Automating FAA Flight Digest Reports

A Capstone Report Presented for the Master of Data Science in Computational Linguistics at the University of British Columbia

In Collaboration with Boeing's Jeppesen Branch

Cole Piche
David Kang
Nicole Lopez
Zhengyi Shan

Under the Supervision of Prof. Scott Mackie, and with Boeing mentors Patrick Mackinney, Amy Lam, and Stuart Zong

June 25th, 2025

1. Executive Summary

This project addresses the significant challenge of automating the parsing of the National Flight Data Digest (NFDD). The NFDD is a critical dataset for aviation, containing regularly updated information essential for operational decision-making. Currently, parsing this data is a daily task and primarily manual, resulting in significant inefficiencies, human errors, and increased operational costs.

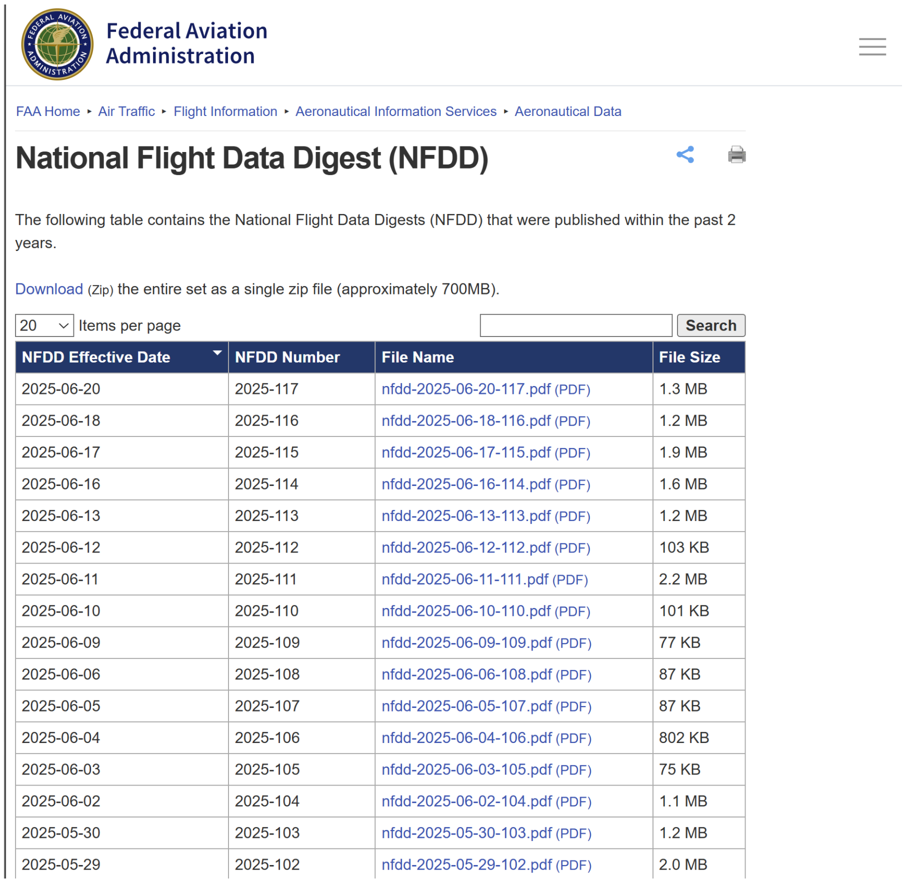
Our project aimed to streamline this process by developing an automated parsing solution using advanced Natural Language Processing (NLP) techniques. We approached this by initially identifying structured and semi-structured components within the NFDD data, applying NLP tools and computational methods to automate the extraction of essential information accurately and efficiently.

Through our efforts, we successfully demonstrated substantial improvements over manual parsing processes, achieving parsing accuracy over 80% for all sections, and over 90% for many, and significantly reducing the time required for data processing. Our automated solution provides a solid foundation for improving operational efficiency, reducing costs, and minimizing human errors. In subsequent sections, we detail our data handling methods, specific parsing approaches, and comprehensive evaluation metrics demonstrating our achievements. In subsequent sections, we will first discuss the data, including its source and domain, and preprocessing steps. Next we will discuss our methods, including how we filtered and parsed it. Then we will discuss how we evaluated our system and the results of this evaluation, and finally any future work that could be built on top of what we have done here.

2. Data

2.1 Source and Domain

The data for this project comes exclusively from the National Flight Data Digest (NFDD), provided by the Federal Aviation Administration (FAA). NFDD data is publicly accessible and can be obtained directly from the FAA's Aeronautical Information Services page: FAA NFDD [1], as shown in **Figure 1**. The domain of the data encompasses aviation-specific information necessary for the safe and efficient management of U.S. airspace. This includes, but is not limited to, navigational aids, airways, routes, airspace boundaries, airports, and instrument flight procedures.



Federal Aviation Administration

FAA Home ‣ Air Traffic ‣ Flight Information ‣ Aeronautical Information Services ‣ Aeronautical Data

National Flight Data Digest (NFDD)

The following table contains the National Flight Data Digests (NFDD) that were published within the past 2 years.

Download (Zip) the entire set as a single zip file (approximately 700MB).

20 Items per page Search

NFDD Effective Date	NFDD Number	File Name	File Size
2025-06-20	2025-117	nfdd-2025-06-20-117.pdf (PDF)	1.3 MB
2025-06-18	2025-116	nfdd-2025-06-18-116.pdf (PDF)	1.2 MB
2025-06-17	2025-115	nfdd-2025-06-17-115.pdf (PDF)	1.9 MB
2025-06-16	2025-114	nfdd-2025-06-16-114.pdf (PDF)	1.6 MB
2025-06-13	2025-113	nfdd-2025-06-13-113.pdf (PDF)	1.2 MB
2025-06-12	2025-112	nfdd-2025-06-12-112.pdf (PDF)	103 KB
2025-06-11	2025-111	nfdd-2025-06-11-111.pdf (PDF)	2.2 MB
2025-06-10	2025-110	nfdd-2025-06-10-110.pdf (PDF)	101 KB
2025-06-09	2025-109	nfdd-2025-06-09-109.pdf (PDF)	77 KB
2025-06-06	2025-108	nfdd-2025-06-06-108.pdf (PDF)	87 KB
2025-06-05	2025-107	nfdd-2025-06-05-107.pdf (PDF)	87 KB
2025-06-04	2025-106	nfdd-2025-06-04-106.pdf (PDF)	802 KB
2025-06-03	2025-105	nfdd-2025-06-03-105.pdf (PDF)	75 KB
2025-06-02	2025-104	nfdd-2025-06-02-104.pdf (PDF)	1.1 MB
2025-05-30	2025-103	nfdd-2025-05-30-103.pdf (PDF)	1.2 MB
2025-05-29	2025-102	nfdd-2025-05-29-102.pdf (PDF)	2.0 MB

Figure 1. FAA's Aeronautical Information Services page, publicly available.

