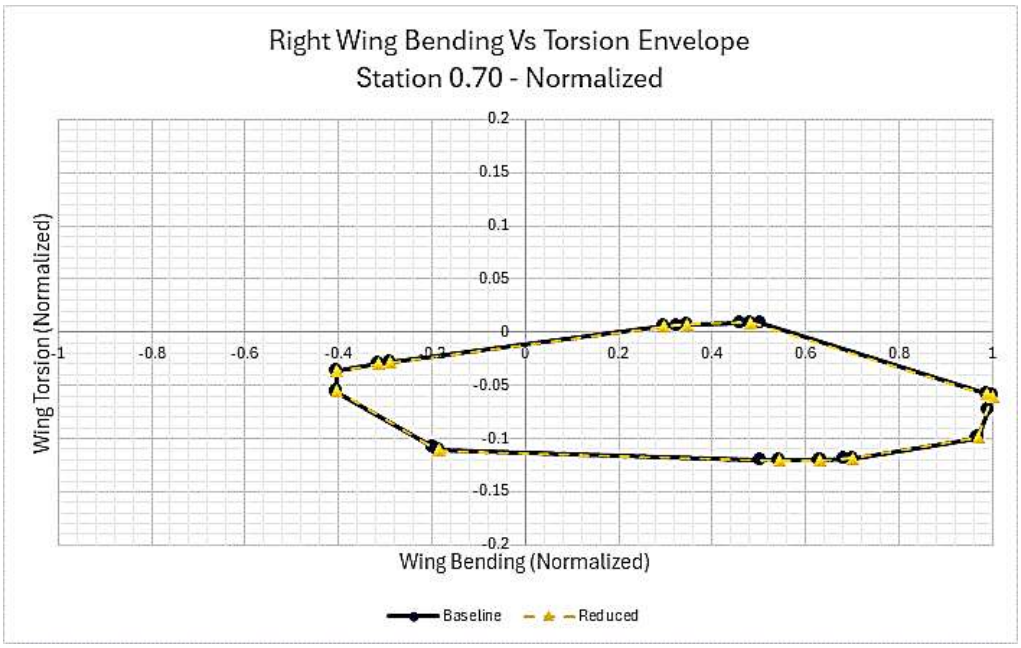

FIGURE 3 Example Potato Plot Comparison (Baseline Vs Reduced)

6.4. EFFECT OF POTATO PLOT SIMPLIFICATION

The convex hull simplification method described previously in Requirement Generation was evaluated to determine its impact on reduction efficiency. When convex hull simplification was enabled, several potato plots experienced reductions in the number of hull vertices. For example, reducing the number of convex hull vertices decreases the number of envelope requirements that must be satisfied by the greedy algorithm, which improves computational efficiency as shown in TABLE .

TABLE 5 Example Convex Hull Simplification Results

Potato Plot	Points	Original Hull Vertices	Simplified Hull Vertices
Left wing station 0.7 (Mx,My)	3234	23	9
Left wing station 0.25 (Mx,My)	3234	21	10
Right wing station 0.25 (Mx,My)	3234	16	9

6.5. REDUCTION EFFICIENCY

The reduction efficiency can be evaluated by comparing the number of retained conditions with the number of envelope requirements. The largest requirement set and results are in TABLE . TABLE shows 94 cases can solve the 987 requirements in this sample, and therefore each case solved ~10.5 requirements each. This indicates that each retained condition satisfies multiple envelope requirements simultaneously.

TABLE 6 Sample Reduction Results

Metric	Value
Envelope requirements	987
Reduced cases	94

The ability for individual conditions to satisfy many envelope constraints reflects the structural correlation of loads across the aircraft and allows the greedy algorithm to achieve significant compression of the dataset.

7. DISCUSSION

The reduction results show that increasing the number of envelope and potato plot requirements leads to only a modest increase in the number of retained conditions. This indicates that many flight conditions simultaneously satisfy multiple load constraints. Because structural loads are correlated across the aircraft, a single condition often produces significant responses at multiple locations and degrees of freedom. As a result, the greedy algorithm can efficiently select a small subset of conditions that collectively satisfy many requirements.

What proved most challenging in the reduction process was the incorporation of potato plot constraints. While envelope-based reduction follows a relatively straightforward formulation using established methods for identifying extrema, the definition of requirements for potato plots is less direct. Depending on the selected degrees of freedom and resolution of the convex hull, potato plots can generate many candidate requirements, many of which would typically be filtered through engineering judgment.

Replicating this judgment using a purely mathematical approach introduced additional complexity. A simplification process was developed to reduce redundant requirements; however, iterative testing showed that it was difficult to reproduce the selectivity of manual engineering review fully. Although the algorithm effectively reduced the overall condition set, the resulting potato plot requirements remained comparatively dense.

