HeronData: File Classifier

Shafir Rahman

Table of Contents

1 Introduction

1.1 Overview

In the financial services sector and beyond, efficient document processing is crucial. A fundamental component of the document processing and parsing pipeline is the classification of files. The existing repository from which this project was forked provides a foundational endpoint for this purpose. However, it employs a very basic classifier that categorizes files solely based on their filenames. This approach presents several challenges, including the inability to handle poorly named files, difficulties in scaling to accommodate different types of documents across new industries, and inefficiencies in processing large volumes of documents.

To address these limitations, this project implements and proposes several solutions. Extensive research and iterative development have led to the creation of a robust classification pipeline that utilizes two distinct classifiers. The first is an enhanced, custom fine-tuned filename classifier based on a large language model (BERT), which improves accuracy by implementing semantic similarity. The second is an advanced classifier based on the Swin Transformer architecture, also custom fine-tuned, which analyzes the actual content of the documents to achieve more precise classification.

Both classifiers are deployed via a Flask-based frontend application, facilitating ease of use and testing. Additionally, to ensure scalability and efficient production deployment, the project leverages AWS SageMaker. This deployment strategy allows the classifiers to handle increased workloads and integrate seamlessly with other AWS services. While the project also planned to incorporate a React Native application for broader accessibility, time constraints limited the completion of this component.

Overall, this project delivers a more sophisticated and scalable document classification system, addressing the key challenges of handling poorly named files, adapting to various industries, and processing large volumes of documents efficiently.

1.2 Goals

Enhance Classification Accuracy:

- **Handle Poorly Named Files:** Develop a classifier capable of accurately categorizing documents even when filenames are ambiguous or non-descriptive by leveraging semantic similarity and advanced natural language processing techniques.

Scalability and Adaptability:

- **Scale to New Industries:** Ensure the classifier can adapt to diverse industry-specific document types by training on a synthetically generated, industry-diverse dataset.
- **Process Larger Volumes:** Optimize the classifier's performance to handle increased document throughput without compromising speed or accuracy.

Advanced Model Implementation:

- **Dual-Classifier System:** Implement a two-tiered classification approach combining a simple semantic-based classifier with an advanced Swin Transformer model to enhance overall classification robustness and precision.

Deployment and Accessibility:

- **Production-Ready Deployment:** Deploy the advanced classifier on AWS SageMaker to provide a scalable and reliable service accessible to other applications and users.
- **Frontend Integration:** Develop a frontend application using Flask to facilitate easy interaction with the classifiers, enhancing user experience and accessibility.

Cost-Efficiency:

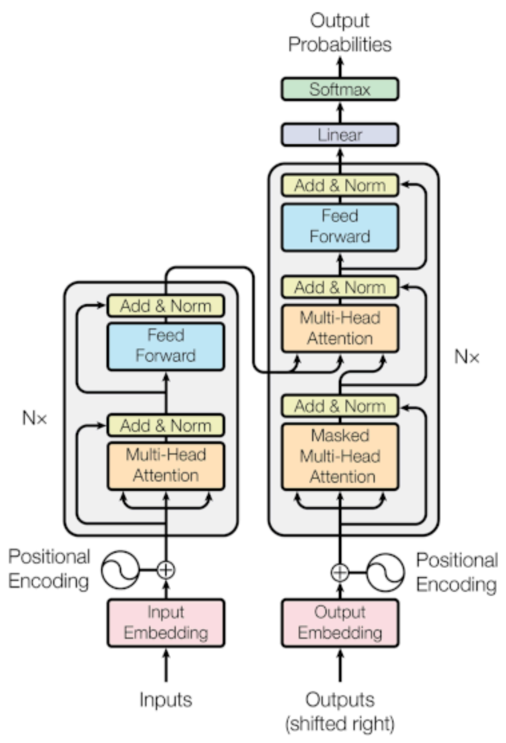
- **Minimize Expenditure:** Implement solutions that require little to no financial investment by utilizing free or low-cost tools and services wherever possible. Optimize the use of cloud resources to ensure that deployment and operational costs remain minimal without sacrificing performance or scalability.

Maintainability and Future Development:

- **Extendability:** Ensure the system is modular and maintainable, allowing for future enhancements such as integration with React Native for mobile accessibility and further scalability improvements.

2 Technical Background

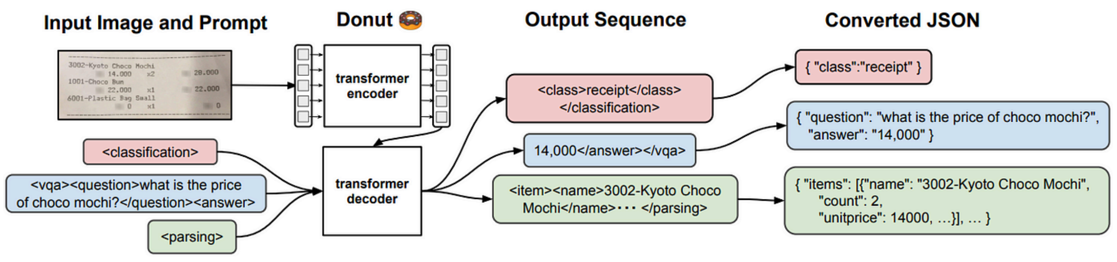
2.1BERT



BERT (Bidirectional Encoder Representations from Transformers) is a natural language processing model built by Google. It is build upon Transformer architecture. BERT utilizes bidirectional training to understand the context of a word based on its preceding and following text. This bidirectional approach allows BERT to capture intricate linguistic nuances and dependencies that unidirectional models may miss, enabling a deeper comprehension of language structure and meaning.

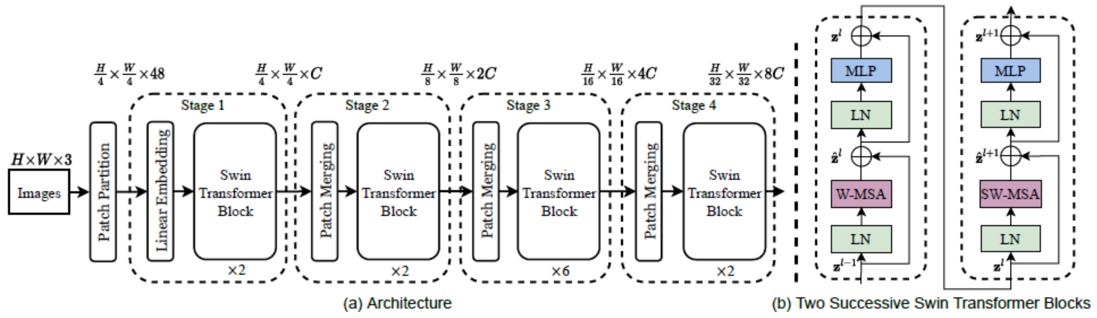
2.2 The Donut Model

The Donut Transformer is an advanced multimodal model designed for document understanding and classification. “Donut” stands for Document Understanding Transformer. It integrates both visual and textual information to achieve a comprehensive understanding of a document.