2.2 Data Format

The NFDD files are distributed as structured PDF documents, which detail aviation navigation data adhering to the Aeronautical Radio, Incorporated (ARINC) 424 specification, as explained in the ARINC 424-17 [2] document. Each NFDD PDF contains tabulated information on aviation-related items and structured entries defined by precise coding guidelines and formats. The ARINC 424-17 specification describes each field, its size, encoding format, and allowed values, providing critical guidance for correctly parsing the NFDD PDFs.

Key features include:

- Detailed field identifiers (Record Identifiers)
- Geographic coordinates (latitude/longitude)

- Navigational data, including frequencies, magnetic variation, and altitude information
- Procedural data for flight operations

2.3 License

The data provided by the FAA through the NFDD is considered public domain information and does not have licensing restrictions. It is freely accessible to the public and intended for use in improving airspace safety and efficiency.

2.4 Data Preprocessing

Given that the NFDD data is provided in PDF format, significant preprocessing was required to convert the data into a structured format suitable for computational tasks.

Preprocessing steps involved:

PDF Analysis and Extraction: Text was extracted from PDF documents to txt format based on how the data was presented in the original PDF document. Each block, separated by its header, was saved as an individual text file.

Structured Data Extraction: Regular expressions and custom scripts were employed to split each header out into its own file.

Cleaning and Validation: Extracted data underwent validation checks against expected formats and values outlined in ARINC 424-17, correcting anomalies such as misaligned fields, incorrect coordinate formatting, or missing entries.

2.5 Volume of Data

The dataset for this project consists of National Flight Data Digest (NFDD) documents published regularly by the Federal Aviation Administration (FAA). The NFDD files are updated daily, and at any given time, approximately 500 of the most recent PDFs are publicly available through the FAA's dissemination channels. For our analysis, we focused on the 494 NFDD PDFs available during the project period, specifically from May 2023 to May 2025.

Each NFDD document contains extensive updates and essential aviation-related information, structured under multiple distinct headers. To better understand the characteristics and scope of this data, we conducted exploratory data analysis (EDA), yielding several informative insights:

- **Average NFDD file size**: Approximately **1.34 MB** per document (standard deviation ±0.96 MB), indicating considerable variability in file sizes and the amount of data included.
- **Total number of processed section files**: **3,250**, indicating the volume of distinct data segments extracted across all headers from the 494 NFDD documents analyzed.
- **Average number of headers per PDF**: **6.58**. This figure highlights the typical structural complexity within each NFDD file, where multiple distinct categories of information are commonly present.

The NFDD data analyzed in this project includes the following specific header categories, along with their respective frequencies across the 494 documents:

Header Category	**Occurrence Count**
AIRPORT	493
AIRSPACE_FIXES	486
HOLDING_PATTERNS	446
NAVAIDS	354
AIR_TRAFFIC_CONTROL_TOWERS	305
NAVAIDCOM	304
INSTRUMENT_LANDING_SYSTEMS	285

PREFERRED_IFR_ROUTE	166
ARTCC_COMMUNICATIONS_FREQUENCIES	80
FLIGHT_SERVICE_STATIONS	80
MILITARY_TRAINING_ROUTE	77
PARACHUTE_JUMPING_AREAS	41
ATS_AIRWAYS	39
MISCELLANEOUS_ACTIVITY_AREAS	31
SPECIAL_ACTIVITY_AIRSPACE	31
VOR_RECEIVER_CHECK_POINTS	30
ARTCC_BOUNDARY_POINTS	2

Table 1. Specific header categories, along with their respective frequencies across the 494 documents analyzed from May 2023 to May 2025.

This detailed breakdown of the data provides clarity on the distribution and prominence of various types of aviation information present within the NFDD files. Although these counts specifically reflect a two-year window, they offer valuable insight into the typical structure and frequency of each information type, setting clear expectations for future processing tasks and analyses. These insights not only provide valuable context for stakeholders but also assist future researchers or developers in effectively managing expectations regarding the data's nature and complexity.

2.6 Annotations

The data does not come pre-annotated in a traditional machine-learning sense. However, the structured format defined by the ARINC 424 specification serves as a robust schema guiding the parsing and annotation process. Each field in the dataset can be viewed as an annotation, clearly identified and explained by the ARINC 424-17 documentation. However, currently there is no available "golden standard" JSON format to be used as a validation set, so we created approximately 5 to 10 examples per header by hand, following a consistent format to use as our validation set.

2.7 Annotation Schema

The annotation schema strictly adheres to the ARINC 424-17 specification, where each field in the dataset is described explicitly. Key annotations include:

Record Identifier: Unique codes identifying the type of record (e.g., Waypoint, Airport, NAVAID).
Geographical Coordinates: Latitude and longitude fields with precise formatting.
Operational Data: Specific fields like altitude limits, runway identifiers, and navigational aid frequencies.
Procedure Data: Instrument approach procedures, departure routes, and standard terminal arrivals clearly annotated with structured fields as per ARINC guidelines.

By using this well-documented annotation schema, we ensured the structured data was accurately parsed and ready for downstream applications in natural language processing and automated parsing systems.

3. Methods

Our primary goal was to automate the parsing and structuring of aviation-related data from the FAA's National Flight Data Digest (NFDD) PDF documents. Due to the complexity and length of these documents, we developed a systematic pipeline to ensure accuracy, scalability, and efficiency. The final method adopted was primarily based on an approach encapsulated in the script `process_unified_LLM.py`, available in the project's Github repository[1]. Below is a detailed breakdown of this

[1] https://github.com/tkang7/boeing

method, followed by an "Experiments" subsection describing alternative approaches and initial exploratory methods.