Given this behavior, a conservative approach was adopted. Rather than aggressively pruning requirements, the method retains a broader set of constraints and provides additional output to support traceability. This allows the engineer to review and validate the selected conditions, ensuring confidence in the reduced dataset while maintaining alignment with established engineering practices.

The database-driven architecture also plays a critical role in enabling this reduction process. By structuring external loads data into a relational format, envelope extraction, requirement generation, and traceability are performed through consistent and repeatable queries. This allows the reduction algorithm to operate on large datasets efficiently while maintaining clear linkage between reduced conditions and their originating flight cases. As a result, the system not only reduces the number of conditions required for analysis but also improves data organization, reproducibility, and traceability compared to traditional workflows.

8. LIMITATIONS

The reduction algorithm employs a greedy approach to maintain computational efficiency as the number of requirements increases. While simpler envelope-only reductions could be solved using optimal methods without significant runtime impact, the inclusion of multidimensional potato plot constraints can substantially increase the number of requirements. In these cases, the greedy algorithm provides a practical tradeoff, enabling efficient execution at the expense of potential non-optimality. This may result in a slightly larger retained condition set compared to an optimal solution, but the reduction remains sufficiently effective for engineering applications.

The performance of the algorithm is dependent on the selection of tolerance values. The method is designed to operate within a typical tolerance range of approximately 3–5%, which provides sufficient flexibility for reduction while maintaining envelope fidelity. If the tolerance is set too high, the algorithm may select conditions that consistently lie near the allowable bounds, potentially reducing confidence in the resulting design loads. Conversely, if the tolerance is too restrictive, the reduction process may yield little to no improvement over the baseline envelope, as alternative conditions no longer satisfy the requirements.

The accuracy of the reduction is also dependent on the quality and completeness of the input data. The algorithm assumes that the database accurately represents the full set of relevant flight conditions and load responses. If the underlying data is

incomplete or contains errors, the reduction process will preserve those inaccuracies. As a result, validation of the input dataset remains the responsibility of the engineer prior to applying the reduction process.

The incorporation of potato plot constraints introduces additional limitations because it is difficult to replicate engineering judgment through purely mathematical methods. A simplification process was developed to reduce redundant convex hull vertices using angular deviation and directional projection checks; however, this approach cannot fully reproduce the selectivity of manual review. To address this, a conservative strategy was adopted in which requirements are retained rather than aggressively pruned, and additional outputs are generated to support engineering validation. This ensures that potentially critical load combinations are not inadvertently removed.

Although the reduction process is designed to scale with increasing dataset size, additional considerations are required when applying the method to dynamic loading conditions. Potato plot validation assumes that the evaluated load cases are correlated representations of realistic loading scenarios. If dynamic conditions are not properly correlated, the resulting convex hulls may not accurately represent valid load combinations, limiting the applicability of the method in such cases.

Finally, the current implementation is limited to reductions performed on a single aircraft and analysis loop at a time. Differences in model definition, stress point locations, or coordinate systems between datasets prevent direct comparison and reduction across multiple configurations. Additionally, while the reduction process significantly decreases the number of conditions requiring evaluation, it does not eliminate the need for engineering oversight. Final review by an engineer remains necessary to confirm that critical conditions have been retained and that the reduced dataset is appropriate for downstream structural analysis.

9. FUTURE WORK

Future work for this system is focused on improving scalability and refining aspects of the reduction methodology. Transitioning the database from a local implementation to a centralized SQL server would enable multi-user access, improved data management, and the ability to incorporate new datasets without overwriting existing results. This would support long-term storage of historical data and allow the reduction process to be applied consistently across multiple analyses.

Additional improvements can be made to the potato plot simplification approach to better align with engineering judgment. While the current implementation provides a conservative and traceable method for handling multidimensional constraints, further refinement could improve selectivity and reduce the number of retained requirements without compromising envelope fidelity.

These enhancements build upon the current framework, which successfully achieves the intended objectives of automated reduction, traceability, and efficiency, while providing a path toward broader application and continued development.

10. CONCLUSION

This work addressed the challenge of reducing large external loads datasets into a manageable set of representative conditions for structural analysis. Traditional workflows rely on flat text-based data and manual iteration, making the reduction process time-consuming, difficult to reproduce, and prone to traceability issues.

To address these limitations, a database-driven reduction framework was developed that organizes external loads data within a relational SQL structure and enables consistent querying of envelope extrema and multidimensional constraints. This data architecture supports a reduction methodology that combines envelope extraction, potato plot convex hull requirements, and a greedy selection algorithm to identify a reduced set of critical conditions. A tolerance-based verification process ensures that the reduced dataset preserves the load envelope behavior of the original dataset.

Application of the method to a representative aircraft dataset demonstrated that the reduction process can significantly decrease the number of required conditions while maintaining envelope fidelity within a specified tolerance. The results show that a relatively small subset of conditions can represent the overall loading behavior, due to the structural correlation of loads across the aircraft.