Input Image and Prompt
Donut
Output Sequence
Converted JSON
transformer encoder
transformer decoder
<classification>
<vqa><question>what is the price of choco mochi?</question><answer>
<parsing>
<class>receipt</class></classification>
14,000</answer></vqa>
<item><name>3002-Kyoto Choco Mochi</name>··· </parsing>
{ "class":"receipt" }
{ "question": "what is the price of choco mochi?", "answer": "14,000" }
{ "items": [{"name": "3002-Kyoto Choco Mochi", "count": 2, "unitprice": 14000, ...}], ... }

Donut Architecture

At the heart of the model lies the Swin Transformer, which enables efficient and scalable feature extraction from high-resolution document images. The Swin Transformer effectively identifies intricate structures and patterns within a document, making it a highly effective solution for document-related tasks, as these tasks typically involve documents with fairly universal structures. For example, most passports follow a standard format. The hierarchical feature extraction ability of the Swin Transformer allows it to process documents at multiple scales, from global layouts to fine-grained details.



Swin Transformer Architecture

3 Project Design and Implementation

3.1 Simple Classifier

The simple classifier serves as the foundational component for scalable and efficient file classification. This classifier enhances the original project's filename-based classification by incorporating semantic understanding through a fine-tuned BERT model. Below, we outline the design approach, synthetic filename generation, and model fine-tuning processes that contribute to the classifier's robustness and effectiveness.

3.1.1 Design Approach

A scalable and fast solution was essential for the simple classifier to handle large volumes of documents efficiently. The initial approach relied on simple string matching, which proved limited in handling variations and errors in filenames. To overcome these limitations, the classifier was enhanced using semantic similarity provided by a large language model (LLM) like BERT.

Key Enhancements:

- **Semantic Understanding:** Using BERT's ability to grasp language nuances and context improves the classifier's accuracy beyond mere string matching.

- **Robustness to Errors:** Fine-tuning with synthetic data equips the model to handle misspelled filenames and various formatting inconsistencies.
- **Contextual Classification:** The model can discern subtle differences and nuances in filenames, enabling more precise categorization.

3.1.2 Filename Generator

A critical component of the simple classifier is the synthetic filename generator, which creates a diverse and representative dataset for training the BERT model.

Generation Strategy:

- **Base Keywords Manipulation:** Starting with base keywords relevant to different categories, the generator simulates common mistakes by:
  - Inserting random characters.
  - Deleting random characters.
  - Replacing letters randomly.
  - Swapping letters.
- **Format Variations:** To mimic real-world scenarios, the generator introduces variations such as:
  - Adding dates to filenames
  - Applying camelCase and random capitalization
  - Incorporating common file formats and extensions.

This approach ensures the synthetic dataset captures a wide range of real-world filename discrepancies, enhancing the model's ability to generalize effectively. The generator has a lot of room for improvements to add more variations.

3.1.3 Model Fine-Tuning

Below are the steps taken to fine-tune the bert model.

Steps Involved:

1. **Data Preprocessing:**
   - **Cleaning and Normalization:** Filenames were cleaned to remove irrelevant characters and standardized to ensure consistency.
   - **Tokenization:** The cleaned filenames were tokenized into manageable chunks suitable for BERT processing.

2. **Fine-Tuning Process:**
    - **Pre-trained Model Selection:** A pre-trained BERT model was chosen for its proven capabilities in understanding and generating semantic embeddings.
    - **Training Configuration:**
        - **Objective:** The model was fine-tuned for classification, enabling it to map filename embeddings to predefined categories.
        - **Hyperparameters:** Optimized settings for learning rate, batch size, and epochs were selected to balance training efficiency and model performance.
        - **Validation:** A portion of the synthetic dataset was set aside for validation to monitor and adjust the model during training.
3. **Training Execution:**
    - The fine-tuning involved adjusting BERT's weights based on the synthetic dataset, allowing the model to better capture the specific nuances and variations in filenames relevant to the classification task.

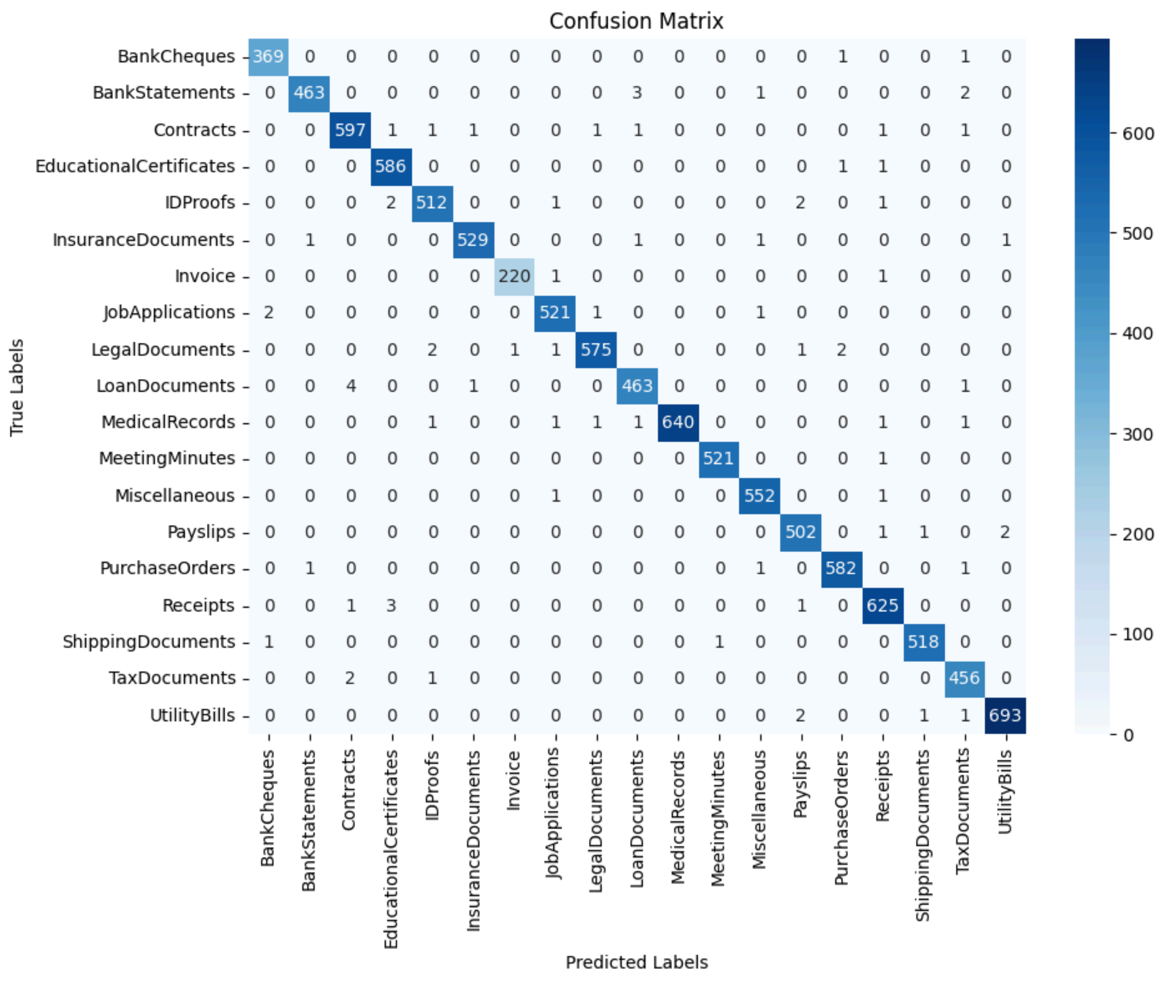
Outcomes:

- **Enhanced Accuracy:** Fine-tuning significantly improved the classifier's ability to correctly categorize filenames, even those with errors or unconventional formats.
- **Increased Robustness:** The model demonstrated resilience against common filename mistakes, maintaining high performance across diverse scenarios.
- **Contextual Insight:** BERT's semantic understanding enabled the classifier to discern and categorize nuanced differences in filenames effectively.

3.1.4 Performance Metrics

The simple classifier's performance was evaluated using a synthetic validation dataset, yielding promising results:

Metric	Value
Accuracy	99.26%
Precision	99.32%
Recall	99.29%
F1-Score	92.37%



True \ Predicted	BankCheques	BankStatements	Contracts	EducationalCertificates	IDProofs	InsuranceDocuments	Invoice	JobApplications	LegalDocuments	LoanDocuments	MedicalRecords	MeetingMinutes	Miscellaneous	Payslips	PurchaseOrders	Receipts	ShippingDocuments	TaxDocuments	UtilityBills
BankCheques	369	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
BankStatements	0	463	0	0	0	0	0	0	0	3	0	0	1	0	0	0	0	2	0
Contracts	0	0	597	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	0
EducationalCertificates	0	0	0	586	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
IDProofs	0	0	0	2	512	0	0	1	0	0	0	0	0	2	0	1	0	0	0
InsuranceDocuments	0	1	0	0	0	529	0	0	0	1	0	0	1	0	0	0	0	0	1
Invoice	0	0	0	0	0	0	220	1	0	0	0	0	0	0	0	1	0	0	0
JobApplications	2	0	0	0	0	0	0	521	1	0	0	0	1	0	0	0	0	0	0
LegalDocuments	0	0	0	0	2	0	1	1	575	0	0	0	0	1	2	0	0	0	0
LoanDocuments	0	0	4	0	0	1	0	0	0	463	0	0	0	0	0	0	0	1	0
MedicalRecords	0	0	0	0	1	0	0	1	1	1	640	0	0	0	0	1	0	1	0
MeetingMinutes	0	0	0	0	0	0	0	0	0	0	0	521	0	0	0	1	0	0	0
Miscellaneous	0	0	0	0	0	0	0	1	0	0	0	0	552	0	0	1	0	0	0
Payslips	0	0	0	0	0	0	0	0	0	0	0	0	0	502	0	1	1	0	2
PurchaseOrders	0	1	0	0	0	0	0	0	0	0	0	0	1	0	582	0	0	1	0
Receipts	0	0	1	3	0	0	0	0	0	0	0	0	0	1	0	625	0	0	0
ShippingDocuments	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	518	0	0
TaxDocuments	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	456	0
UtilityBills	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	1	693

Analysis:

- **Accuracy of 99.26%** indicates that the classifier correctly categorizes the majority of filenames.
- **Precision and Recall** scores demonstrate a balanced ability to minimize both false positives and false negatives.
- **F1-Score** of 81.5% reflects a strong harmonic mean between precision and recall, affirming the classifier's reliability.
- **Fast Processing Time** ensures that the classifier can handle large volumes of documents efficiently, meeting scalability requirements.

3.1.5 Conclusion

The simple classifier, enhanced by a fine-tuned BERT model and trained on a carefully designed synthetic dataset, offers an effective and scalable solution for filename-based file classification. Its capability to capture semantic nuances and address common inconsistencies in filenames makes it a dependable tool for automating document processing workflows.

While the classifier demonstrates strong performance on synthetic data, its effectiveness on real-world data remains uncertain due to the absence of access to real-life datasets during development. This limitation highlights the need for further testing and refinement with real-world examples to fully validate its generalizability. Nevertheless, the current implementation successfully meets the immediate project requirements.

3.1.6 Future Work

While the simple classifier demonstrates strong foundational performance, there are several avenues for future enhancements that could significantly improve its effectiveness in real-world scenarios:

- **Expansion of the Dataset:** A more extensive and diverse dataset would enable the model to capture a wider variety of filename patterns and variations. Incorporating additional categories and more complex filename structures would enhance the classifier's ability to generalize across different use cases.
- **Inclusion of Real-Life Data:** Integrating real-world datasets, in addition to synthetic data, would provide the model with authentic examples of filename discrepancies and complexities. Real-life data often contain unique patterns and anomalies that synthetic data might not fully replicate, thereby improving the classifier's robustness and accuracy in practical applications.
- **Handling More Variations:** Introducing a broader range of filename variations, such as different languages, uncommon abbreviations, and specialized industry-specific terminology, would further refine the classifier's semantic understanding and contextual awareness.
- **Scaling the Dataset Size:** Training on a larger dataset, despite the associated costs, could lead to significant performance improvements. A bigger dataset would allow the model to learn from more examples, reducing overfitting and enhancing its ability to handle unseen filename variations effectively.
- **Cost Optimization Strategies:** To address the constraints of training on larger datasets, future work could explore cost-effective strategies such as using more efficient data augmentation techniques, utilizing transfer learning more extensively,

or optimizing model architectures to require less computational power without compromising performance.
- **Continuous Learning and Adaptation:** Implementing mechanisms for the classifier to continuously learn from new data as it becomes available would ensure that the model remains up-to-date with evolving naming conventions and industry-specific requirements.
- **More Suitable Base Model:** Fine-tune a pretrained model already tailored to the industry's specifics.