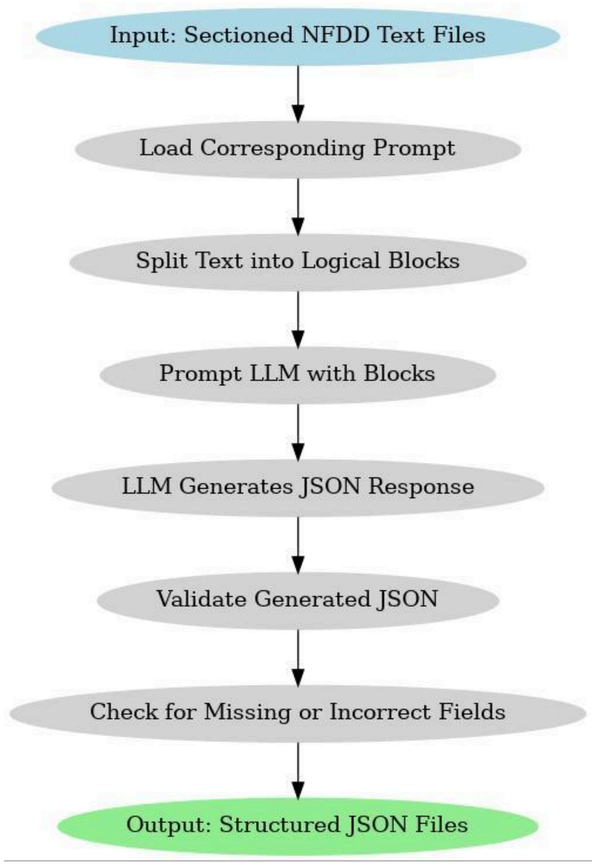

Input: Sectioned NFDD Text Files
Load Corresponding Prompt
Split Text into Logical Blocks
Prompt LLM with Blocks
LLM Generates JSON Response
Validate Generated JSON
Check for Missing or Incorrect Fields
Output: Structured JSON Files

Figure 2. An illustration of our LLM-assisted pipeline.

3.1 Data Acquisition and Pre-processing

The pipeline begins with downloading these lengthy NFDD documents provided as PDF files. These PDFs store text as coordinates rather than ordered characters, presenting an immediate preprocessing challenge. To handle this, we created a script that converts each PDF into a readable text file by sorting characters according to their positional coordinates. The sorted characters are then compiled into cohesive, ordered text files.

Once in text format, each NFDD document is segmented based on clearly defined headers. Specifically, we developed a rule-based script, section_splitter.py, to detect headers and isolate the associated content into distinct text files. Each generated file is named based on its originating document and header type, creating a structured and accessible dataset for subsequent parsing.

3.2 Parsing with Large Language Models (LLMs)

Following segmentation, we implemented an advanced parsing approach utilizing Large Language Models (LLMs), specifically OpenAI's GPT-4o mini. This was managed primarily through the script `process_unified_LLM.py`. Each header-specific text file, representing data blocks such as individual airports, air traffic control towers, and various communication frequencies, is fed individually into the LLM.

To guide the LLM effectively, each query incorporates carefully crafted prompts that demonstrate exactly how the unstructured NFDD text should be converted into structured JSON format. These prompts include explicit examples to guide the LLM's parsing behavior. We expand on our Prompt Engineering methods in section 4.3 below. An example of the prompts for the “ARTCC Communications Frequencies” section is also provided in the appendix (refer to Appendix A).

The prompt structure follows a rigorous standard, beginning with clear instructions that specify how to interpret and transform the raw text. It includes explicit guidance on handling edge cases, such as nested fields, repeated fields (e.g., "RMK" or "remarks"), and status indicators (e.g., "ADDED," "DELETED," "MODIFIED," and "UNCHANGED"). This approach ensures consistency and precision across diverse data formats.

3.3 Post-processing and Validation

After the initial JSON generation by the LLM, the pipeline involves critical post-processing steps. First, we flatten all single-item lists into their constituent items to simplify the JSON structure. Second, an essential validation step ensures data integrity: each generated JSON file is cross-referenced with its original text input to verify that no information has been inadvertently omitted. If discrepancies are found, the system automatically issues warnings, allowing for immediate attention and resolution.

Our final method is designed with scalability as a primary concern. The structured modular approach, separating data acquisition, header extraction, text segmentation, parsing, and validation into distinct scripts, facilitates easy maintenance, updates, and expansions to accommodate additional header types or data sources in the future.

4. Experiments

Throughout the project, we explored multiple methodologies to identify the optimal parsing strategy.

4.1 Initial Rule-based Exploration

Initially, we considered a purely rule-based parsing approach. This involved manually defined rules to detect patterns and segment text files. While straightforward and interpretable, this method quickly demonstrated significant limitations in scalability and adaptability. Specifically, the rigidity of manually defined rules made it difficult to handle the variability inherent in NFDD documents.

4.2 An Alternative LLM Method

In parallel to the main approach described above, our team developed a separate but related LLM pipeline for comparison and exploration purposes. This pipeline also began with converting PDFs into structured text files, followed by segmenting these texts into blocks based on identified headers. However, it diverged notably in how the text segments were fed into LLMs.

This alternative pipeline involved creating specific, highly detailed prompts customized for complex or ambiguous headers (e.g., differentiating between "NAVAIDS" and "NAVAIDS/COM"). A rule-based exclusion filter embedded in a function ensured accurate header matching by ignoring certain keywords during extraction. Additionally, this pipeline automatically validated LLM-generated outputs to confirm JSON validity. If an output failed JSON validation, it triggered an automatic warning and saved the raw output for manual review.

The primary advantage of this method was its highly targeted handling of complex headers. However, given the slight differences and overlaps with our primary approach, we ultimately integrated its strengths (such as detailed prompt engineering and robust JSON validation) into our final method.