In addition to reducing the number of conditions, the database-driven approach improves traceability, consistency, and scalability compared to traditional text-based workflows. By linking reduced conditions directly back to their originating data and enabling repeatable query-based processing, the framework provides a robust and reproducible solution that supports downstream structural analysis while maintaining alignment with engineering practices.

ACKNOWLEDGMENTS

The author would like to thank Camille Fioranelli for her editorial support, including improvements to clarity, wording, and overall readability. Her contributions strengthened the communication and presentation of this work. Moreover, this work was partially supported by the Askew Institute of the University of West Florida.

Appendix A

Reducer Input Configuration File: Reducer_Input.txt

```
BASIC INPUT BLOCK
# Core configuration for reduction execution
DB_PATH:                ..\database.db
Aircraft:               Aircraft_1
Loop:                   1.0
Analysis:               Static_Flight_Load

# Reduction tolerances (%)
EnvelopeTolerancePercent: 3
PotatoTolerancePercent:   3

# Potato plot simplification parameter (degrees)
PotatoAngularDeviation:   3

# Data filtering options
IgnoreZeroLoads:        true
END BLOCK

CONDITION BLOCK
# Optional filtering of conditions
# If left blank, all conditions are included
# Provide partial or full string matches to include specific cases
END BLOCK

COMPONENTS BLOCK
# Define components and associated degrees of freedom (DOFs)
Left_wing:          Fz, Mx, My
Right_wing:         Fz, Mx, My
Forward_Fuselage:   Fy, Fz, Mx, My, Mz
Aft_Fuselage:       Fy, Fz, Mx, My, Mz
Left_H-Stabilizer:  Fz, Mx, My
Left_V-Stabilizer:  Fz, Mx, My
Right_H-Stabilizer: Fz, Mx, My
Right_V-Stabilizer: Fz, Mx, My
END BLOCK

POTATO PLOT BLOCK
# Two-dimensional envelope definitions (Component, Location, DOF pair)
# Configuration shown: Envelope + 15 potato plots

# Left Wing
Left_wing:  -32.054,  Mx, My
Left_wing: -121.840,  Mx, My
Left_wing: -185.250,  Mx, My
Left_wing:  -32.054,  Fz, Mx
Left_wing: -121.840,  Fz, Mx
Left_wing: -185.250,  Fz, Mx

# Right Wing
Right_wing:   32.054,  Mx, My
Right_wing:  121.842,  Mx, My
Right_wing:  185.253,  Mx, My
```

```
Right_wing:  32.054,  Fz, Mx
Right_wing:  121.842,  Fz, Mx
Right_wing:  185.253,  Fz, Mx

# Forward Fuselage
Forward_Fuselage: 207.500,  Fz, My
Forward_Fuselage: 207.500,  Fy, Mz
Forward_Fuselage: 207.500,  My, Mz
END BLOCK
```

REFERENCES

[1] Emre Ünay et al., "Tool Development for Aircraft Loads Post-Processing," *Proceedings of the 28th International Congress of the Aeronautical Sciences,* pp. 1-6, 2012.

[2] R. Nazzeri, M. Haupt, F. Lange, and C. Sebastien, "Selection of critical load cases using an artificial neural network approach for reserve factor estimation," *Proceedings of the Deutscher Luft- und Raumfahrtkongress* pp. 1-10, 2015.

[3] E. F. Codd, "A relational model of data for large shared data banks," Communications of the ACM, vol. 13, no. 6, pp. 377–387, June 1970, doi: 10.1145/362384.362685.

[4] J. M. Hellerstein, M. Stonebraker, and J. Hamilton, "Architecture of a Database System," Foundations and Trends® in Databases, vol. 1, no. 2, pp. 141–259, 2007, doi: 10.1561/1900000002.

[5] Y. Babuji et al., "Parsl," Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, June 2019, doi: 10.1145/3307681.3325400.

[6] V. Chvatal, "A Greedy Heuristic for the Set-Covering Problem," Mathematics of Operations Research, vol. 4, no. 3, pp. 233–235, Aug. 1979, doi: 10.1287/moor.4.3.233.

[7] U. Feige, "A threshold of ln n for approximating set cover," Journal of the ACM, vol. 45, no. 4, pp. 634–652, July 1998, doi: 10.1145/285055.285059.

[8] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, "Geometric approximation via coresets," J. E. Goodman & J. O'Rourke (Eds.), *Combinatorial and computational geometry*, Cambridge University Press, vol. 52, pp. 1-30, 2004.

[9] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," ACM Transactions on Mathematical Software, vol. 22, no. 4, pp. 469–483, Dec. 1996, doi: 10.1145/235815.235821.

[10] S. Saha Bagui and R. Walsh Earp, Database Design Using Entity-Relationship Diagrams. CRC Press, 2022.