By addressing these areas, the simple classifier can be further refined to achieve higher accuracy and reliability, making it even more valuable for real-world document processing tasks.

3.2 Advanced Classifier

While the simple classifier effectively categorizes files based on filenames, its reliance solely on filename limits its accuracy and applicability, especially when filenames are ambiguous or non-descriptive. To address these limitations, an advanced classifier was developed to analyze the actual content of the documents, thereby enhancing classification precision and robustness. This section details the design approach, dataset gathering, model fine-tuning, and the outcomes of the advanced classifier.

3.2.1 Design Approach

The advanced classifier aims to overcome the shortcomings of the simple filename-based approach by leveraging the document's intrinsic content. Initially, the project explored the UDOP (Unifying Vision, Text, and Layout for Universal Document Processing) model developed by Facebook, which integrates image and text inputs for document processing. However, the implementation of Optical Character Recognition (OCR) with UDOP proved problematic due to the high inaccuracy rates in the extracted text, which adversely affected classification performance.

To circumvent the dependency on OCR, extensive research led to the adoption of the DONUT (Document Understanding Transformer) model. DONUT is a pre-trained model based on the Swin Transformer architecture, specifically designed for document AI tasks. Its OCR-free approach processes document images directly, making it ideal for scenarios where OCR accuracy is unreliable.

Key Enhancements:

- **OCR-Free Processing:** Eliminates the dependency on OCR, reducing errors associated with text extraction.

- **Document Understanding:** Utilizes spatial and visual features of documents to improve classification accuracy.
- **Pre-trained Robustness:** Builds on the strengths of the pre-trained DONUT model, which has been trained on a large and diverse dataset.

3.2.2 Dataset Gathering

A comprehensive and well-augmented dataset is crucial for fine-tuning the DONUT model to suit specific classification tasks. The dataset for the advanced classifier was sourced from reputable platforms to ensure diversity and quality.

Sources:

- **Hugging Face:** Provided access to the DONUT model and various datasets tailored for document understanding tasks.
- **Roboflow:** Supplied a pre-augmented dataset with applied techniques such as contrast adjustment, brightness modification, and rotation. These augmentations enhance the model's ability to generalize across different document conditions.

Augmentation Techniques:

- **Contrast and Brightness Adjustment:** Simulates varying lighting conditions.
- **Rotation:** Mimics documents scanned or photographed at different angles.
- **Noise Addition:** Introduces realistic imperfections found in real-world documents.

This curated and augmented dataset ensures that the DONUT model is exposed to a wide range of document variations, thereby improving its ability to accurately classify diverse document types.

3.2.3 Model Fine-Tuning

Fine-tuning the DONUT model involved several critical steps to tailor it to the project's specific classification needs.

Steps Involved:

1. **Dataset Formatting:**
   - **DONUT-Specific Format:** The gathered dataset was reformatted to comply with the DONUT model's input requirements, ensuring compatibility and optimal performance.
   - **Label Expansion:** The original classification labels from the pre-trained model were expanded to include additional categories relevant to the project's industry-specific needs.

2. **Data Processing:**
    - **DONUT Processor:** Utilized to tokenize and encode the dataset, preparing it for efficient ingestion by the model.
3. **Training Configuration:**
    - **Trainer API from Hugging Face:** Employed to manage the training process, applying its robust infrastructure for efficient model optimization.
    - **Hyperparameter Tuning:** Adjusted parameters such as learning rate, batch size, and epochs to balance training speed and model performance.
4. **Fine-Tuning Execution:**
    - **Training Process:** The pre-trained DONUT model was fine-tuned using the processed dataset, allowing it to adapt to the specific nuances and classification requirements of the project.
    - **Validation:** A subset of the dataset was reserved for validation to monitor the model's performance and prevent overfitting during training.

Outcomes:

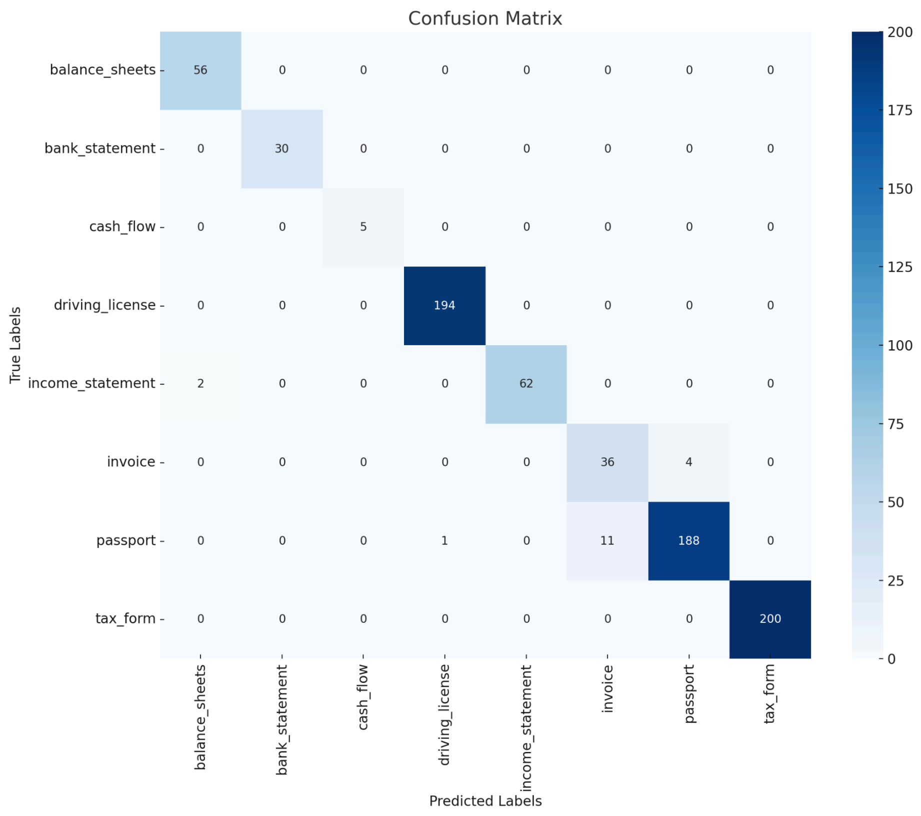
- **Enhanced Classification Accuracy:** Fine-tuning on the expanded and augmented dataset significantly improved the model's ability to accurately classify documents across a broader range of categories.
- **Increased Robustness:** The advanced classifier demonstrated strong performance in handling varied document formats and conditions, including those with complex layouts and backgrounds.
- **Contextual Understanding:** Exploiting DONUT's architecture, the model effectively captured both visual and contextual information, leading to more precise and reliable classifications.