4.3 Prompt Engineering and Design

The success and accuracy of our LLM-based parsing approach depend heavily on meticulously designed prompts. Prompts are carefully constructed text instructions fed to the LLM to clearly specify how the unstructured NFDD text should be converted into structured JSON. Each prompt explicitly guides the LLM's interpretation, ensuring consistent formatting, handling of edge cases, and correct labeling of status indicators ("ADDED," "DELETED," "MODIFIED," "UNCHANGED").

The process for developing these prompts was systematic. Initially, we reviewed multiple NFDD documents manually, analyzing the variety of content and structure present under each header type. We identified consistent patterns, irregularities, and potential edge cases such as nested fields, repeated fields (particularly "RMK" or "remarks"), and varying indentations. Based on this analysis, we created detailed prompt templates for each specific header type—such as "AIRPORT," "ATS AIRWAYS," and "ARTCC BOUNDARY POINTS"—each customized to handle the unique characteristics of its respective data section.

An example prompt structure includes the following components (a full example is provided in Appendix A):

- **Explicit Parsing Instructions**: Clear, step-by-step directions on how the text is structured and how it should be converted into JSON, emphasizing preserving original formatting (spaces, tabs, punctuation, capitalization).

- **Edge Case Handling**: Instructions on parsing nested or repeated fields and explicit rules on status tagging ("ADDED," "DELETED," etc.).

- **Example Input and Output**: Detailed examples demonstrating the exact expected transformation from raw NFDD text to structured JSON.

Each header-specific prompt was stored as a `.txt` file within the `prompts_concat` folder in the directory in our GitHub repository. The parsing script `process_unified_LLM.py` reads these prompts dynamically at runtime, concatenating them to form a complete and explicit set of instructions for each header type during the parsing operation. This approach ensures clarity, consistency, and ease of prompt updates or modifications, providing an adaptable framework for expanding the parsing pipeline to new headers or different data types in the future.

4.4 Test Cases Design and Validation

To systematically validate the parsing pipeline and ensure prompt accuracy, we developed detailed test cases, pairing specific raw input examples with their exact expected structured JSON outputs. These test cases serve a dual purpose: they validate the parsing logic and assess the robustness of the prompt instructions.

The test cases were methodically designed based on real examples from NFDD documents, aiming to capture a wide range of realistic scenarios. For each header type, we identified at least 10 distinct test cases to ensure comprehensive coverage of all expected situations, variations, and edge cases that might occur in the real data. The decision to include 10 test cases per header was guided by the need for thoroughness and the trade-off between ensuring coverage and managing complexity.

Each test case comprises two text files stored within the `test_cases` directory on our GitHub repository:

- **Test Input (`test_<header>_inputs.txt`)**: Raw NFDD text segments exactly as they appear in the source documents.

- **Expected Output (`test_<header>_outputs.txt`)**: The correct structured JSON format that the pipeline is expected to produce from the corresponding test input.

A typical test case pair example for the "ATS AIRWAYS" section is structured as follows (a full example is provided in Appendix B):

Input Example:

```
ATS AIRWAYS
V-123
    FROM         ABC VOR/DME                        TO DEF VORTAC
        MEA      3000                               MOCA      2500
    RMK      AIRWAY SEGMENT ESTABLISHED.                  ADDED
```

Figure 3. ATS Airways header example input.

Expected output example:

```
{
  "ATS AIRWAYS": {
    "airway": "V-123",
    "segments": [
      {
        "FROM": {
          "value": "ABC VOR/DME",
          "status": "UNCHANGED"
        },
        "TO": {
          "value": "DEF VORTAC",
          "status": "UNCHANGED"
        },
        "MEA": {
          "value": "3000",
          "status": "UNCHANGED"
        },
        "MOCA": {
          "value": "2500",
          "status": "UNCHANGED"
        },
        "RMK": {
          "value": "AIRWAY SEGMENT ESTABLISHED.",
          "status": "ADDED"
        }
      }
    ]
  }
}
```

Figure 4. ATS Airways example output.

During pipeline execution, the script `process_unified_LLM.py` automatically iterates through each pair of test inputs and expected outputs. It processes each test input through the LLM using the defined prompts, then systematically compares the LLM-generated JSON output against the expected JSON. The validation routine checks for JSON format accuracy, completeness, and ensures that all original text elements are present and correctly structured. Any discrepancies trigger detailed warnings, prompting immediate review and iteration of either the prompt design or the pipeline logic.

This structured testing methodology allows us to rigorously assess our pipeline's accuracy, quickly pinpoint and resolve errors, and continuously refine our parsing logic and prompt engineering. The detailed documentation of these tests in our repository ensures replicability and straightforward extension for future development efforts.

5. Evaluation

Given the nature of our task, which involved leveraging LLMs through few-shot prompting, our approach differed fundamentally from traditional supervised learning methods. Rather than training a model on a large annotated dataset, we utilized few-shot prompting, providing explicit, structured examples within prompts to guide the LLM in generating accurate JSON structures from unstructured NFDD data. Unlike supervised learning, which requires extensive labeled data for model training, few-shot prompting relies on the LLM's capability to generalize from a limited number of explicitly given examples. This choice was particularly suitable for our scenario since no extensive human-labeled training data was available, and manual annotation was infeasible at scale.

5.1 Evaluation Metric

The primary metric we utilized for evaluating the parsing accuracy was defined at the item level. We define an “item” as a distinct data entry under a single header (e.g., an individual airport or air traffic control tower). We considered an item correctly parsed if, and only if, every field within it—including field name, value, status (e.g., "ADDED," "DELETED," "MODIFIED," "UNCHANGED"), and formatting—was correct. Mathematically, we represented accuracy as follows:

$$Accuracy \ = \frac{Number\ of\ items\ correctly\ parsed}{Total\ number\ of\ items\ evaluated} \ \times\ 100\%$$

We deliberately chose this strict accuracy definition rather than a field-level evaluation to reflect the critical importance of accuracy for end-users. A field-level approach might misleadingly suggest high accuracy rates even if most items contain some errors. For example, even minor inaccuracies—such as dropping crucial communication frequencies—could have serious operational consequences. Thus, our item-level evaluation metric provides a more realistic measure of reliability and usability.