3.2.4 Performance Metrics

The advanced classifier was evaluated using a comprehensive validation dataset to assess its effectiveness and reliability.

Metric	Value
Accuracy	97.47%
Precision	97.92%
Recall	97.47%

F1-Score 97.63%



Analysis:

- **High Accuracy of 97%** reflects the advanced classifier's superior ability to correctly categorize documents based on content.
- **Precision and Recall** scores of 97% and 97%, respectively, indicate a strong balance between minimizing false positives and false negatives.
- **F1-Score of 97%** demonstrates a robust harmonic mean of precision and recall, underscoring the classifier's overall reliability.
- **Test Data Limitations**: A closer analysis of the confusion matrix reveals that some labels have insufficient representation in the test data, leading to skewed metrics for these categories. This limitation highlights the need for a more diverse and comprehensive test set.
- **Future Improvement**: Evaluating the model on a larger and more representative dataset that incorporates real-life variability and edge cases would provide a more accurate assessment of the classifier's true performance and generalizability.

3.2.5 Conclusion

The advanced classifier significantly enhances the file classification system by analyzing document content directly, thereby overcoming the limitations of filename-based approaches. By using the DONUT model's OCR-free architecture and fine-tuning it with a diverse and augmented dataset, the classifier achieves high accuracy and robustness. This advancement ensures more reliable and precise document categorization, catering to the complexities of real-world document processing workflows.

3.2.6 Future Work

While the advanced classifier demonstrates impressive performance, several avenues exist for further improvement and optimization:

- **Expansion of the Dataset:**
    - **Larger and More Diverse Datasets:** Incorporating a larger dataset with more variations would enable the model to handle an even wider range of document types and formats, enhancing its generalization capabilities.
    - **Synthetic Data Generation:** Implementing more sophisticated synthetic data generation techniques, such as leveraging the DONUT model's synthetic generator, can create additional training examples that mimic real-world document variations.
- **Inclusion of Real-Life Data:**
    - **Authentic Datasets:** Integrating real-life document datasets alongside synthetic data would provide the model with genuine examples of filename discrepancies and complex document layouts, improving its performance in practical applications.
- **Advanced Data Augmentation:**
    - **Stable Diffusion Techniques:** Utilizing models like Stable Diffusion to generate synthetic documents with realistic backgrounds can help the classifier become more robust against variations introduced by photographing documents in different environments.
    - **Enhanced Augmentation:** Further augmenting the dataset with diverse transformations, such as varying lighting conditions, adding noise, and simulating wear and tear, can improve the model's resilience to real-world imperfections.
- **Preprocessing Enhancements:**
    - **Document Cropping:** Developing a pre-processing step to accurately crop documents from their backgrounds before classification can enhance the model's focus on relevant content, especially for documents with intricate backgrounds.

    - **Automated Background Removal:** Training a separate model to detect and remove backgrounds can streamline the classification process and improve accuracy for documents captured in less controlled environments.
- **Continuous Learning and Adaptation:**
    - **Incremental Training:** Implementing mechanisms for the model to continuously learn from new data as it becomes available ensures that it stays updated with evolving document formats and industry-specific requirements.
    - **Feedback Loops:** Establishing feedback systems where misclassifications are reviewed and used to further train and refine the model can lead to ongoing performance improvements.
- **Cost Optimization Strategies:**
    - **Efficient Training Techniques:** Exploring methods such as transfer learning more extensively or optimizing the model architecture can reduce computational costs while maintaining or enhancing performance.
    - **Scalable Infrastructure:** Utilizing cloud-based solutions with dynamically scalable resources enables efficient training on larger datasets while maintaining cost-effectiveness. A detailed discussion on AWS implementation is provided later in the report.

By addressing these future work areas, the advanced classifier can achieve even higher levels of accuracy and reliability, making it an indispensable tool for automating complex document processing tasks in diverse and dynamic environments.

4 Frontend Application

A user-friendly frontend application was developed to provide an accessible interface for interacting with both the simple and advanced classifiers. This application facilitates file uploads and displays classification results, enabling users to seamlessly utilize the classification services.

4.1 Design and Features

The frontend application was designed with simplicity and functionality in mind, focusing on ease of use and basic error handling.

- **User Interface:** Provides a clean and intuitive interface for uploading files and viewing classification results.
- **File Upload:** Supports various file formats, allowing users to upload documents for classification.

- **Basic Validation:** Implements checks to ensure that uploaded files are of the correct type and exist, providing informative error messages for invalid inputs.
- **Result Display:** Shows the classification outcome from both the simple and advanced classifiers, allowing users to compare results.

4.2 Technology Stack

The initial frontend application was built using Flask, a lightweight Python web framework, chosen for its simplicity and ease of integration with the backend classifiers.

- **Framework:** Flask
- **Languages:** Python, HTML, CSS
- **Libraries:** Bootstrap for basic styling

4.3 Limitations

While the Flask application serves as an effective testing tool, it is not production-ready. Its primary purpose is to validate the functionality of the classifiers and provide a foundation for future frontend development.

4.4 Conclusion

The development of the simple Flask-based frontend application successfully demonstrated the ability to interact with both classifiers in a user-friendly manner. Although functional for testing purposes, it highlights the need for a more robust and scalable frontend solution for deployment in a production environment.