5.2 Reliability and Consistency of Results

During our evaluation, we observed that accuracy results varied approximately ±10% relative to our baseline accuracy measurement due to the inherent probabilistic nature of LLM outputs. This variability stems from the generative models' tendency to produce slightly different outputs across different inference runs, even with the same input prompts. To mitigate this variability, we set the model's temperature parameter explicitly to a low value (close to 0), which reduces randomness in the model's predictions and helps stabilize the outputs. Despite this measure, some variability persisted, highlighting the intrinsic non-deterministic behavior of these models.

To further ensure robustness and consistency in our accuracy metrics, each evaluation was repeated four times under identical conditions, and the final reported accuracy was calculated as the average across these runs. A sample set of results illustrating this variance is provided below:

Run	Accuracy (%)
1	89
2	92
3	91
4	90
Average	**90.5**

Table 2. Sample accuracy results for parsing one header (e.g. AIRWAY*S)*

This approach provided a balanced representation of the system's performance, ensuring that our reported accuracy metrics are both transparent and reliable for assessing the practical effectiveness of the parsing system.

5.3 Baseline and Expected Performance

Since no human-labeled benchmark data was provided for direct comparison, our evaluation relied solely on internally annotated development sets created by our team. Thus, we did not have external baselines or reference data against which to benchmark our system. However, internal discussions established clear accuracy thresholds for usability and acceptance:

- Prototype Acceptance Threshold: 75%
- Production Readiness Threshold: 90%

These thresholds guided our iterative refinement process, wherein prompts were continuously adjusted and improved based on evaluation outcomes on the development set.

5.4 Final Evaluation Results

Our evaluation results, conducted across each header section, are summarized in Table 3 below. For a detailed table with results across runs, refer to Appendix D.

Header Section	Average (%)
AIRPORT	92.46
AIR TRAFFIC CONTROL TOWERS	92.85
ATS AIRWAYS	95
ARTCC BOUNDARY POINTS	85
ARTCC COMMUNICATIONS FREQUENCIES	90
INSTRUMENT LANDING SYSTEMS	92.5

FLIGHT SERVICE STATIONS	90
AIRSPACE FIXES	0.9
PREFERRED IFR ROUTE	90
HOLDING PATTERNS	90
MILITARY TRAINING ROUTES	89
NAVAIDS	95
NAVAIDS/COM	90
VOR RECEIVER CHECK POINTS	100
PARACHUTE JUMPING AREAS	91
SPECIAL ACTIVITY AIRSPACE	88
MISCELLANEOUS ACTIVITY AREAS	88

Table 3. *Evaluation results, conducted across each header section.*

Overall, initial evaluations suggest that our few-shot prompting pipeline consistently achieves accuracy surpassing the production readiness threshold of 90%. Detailed results for each header will be presented in the appendix, including full tables and visualizations to provide a comprehensive understanding of the model's performance.

This rigorous evaluation process not only ensures transparency and reliability but also provides clear insights into areas needing further refinement, facilitating ongoing improvements and potential future enhancements of the parsing system.

6. Analysis

Throughout our project, we obtained results that met the partner's expectations in terms of accuracy, yet our analyses revealed several intriguing insights into the behavior of LLMs, specifically GPT-4o. Our detailed exploration yielded important observations, particularly about zero-shot prompting, persistent formatting behaviors, and model robustness. These insights are vital for future implementations or adaptations of this technology by Boeing or any stakeholders interested in applying similar methods.

6.1 Zero-Shot Prompting Capabilities

One notable observation was the LLM's capability to accurately format certain data structures without explicit examples, an approach known in the literature as zero-shot prompting. For example, during the parsing of sections containing frequency data, each frequency entry had attributes that were implicitly linked, such as frequency uses not explicitly defined as subfields. Remarkably, GPT-4o recognized these implicit relationships naturally, correctly associating each frequency use with its corresponding frequency number without explicit guidance. This ability is attributed to the extensive pre-training of GPT-4o on diverse datasets, enabling it to infer such relationships effectively based solely on contextual cues in the input text.

A practical example of zero-shot performance involved frequencies listed in the ARTCC_COMMUNICATIONS_FREQUENCIES sections, where the model consistently interpreted frequency uses as related to the preceding frequency number, without any provided examples (See Appendix B & C for detailed input-output examples).

This phenomenon is especially significant because it implies a strong baseline understanding from the model's pretrained knowledge. A valuable follow-up experiment would involve benchmarking multiple LLMs, such as Llama3 and DeepSeek, to determine if these models exhibit similar comprehension capabilities when provided with identical zero-shot prompts, thereby evaluating the consistency and reliability of different LLMs' pretrained knowledge.

6.2 Persistent Formatting Issues

Despite extensive prompting efforts, certain formatting behaviors proved persistently challenging. Notably, runway data presented a particular difficulty. The NFDD documents often listed runways with associated runway ends; our intended JSON structure required separating runway ends as independent objects. However, GPT-4o repeatedly formatted these runway ends as subfields within the runway objects, regardless of the specificity and detail provided in the prompts. This consistent behavior underscores limitations inherent to GPT-4o's pretrained formatting preferences, suggesting deeply embedded schema biases within its training data.

For example, consider the following input snippet from our prompt experiments:

```
RWY ID        10/28
  RWY END     10
    ELEVATION 6054.7
  RWY END     28
    ELEVATION 6033
```

We consistently prompted GPT-4o to separate "RWY END" entries as distinct objects parallel to "RWY ID," but it repeatedly nested them within "RWY ID." Due to this persistent challenge, we implemented an additional rule-based post-processing step within our pipeline (`process_unified_LLM.py`) to automatically correct the nested structure.

These insights are particularly valuable for Boeing as they highlight which data structures are naturally comprehensible to LLMs and which require additional interventions, thus guiding effective future model implementation and integration.

6.3 Robustness and Accuracy Considerations

Our comprehensive evaluations indicated a consistent accuracy slightly above the 90% threshold set by our partners. While this level of accuracy validates the efficacy of our LLM-based approach, it is imperative to recognize standard industry precautions. Given that critical safety information might be involved, we recommend a human-in-the-loop verification process to validate crucial outputs systematically. Although a 90% accuracy rate is robust for many use-cases, mission-critical scenarios inherently require rigorous verification protocols.

To facilitate this, we developed preliminary validation mechanisms within our pipeline. Specifically, our system includes a validation check where the model output is rejected if it omits any portion of the input text block. This ensures completeness and helps maintain data integrity. However, more sophisticated validation measures could be beneficial, such as automatic checks confirming the accuracy of each field’s status (e.g., "ADDED," "DELETED," "MODIFIED," "UNCHANGED"). These would further enhance reliability and are recommended for future development.

6.4 Comparative Insights from Methodological Experiments

Our experimental comparisons clearly illustrated the advantages of employing advanced LLM-based parsing methods over strictly rule-based systems. The flexibility and adaptability of GPT-4o significantly streamlined the parsing process, notably outperforming purely manual rule-driven methods.

However, our experiences also underscored the complementary value of rule-based approaches. While purely rule-based systems proved inadequate due to their lack of scalability and adaptability, integrating rule-based validation mechanisms into LLM workflows provided a balanced, robust solution. For instance, rule-based post-processing was critical in handling stubborn formatting issues, as previously discussed.

6.5 Summary and Recommendations

In summary, our analysis highlights crucial insights:

- Zero-shot prompting reveals GPT-4o’s impressive innate understanding derived from extensive pretraining but should be further validated across multiple models.
- Persistent formatting behaviors indicate deeply embedded schema preferences that must be addressed through complementary rule-based processing.
- Accuracy achieved (just above 90%) warrants caution; integrating systematic human verification and robust automated validation steps is critical for sensitive applications.

These findings provide a comprehensive perspective, informing Boeing and future stakeholders of key considerations necessary to leverage LLM capabilities effectively and safely.

6.6 Limitations & Future Work

One significant limitation of our approach, inherent to most machine learning-based methods, is that it achieves less than perfect accuracy. Although this limitation is common, it is particularly relevant in aviation contexts where even minor inaccuracies may carry implications for operational efficiency and safety. Therefore, we recommend a standard safety protocol wherein generated outputs undergo human review or additional automated verification to mitigate potential errors.

To address this concern, we began developing an automated validation system that rejects outputs if any original input text is missing from the final parsed JSON structure. While this initial version has been integrated into our pipeline, further refinement could enhance its capabilities. Specifically, more sophisticated validation could include format validation of key fields and cross-checks with known valid data structures. Implementing such validation measures could significantly improve the robustness and trustworthiness of our system. Future work should involve fully developing and integrating these additional automated checks.

Additionally, the current system relies heavily on general-purpose large language models (LLMs) like GPT-4o, which were trained on broad datasets not specific to the NFDD documentation style or format. While comprehensive pre-training of an entirely new transformer model for this specific task may not be feasible, considerable accuracy improvements might be achievable through domain-specific fine-tuning. One practical approach would involve training a specialized tokenizer and embedding model tailored explicitly to NFDD document characteristics, followed by fine-tuning an existing LLM on a carefully curated set of annotated NFDD documents. This method would allow the model to learn specific patterns, formatting nuances, and terminologies unique to aviation documentation, likely leading to increased parsing accuracy and consistency.

Another significant constraint encountered in this project was the absence of a substantial annotated dataset for training or detailed evaluation. The lack of existing labeled data necessitated extensive manual annotation efforts and limited our ability to thoroughly benchmark the system's performance. Future efforts should aim to compile a larger, comprehensive, annotated dataset—potentially through semi-automated annotation processes or by leveraging outputs from our developed pipeline for manual verification. This expanded dataset would not only facilitate more robust evaluation but also support future model fine-tuning or custom training, significantly enhancing accuracy and reliability.

In summary, future enhancements should focus on strengthening automated validation systems, performing targeted fine-tuning of existing language models using aviation-specific embedding models, and developing comprehensive annotated datasets. These improvements represent practical next steps towards maximizing accuracy and reliability, ensuring the system meets the demanding standards of aviation documentation processing.

7. Conclusion

In conclusion, we successfully developed an automated pipeline designed to parse FAA PDF documents into structured JSON format, significantly streamlining a previously manual and labor-intensive process. This automated system has the potential to eliminate approximately 12-18 worker-hours daily, representing substantial time and cost savings for the organization. The methodology employed leveraged advanced large language models, guided by carefully crafted few-shot and zero-shot prompts, achieving accuracy scores consistently above the 75% threshold set for a successful prototype, with all but one header categories achieving 90% accuracy, and many surpassing the 90% accuracy threshold necessary for deployment in production.

While our results demonstrate clear viability, some limitations were identified, primarily stemming from the inherent variability and complexity of the FAA documents. Future enhancements could include the development of advanced validation systems to further ensure output reliability, the fine-tuning of language models on task-specific data, and the creation of an extensive, annotated dataset to support ongoing improvements. Overall, our work establishes a robust foundation for continued optimization and significantly contributes to operational efficiency.

References

[1] **National Flight Data Digest (NFDD).** (2025, March 26). FAA. Retrieved June 25, 2025, from https://www.faa.gov/air_traffic/flight_info/aeronav/Aero_Data/NFDD/

[2] **Aeronautical Radio, Inc.** (2004, August 31). *ARINC Specification 424-17: Navigation System Data Base* (Prepared by the Airlines Electronic Engineering Committee) [PDF]. Annapolis, MD: Aeronautical Radio, Inc. Retrieved from https://aviation-ia.sae-itc.com/standards/arinc424-17-424-17-navigation-system-data-base