4.5 Future Work

To enhance the frontend application and transition it from a testing tool to a production-ready solution, the following improvements are proposed:

- **Modern Frontend Framework:**
  - **React Native with Expo:** Develop a cross-platform mobile application using React Native and Expo to provide a seamless user experience on both iOS and Android devices.
  - **State Management:** Implement Redux for efficient state management, ensuring scalable and maintainable code.
- **Enhanced Styling:**
  - **Tailwind CSS:** Utilize Tailwind CSS to achieve a more polished and responsive design, improving the overall user interface aesthetics.
- **Robust Backend Integration:**
  - **AWS Services:**

        - **Amazon Cognito:** Manage user authentication and authorization securely.
        - **Amazon API Gateway:** Create and manage RESTful APIs to connect frontend requests with backend services.
        - **Amazon SageMaker:** Deploy the advanced classifier model, ensuring scalable and reliable model serving.
        - **Amazon RDS & S3:** Use Amazon RDS for relational database needs and Amazon S3 for efficient file storage.
        - **Amazon SNS:** Implement notification services for real-time updates and alerts.
        - **AWS Amplify & CloudFront:** Host the full-stack application using AWS Amplify and deliver content globally with Amazon CloudFront CDN for low latency.
- **DevOps and CI/CD:**
    - **Amazon CodePipeline & CodeBuild:** Establish continuous integration and continuous deployment pipelines to automate testing, building, and deployment processes, ensuring rapid and reliable updates.

By implementing these enhancements, the frontend application will transition into a scalable, secure, and user-friendly platform capable of supporting large-scale document classification tasks in real-world environments.

5 Deployment and Managing Strategy

5.1 Deployment Strategy

The fine-tuned advanced classifier model was deployed on **Amazon SageMaker**, selected for its scalability and cost-effectiveness. SageMaker offers multiple deployment options, each with distinct advantages and limitations:

- **Real-Time Inference:** While real-time endpoints provide immediate results, the inference time for our advanced classifier was excessively long, leading to timeouts. This issue was compounded by the constraints of the free tier, which did not offer sufficient GPU resources to handle the model's computational demands effectively.
- **Asynchronous Inference:** Suitable for longer inference tasks, asynchronous endpoints queue jobs and manage them more efficiently. However, this approach necessitates extensive use of **S3** for data storage and introduces additional steps,

such as uploading data to S3, invoking the inference job, and retrieving results. Additionally, costs accrue continuously as the endpoint remains active, making it less ideal for smaller user bases.

- **Serverless Inference:** Serverless options present a pay-as-you-go model with minimal setup, charging only for actual inference time. Despite its benefits, the serverless approach was limited by a maximum memory allocation of 3GB (a restriction of the free tier), which is insufficient for the advanced classifier's 6GB requirement. Moreover, cold start latency posed potential performance issues.
- **Batch Transform:** Identified as the most suitable option for our needs, Batch Transform offers scalability and cost efficiency by charging only for the resources used during processing. This method is ideal for handling large-scale deployments, such as processing over 100,000 documents daily, aligning with Heron Data's operational demands.

Batch Transform Workflow:

1. **Client Request:** A batch of data is submitted via a RESTful API hosted on **AWS API Gateway**.
2. **Data Preparation: AWS Lambda** functions process the request, preparing and storing the data in an **S3 bucket**.
3. **Batch Processing:** SageMaker Batch Transform processes the data, performing classification, and stores the results back in S3.
4. **Result Retrieval:** The Lambda function monitors the S3 bucket, retrieves the results upon completion, and returns them to the client through API Gateway.
5. **Security: AWS IAM** manages roles and permissions to ensure secure access. For handling sensitive data, **AWS KMS** (Key Management Service) can be implemented, although it was not utilized in this project.

Model Packaging and Deployment:

The fine-tuned DONUT model was packaged into a `.tar.gz` archive, adhering to AWS SageMaker's requirements for model artifacts. This package included the model's serialized weights and any necessary configuration files. SageMaker's prebuilt environments eliminated the need to containerize the code manually, simplifying the deployment process. The prebuilt environments provided by SageMaker ensured compatibility and streamlined integration, allowing for a seamless deployment experience without the additional overhead typically associated with containerization required for services like ECS and EC2.

Availability of AWS Files:

All AWS-related files, including the packaged model archive and deployment scripts, are available in the project repository.

This deployment strategy effectively balances scalability, cost, and complexity, making it suitable for Heron Data's requirements. By leveraging SageMaker's Batch Transform, the project ensures efficient processing of large document volumes while maintaining cost-effectiveness and operational reliability.

5.2 Management Strategy - CI/CD

Effective project management and deployment processes were critical to the successful implementation and maintenance of the classification system. Although the project was undertaken solo, management strategies were implemented and simulated to reflect best practices in a full-scale team environment.

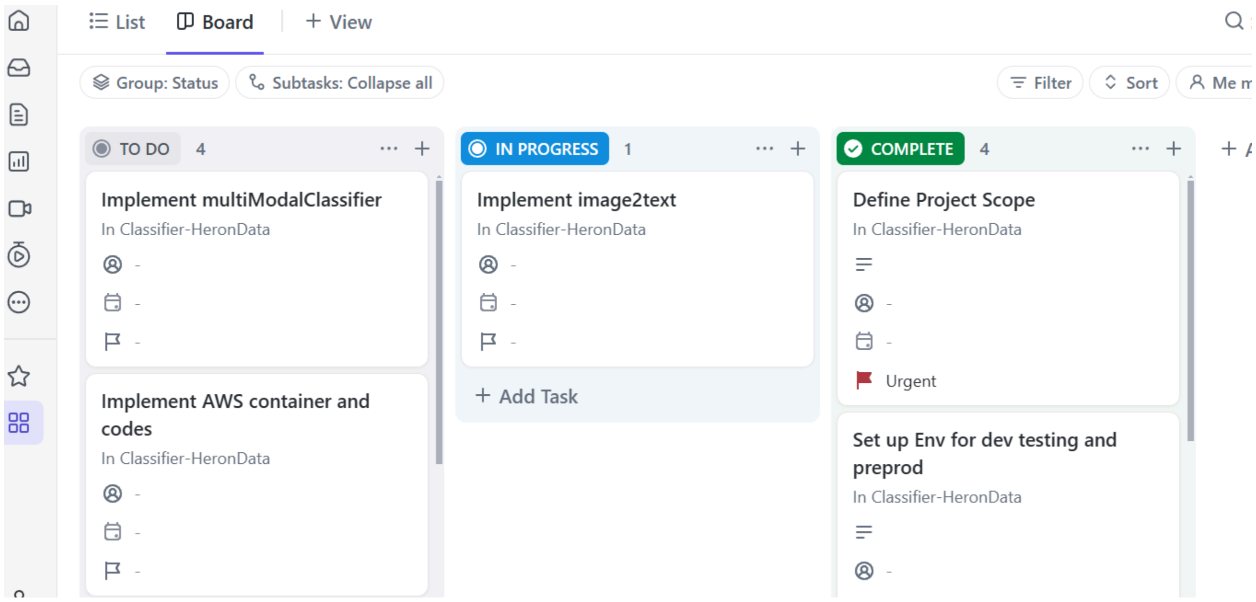
5.2.1 Agile Development:

- **Sprint Planning:** At the beginning of each week, a list of tasks (sprint) was created and prioritized, facilitating organized workflow management.
- **Task Tracking:** Tasks were tracked using **ClickUp**, a project management tool that allowed for task listing, prioritization, assignment, and progress monitoring, simulating a collaborative team setting.
- **Sprint Review:** At the end of each week, a review was conducted to assess completed tasks, evaluate sprint outcomes, and identify areas for improvement in subsequent sprints.

5.2.2 Git Branching Strategy:

- **Branch Structure:** The repository followed a structured branching strategy, starting from the **production (prod)** branch to the **development (dev)** branch, and further to feature-specific branches.
- **Feature Isolation:** Each feature was developed in its own branch to prevent conflicts and ensure isolated development, allowing for seamless integration once features were completed.
- **Merge Process:** Completed feature branches were merged into the dev branch. After successful integration and testing, changes were then merged from dev to test, and subsequently to pre-prod for final validation before deployment to prod.
- **CI/CD Integration:** This branching strategy facilitated **Continuous Integration and Continuous Deployment (CI/CD)**, ensuring that code was systematically tested and

deployed. Automated tests and deployment scripts were integrated to enhance



TO DO 4

Implement multiModalClassifier
In Classifier-HeronData

Implement AWS container and codes
In Classifier-HeronData

IN PROGRESS 1

Implement image2text
In Classifier-HeronData

COMPLETE 4

Define Project Scope
In Classifier-HeronData
Urgent

Set up Env for dev testing and preprod
In Classifier-HeronData

reliability and reduce deployment risks.

Screenshot of clickUp

5.2.3 CI/CD Implementation:

- **Continuous Integration:** Automated testing pipelines were set up to run unit and integration tests upon each commit, ensuring code quality and functionality.
- **Continuous Deployment:** Deployment scripts automated the process of deploying updates to SageMaker and other AWS services, reducing manual intervention and accelerating the release cycle.
- **Version Control:** Git was utilized for version control, enabling tracking of changes, rollback capabilities, and collaborative development practices.

Simulation of Team Practices:

Despite being a solo project, these management strategies were meticulously simulated to mirror the workflows and collaboration techniques used in larger teams. This approach not only ensured disciplined project progression but also demonstrated the scalability and effectiveness of these strategies in a professional setting.

By implementing and simulating these management strategies, the project achieved organized development, efficient task management, and streamlined deployment processes. These practices lay a solid foundation for future scalability and maintenance, ensuring that the classification system can be effectively managed and expanded as operational demands grow.

6 Evaluation and Testing

Ensuring the reliability and effectiveness of the HeronData File Classifier was a crucial aspect of the project. Due to time constraints, the testing process is ongoing, and only a subset of the planned test functions have been initiated using `pytest`. The majority of the comprehensive tests remain to be implemented to fully validate the system's robustness, scalability, and security.

6.1 Testing Overview

The evaluation strategy focuses on verifying the core functionalities of the simple and advanced classifiers, the frontend application, and the deployment pipeline. The testing approach encompasses:

- **Functionality Testing**: Ensuring that individual components perform as expected.
- **Performance Testing**: Assessing the system's efficiency and response times.
- **Robustness Testing**: Evaluating the classifiers' ability to handle various edge cases and erroneous inputs.
- **Integration Testing**: Verifying seamless interaction between different system components.
- **Security Testing**: Ensuring data protection and secure access controls.

6.2 Implemented and Planned Test Functions

While the majority of the test functions are still under development, the following outlines some of the key tests that have been or will be implemented to ensure comprehensive coverage:

6.2.1 Simple Classifier Tests

- **Filename Generation**
    - `test_filename_generator_insertions()`: Verify that random characters are correctly inserted into base keywords.
    - `test_filename_generator_deletions()`: Ensure that characters are properly deleted from base keywords.
- **Basic Classification**

- `test_classify_filename_correct()`: Test the classifier's ability to accurately categorize well-named files.
- `test_classify_filename_with_typos()`: Assess the classifier's performance with filenames containing typos or minor alterations.

6.2.2 Advanced Classifier Tests

- **Model Loading**
  - `test_donut_model_loading()`: Confirm that the DONUT model loads without errors and is ready for inference.
- **Basic Classification**
  - `test_classify_document_basic()`: Evaluate the classifier's ability to correctly categorize a small set of document images.

6.2.3 Frontend Application Tests

- **File Upload**
  - `test_file_upload_valid_formats()`: Ensure that supported file formats are accepted and processed correctly.
  - `test_file_upload_invalid_formats()`: Verify that unsupported file formats are rejected with appropriate error messages.
- **Result Display**
  - `test_result_display()`: Check that classification results from both classifiers are accurately displayed to the user.

6.2.4 Deployment Pipeline Tests

- **Batch Transform Workflow**
  - `test_batch_transform_end_to_end()`: Simulate the entire Batch Transform process from data submission to result retrieval to ensure seamless operation.

6.2.5 Future Test Functions

To achieve comprehensive evaluation and ensure the system's robustness, the following additional test functions are planned:

- **Simple Classifier**
  - `test_classify_filename_edge_cases()`: Handle unusually long filenames, special characters, and non-English names.
  - `test_performance_metrics()`: Measure accuracy, precision, recall, and processing time on larger and more diverse datasets.
- **Advanced Classifier**

    - `test_classify_document_various_formats()`: Ensure accurate classification across different document formats and layouts.
    - `test_stress_testing()`: Evaluate the classifier's ability to handle high volumes of concurrent classification requests.
- **Frontend Application**
    - `test_usability()`: Gather user feedback to identify and address interface usability issues.
    - `test_load_performance()`: Assess the application's performance under heavy usage to identify potential bottlenecks.
- **Deployment Pipeline**
    - `test_security_controls()`: Verify that access controls and data protection measures are effectively enforced.
    - `test_api_gateway_handling()`: Ensure that API Gateway correctly routes requests and handles errors gracefully.
- **CI/CD Pipeline**
    - `test_automated_deployments()`: Confirm that continuous integration and deployment pipelines function correctly, including automated rollbacks in case of failures.
    - `test_integration_with_sagemaker()`: Ensure seamless interaction between the CI/CD pipeline and SageMaker for model deployments.