Appendix

A. Examples of prompts used in LLM-Assisted Pipeline[2]

```
Please follow the example that I am about to give you to parse the text I will give after the
example. Please do not output anything except for the json that I request, no comments, nothing,
as I am going to feed it directly into a python script.

Please make sure to not change field names or values that you find at all, meaning don't change
the number of spaces they contain, don't change the case, and don't add any underscores. Also be
careful of cases where there are many many spaces in a single value. You might have trouble
remembering to include the text that appears after the long string of spaces. Please do not
forget to include it. Also if you see two fields with the same name, do not change the name to be
plural, just leave the name as it is. Also you can tell where the breaks between key and value
are based on tabs. Tabs mean break, spaces are just part of the string. Also please make sure not
to forget any fields.

If you see any fields that are further indented than the ones above them, that means they are
subfields. Fields that are not indented are the top-level. Fields that are indented once are one
level down. Fields that are indented twice are nested under the previous field. Always preserve
this nesting in the output.

If a field appears multiple times (e.g. multiple RCAG sites), store them in an array. Each RCAG
entry may have nested LATITUDE and LONGITUDE fields, and may also contain multiple blocks of
{FREQUENCY, ALTITUDE, CHARTED} that should each be kept as a separate grouped object.

For example:

Input:
ARTCC  COMMUNICATIONS  FREQUENCIES
INDIANA
                                                            NFDD   120 - 1
06/23/2023
INDIANAPOLIS
       INDIANAPOLIS ARTCC
       IDENT

       ZID
       ICAO IDENT                                                          KZID
       RCAG
              SITE CATAWBA, OH
       LATITUDE -     39-58-18.8 N
LONGITUDE -    083-37-04.1 W
                     FREQUENCY                                                     128.775

                                                     ADDED
```

[2] Access https://github.com/tkang7/boeing/blob/main/prompts_concat/FREQUENCIES_prompt_concat.txt to view all prompts used.

```
                    ALTITUDE                                                          HIGH

                                                      ADDED
                    CHARTED                                                   NO

                                                      ADDED
                    FREQUENCY                                                 346.3

                                                      ADDED
                    ALTITUDE                                                          HIGH

                                                      ADDED
                    CHARTED                                                   NO

                                                      ADDED
```

```
Output:
{
  "state": {
    "value": "INDIANA",
    "status": "UNCHANGED"
  },
  "NFDD": {
    "value": "120 - 1",
    "status": "UNCHANGED"
  },
  "date": {
    "value": "06/23/2023",
    "status": "UNCHANGED"
  },
  "location": {
    "value": "INDIANAPOLIS",
    "status": "UNCHANGED"
  },
  "ARTCC name": {
    "value": "INDIANAPOLIS ARTCC",
    "status": "UNCHANGED"
  },
  "IDENT": {
    "value": "ZID",
    "status": "UNCHANGED"
  },
  "ICAO IDENT": {
    "value": "KZID",
    "status": "UNCHANGED"
  },
  "RCAG": [
    {
      "SITE": {
        "value": "CATAWBA, OH",
        "status": "UNCHANGED"
      },
      "LATITUDE": {
        "value": "39-58-18.8 N",
        "status": "UNCHANGED"
      },
      "LONGITUDE": {
        "value": "083-37-04.1 W",
        "status": "UNCHANGED"
      },
      "frequencies": [
        {
          "FREQUENCY": {
            "value": "128.775",
            "status": "ADDED"
          },
          "ALTITUDE": {
            "value": "HIGH",
            "status": "ADDED"
          },
          "CHARTED": {
            "value": "NO",
            "status": "ADDED"
          }
        },
        {
          "FREQUENCY": {
            "value": "346.3",
            "status": "ADDED"
          },
          "ALTITUDE": {
            "value": "HIGH",
            "status": "ADDED"
          },
          "CHARTED": {
            "value": "NO",
            "status": "ADDED"
          }
        }
      ]
    }
  ]
}
```

Also be aware of the case where the field `RMK` (Remark) appears in the block. Treat it like any other top-level field and preserve its spacing and indentation exactly. There may also be fields like `FREQUENCY`, `ALTITUDE`, and `CHARTED` that appear **outside** the RCAG block in rare cases; treat them as top-level unless they are nested.

Also handle empty values. For example:

```
       IDENT
                      ZID
```

means key is `IDENT`, value is `ZID`.

Please do not forget any fields.

B. Example of test case input txt file used in the LLM-Assisted Pipeline[3]

```
INDIANA
                                                            NFDD   120 - 1
06/23/2023
INDIANAPOLIS
       INDIANAPOLIS ARTCC
       IDENT

       ZID
       ICAO IDENT                                                          KZID
       RCAG
              SITE CATAWBA, OH
       LATITUDE -     39-58-18.8 N
LONGITUDE -    083-37-04.1 W
                      FREQUENCY                                                   128.775

                                                     ADDED
                      ALTITUDE                                                           HIGH

                                                            ADDED
                      CHARTED                                                     NO

                                                            ADDED
                      FREQUENCY                                                   346.3

                                                            ADDED
                      ALTITUDE                                                           HIGH

                                                            ADDED
                      CHARTED                                                     NO

                                                            ADDED
```

[3] Access https://github.com/tkang7/boeing/tree/main/test_cases to view all test cases used.

C. Example of test case output txt file used in the LLM-Assisted Pipeline

```
[
  {
    "state": {
      "value": "INDIANA",
      "status": "UNCHANGED"
    },
    "NFDD": {
      "value": "120 - 1",
      "status": "UNCHANGED"
    },
    "date": {
      "value": "06/23/2023",
      "status": "UNCHANGED"
    },
    "location": {
      "value": "INDIANAPOLIS",
      "status": "UNCHANGED"
    },
    "ARTCC name": {
      "value": "INDIANAPOLIS ARTCC",
      "status": "UNCHANGED"
    },
    "IDENT": {
      "value": "ZID",
      "status": "UNCHANGED"
    },
    "ICAO IDENT": {
      "value": "KZID",
      "status": "UNCHANGED"
    },
    "RCAG": {
      "SITE": {
        "value": "CATAWBA, OH",
        "status": "UNCHANGED"
      },
      "LATITUDE": {
        "value": "39-58-18.8 N",
        "status": "UNCHANGED"
      },
      "LONGITUDE": {
        "value": "083-37-04.1 W",
        "status": "UNCHANGED"
      },
      "frequencies": [
        {
          "FREQUENCY": {
            "value": "128.775",
            "status": "ADDED"
          },
          "ALTITUDE": {
            "value": "HIGH",
            "status": "ADDED"
          },
          "CHARTED": {
            "value": "NO",
            "status": "ADDED"
          }
        },
        {
          "FREQUENCY": {
            "value": "346.3",
            "status": "ADDED"
          },
          "ALTITUDE": {
            "value": "HIGH",
            "status": "ADDED"
          },
          "CHARTED": {
            "value": "NO",
            "status": "ADDED"
          }
        }
      ]
    }
  }
]
```