6.3 Conclusion

While initial testing efforts have validated some fundamental aspects of the HeronData File Classifier, extensive testing remains to be completed to ensure the system's full reliability and performance. The ongoing development of test functions will focus on covering all critical areas, including edge cases, performance under load, and security measures. Comprehensive testing is essential to validate that the classifier meets all project objectives and operates effectively in real-world environments.

Future testing activities will aim to implement the remaining test functions, thereby enhancing the system's robustness, scalability, and security to deliver a dependable document classification solution for HeronData.

7 Conclusion

The development of the HeronData File Classifier is a very important component in building a Document AI service. By addressing the limitations of traditional filename-based classification methods, this project has successfully implemented a dual-classifier system that enhances both accuracy and scalability.

The **simple classifier**, fortified with a fine-tuned BERT model, demonstrates robust performance in categorizing files based on semantic understanding of filenames, even when faced with ambiguously named or poorly formatted files. The simple classifier lays a strong foundation for efficient fast document processing, effectively mitigating common challenges associated with filename discrepancies.

Building upon this foundation, the **advanced classifier** leverages the DONUT model's OCR-free architecture to analyze the actual content of documents. With an impressive accuracy, the advanced classifier significantly surpasses the simple classifier in precision and reliability, especially in handling complex and varied document formats. This enhancement ensures that the classification system is not only accurate but also resilient against the intricacies of real-world document variations.

The integration of both classifiers into a Flask-based frontend application further exemplifies the project's commitment to usability and accessibility. Although the frontend currently serves as a testing tool, it successfully facilitates interaction with both classifiers, providing a clear pathway for future enhancements and deployment.

Deployment strategies utilizing AWS SageMaker's Batch Transform capabilities have been meticulously planned to ensure scalability and cost-effectiveness, aligning with HeronData's operational demands. Additionally, the implementation of robust management strategies, including Agile development practices and a structured Git branching strategy, underscores the project's adherence to industry best practices, even within a solo development environment.

While the project has achieved some milestones, ongoing evaluation and testing remain critical to ensuring the system's full reliability and performance. The foundational work laid out in this project opens up scope for further work.

8 Future Work

To further enhance the HeronData File Classifier and ensure its continued effectiveness and adaptability in dynamic real-world environments, the following areas have been identified for future development:

1. Integration of a Feedback System for Continuous Learning

Implementing a feedback mechanism will allow the classifier to learn from user corrections and real-world usage patterns. By collecting feedback on misclassifications and incorporating this data into the training pipeline, the system can continuously refine its models, leading to incremental improvements in accuracy and robustness over time. This adaptive learning approach ensures that the classifier remains relevant and effective as document types and naming conventions evolve.

2. Expansion and Diversification of the Dataset

Enhancing the diversity and volume of the training dataset will enable the classifiers to generalize better across a wider array of document types and formats. This includes:

- **Incorporating Real-World Data**: Integrating authentic documents from various industries to capture unique patterns and anomalies not present in synthetic data.
- **Generating More Sophisticated Synthetic Data**: Utilizing advanced data generation techniques to simulate a broader range of filename variations and document layouts, thereby improving the classifiers' ability to handle edge cases.

3. Advanced Data Augmentation Techniques

Employing state-of-the-art data augmentation methods, such as Stable Diffusion, can create highly realistic synthetic documents with diverse backgrounds and conditions. This will enhance the advanced classifier's resilience to variations introduced by different scanning or photographing environments, further improving its classification accuracy.

4. Development of a Production-Ready Frontend Application

Transitioning the current Flask-based frontend into a more robust, scalable, and user-friendly platform is essential for broader deployment. Proposed enhancements include:

- **Adopting Modern Frontend Frameworks**: Utilizing React Native with Expo to develop a cross-platform mobile application, providing seamless user experiences on both iOS and Android devices.
- **Implementing Advanced Styling and State Management**: Leveraging Tailwind CSS for responsive design and Redux for efficient state management to ensure a polished and maintainable user interface.
- **Enhancing Backend Integration**: Integrating additional AWS services such as Amazon Cognito for secure user authentication, Amazon API Gateway for robust API management, and AWS Amplify for streamlined hosting and content delivery.

5. Scaling and Optimizing Deployment Infrastructure

Further optimization of the deployment pipeline will ensure that the system can handle increasing workloads efficiently. This includes:

- **Exploring Cost-Efficient Training Techniques**: Utilizing transfer learning more extensively and optimizing model architectures to reduce computational costs without compromising performance.
- **Enhancing Scalable Infrastructure**: Implementing dynamic resource scaling on AWS to accommodate fluctuating workloads, ensuring consistent performance during peak usage periods.

6. Comprehensive Evaluation and Testing

Expanding the testing framework to cover all critical aspects of the system is paramount for ensuring reliability and security. Planned enhancements include:

- **Implementing Extensive Test Suites**: Developing additional test functions to cover edge cases, performance under load, and security vulnerabilities.
- **Automating Testing Processes**: Integrating automated testing into the CI/CD pipeline to ensure continuous validation of system integrity with each update.

7. Enhancing Security Measures

Ensuring the protection of sensitive data and maintaining secure access controls are essential for the system's integrity. Future work will focus on:

- **Implementing AWS KMS for Data Encryption**: Enhancing data security by encrypting data at rest and in transit using AWS Key Management Service.
- **Strengthening IAM Policies**: Refining AWS Identity and Access Management (IAM) roles and permissions to enforce the principle of least privilege, minimizing potential security risks.

8. Exploring Alternative Classification Models

Investigating and integrating alternative or complementary machine learning models could further improve classification performance. This includes:

- **Evaluating Other Transformer Architectures**: Exploring models beyond BERT and DONUT that may offer superior performance for specific classification tasks.
- **Ensemble Methods**: Combining multiple models to leverage their strengths and achieve higher overall accuracy and robustness.

By addressing these future work areas, the HeronData File Classifier will evolve into a more sophisticated, reliable, and adaptable solution, capable of meeting the growing and changing demands of automated document processing in various industries. Continuous improvement and innovation will ensure that the system remains at the forefront of document classification technology, delivering sustained value to HeronData and its stakeholders.