D. Evaluation Results across each header category (for each run)

Header Section	Run #1 (%)	Run #2 (%)	Run #3 (%)	Run #4 (%)	Average (%)
AIRPORT	92.307	**100**	84.6	92.307	92.46
AIR TRAFFIC CONTROL TOWERS	92.85	92.85	92.85	-	**92.85**
ATS AIRWAYS	90	100	90	**100**	95
ARTCC BOUNDARY POINTS	80	80	**100**	80	85

ARTCC COMMUNICATIONS FREQUENCIES	90	90	-	-	**90**
INSTRUMENT LANDING SYSTEMS	90	**100**	90	90	92.5
FLIGHT SERVICE STATIONS	90	90	-	-	**90**
AIRSPACE FIXES	90	90	-	-	0.9
PREFERRED IFR ROUTE	90	70	**100**	**100**	90
HOLDING PATTERNS	-	-	-	-	90
MILITARY TRAINING ROUTES	-	-	-	-	89
NAVAIDS	**100**	90	90	**100**	95
NAVAIDS/COM	-	-	-	-	90
VOR RECEIVER CHECK POINTS	100	100	100	99	**100**
PARACHUTE JUMPING AREAS	85	87	90	**100**	91

SPECIAL ACTIVITY AIRSPACE	80	**90**	90	90	88
MISCELLANEOUS ACTIVITY AREAS	-	-	-	-	**88**

Note: some runs were not recorded on this table. For these, the average accuracy is included.

E. Project Timeline

May 4–10:
Initial development of the PDF-to-text conversion system.

May 11–17:
Creation of the section splitter and early experimentation with a rule-based parsing approach.

May 18–24:
Rule-based parser development continued, but the approach was set aside due to complexity and scalability concerns. The team began working on automated parsing systems for their assigned sections.

May 25–31:
Basic versions of the automated parsing systems were operational, with initial test cases written for each section. Team members expanded their test sets and began iterating on prompt design to improve accuracy.

June 1–7:
The majority of test cases were completed, and prompt customization led to high accuracy in the targeted sections.

June 8–14:
Finalization of all test cases and prompts. All sections achieved target accuracy thresholds, and the pipeline was ready for evaluation and integration.

June 15-24:
Documentation and cleanup of github repo. Preparing the code for handoff, and all its accompanying documentation, reports, and presentation.

F. Contributions (for internal use only – i.e. capstone evaluator)

Zhengyi Shan

Throughout the project, I played a central role in designing, implementing, and refining the LLM-based pipeline for converting source data into structured JSON format. Early in the project, I collaborated with Cole to explore high-level technical strategies and engaged with the Boeing team to clarify project requirements and expectations.

One of my key contributions was developing a modular LLM pipeline capable of parsing complex data files by applying tailored prompts to different header types. I took the initiative to design and refine prompts for each section, ensuring that the LLM outputs were both accurate and consistent. Recognizing the limitations of available models, I experimented with multiple LLM platforms, including LLaMA-3.3-70B and LLaMA-Vision, and ultimately standardized the pipeline using GPT-4o-mini for its superior reliability and token capacity.

Additionally, I contributed to workload distribution by taking ownership of specific header processing tasks and assisted in transitioning our prompting approach from one-shot to few-shot learning based on evolving data challenges and feedback from Boeing. My efforts significantly enhanced the robustness and precision of the JSON conversion process, ensuring alignment with project standards.

David Kang

Throughout the project I focused on building the team's engineering backbone—first by setting up our workflows, and later by delivering critical parsing components that expanded the LLM pipeline's coverage.

First, I configured the development environment end-to-end, standing up our GitHub repo, Linear board, and CI settings so teammates could work in a consistent, reproducible space. I also reviewed and merged the initial pull requests, keeping coding standards uniform while we clarified Boeing's deliverable expectations. While the core framework was still in flight, I authored the project blog post that summarizes our approach for external stakeholders and future maintainers. This let the rest of the team concentrate on framework code while I ensured our methodology was clearly communicated.

I implemented the full PDF → TXT → JSON workflow for the MILITARY_TRAINING_ROUTE header. During this effort I created stringify_json_object, a utility that cleans model-generated JSON strings—removing escape sequences and newlines—so they can be ingested seamlessly by downstream modules. I also reorganized the repository structure so any pipeline step can be invoked from the project root without deep path traversal. Extending header coverage & automating conversion (early June). Building on the earlier work, I completed the same end-to-end decomposition for the NAVAIDCOM and MISCELLANEOUS headers. I then wired a new LLM sub-pipeline that automatically calls stringify_json_object when needed, giving the team a plug-and-play pattern for future headers.

Cole Piche

I handled the parsing for 5 of the headers, and also wrote most of the pre-processing code. The pdf to text step, the section splitter program, and the basic block parser were all me. I also double checked everyone's code at the end to ensure that there were no obvious bugs, I achieved an accuracy of at least 90% on every header that I was in charge of handling, and I wrote the basic pipeline that most of the other group members built their work on. I also wrote the system that adds the default dates into the json objects.

Nicole Lopez

Throughout the project, I primarily focused on operations, documentation, organizational effectiveness, and final report writing and presentation building. Initially, I established best practices for team communication, delegated tasks strategically, and set up clear guidelines for meeting deadlines. I ensured that our development workflows remained streamlined by creating comprehensive documentation, including detailed README files for each script and folder, and implementing best Git practices such as maintaining an organized repository, adding .gitignore files, and fostering good commit and pull request habits. Additionally, I was responsible for structuring the final report and presentation, assigning and coordinating team contributions weekly to avoid work bottlenecks. Technically, I developed exploratory data analysis (EDA) scripts that provided valuable insights for internal use, such as counting header occurrences, assessing the average number of headers per PDF, and filtering headers to facilitate efficient task management for team members. I also handled the parsing for five specific headers, thoroughly understanding their structures and creating robust test cases for pipeline evaluation, achieving over 90% accuracy in all but one section, which achieved ~82% accuracy on average. I also actively reviewed the overall pipeline implementation, providing critical feedback and suggesting targeted enhancements to increase accuracy and reliability.