Voice-to-Text Classification of Maritime Incidents

Tango x

Eleni-Theodora Lyssea (p2822418) &
Natalia Tsamtsouri (p2822435)
Machine Learning and Content Analytics - PT | MSc in Business Analytics | AUEB

Introduction

In today's maritime industry, voice communication plays a critical role in reporting technical issues on board. However, the process of reporting technical issues (via email, messages, or "final" reports) is often proved time-consuming and problem-solving. Our project explores how Natural Language Processing (NLP) and Machine Learning (ML) can be used to automatically classify such voice reports into predefined technical categories, helping crews and shipping companies respond faster and more effectively.

Our Project

We have developed a prototype application that processes voice reports from crew members and classifies them into categories such as Accident to person(s), Damage / Loss Of Equipment, Loss Of Control, Grounding / Stranding, Contact, Collision, Fire / Explosion, Flooding / Foundering and Capsizing / Listing. Using a fine-tuned transformer model "DistilBERT", trained on simulated and augmented data, the system learns to identify the core issue described in each report, even when the language used varies significantly across crew members.

Our Vision / Goals

Our ultimate goal is to support the digitalization of ship operations by turning unstructured verbal data into structured insights in real time. With this application, the seafarer could immediately submit their report on their mobile or tablet, which is viewed in real time by the person in charge in the office.This way, actions are coordinated quickly and efficiently, while all incidents are recorded in a common summarized table, categorized by type and degree. This project serves as a first step toward integrating AI-powered decision support into daily maritime operations.

Data Collection

The dataset "occurences.csv" used in this project was abstracted from real-world maritime incident reports from Marine Accident Investigation Branch (MAIB Data Portal - UK). The raw data was manually processed and the final dataset used included only 3 columns, "**Main event 1", "Description**", and "**Severity**". No sensitive or personally identifiable information was included in the dataset.

Dataset Overview

The dataset consists of **4,354 incident reports** related to maritime occurrences. Each row corresponds to an individual incident and includes a free-text description, structured metadata e.g., location, date, vessel type, as well as manually annotated labels for **incident category** and **severity level**.

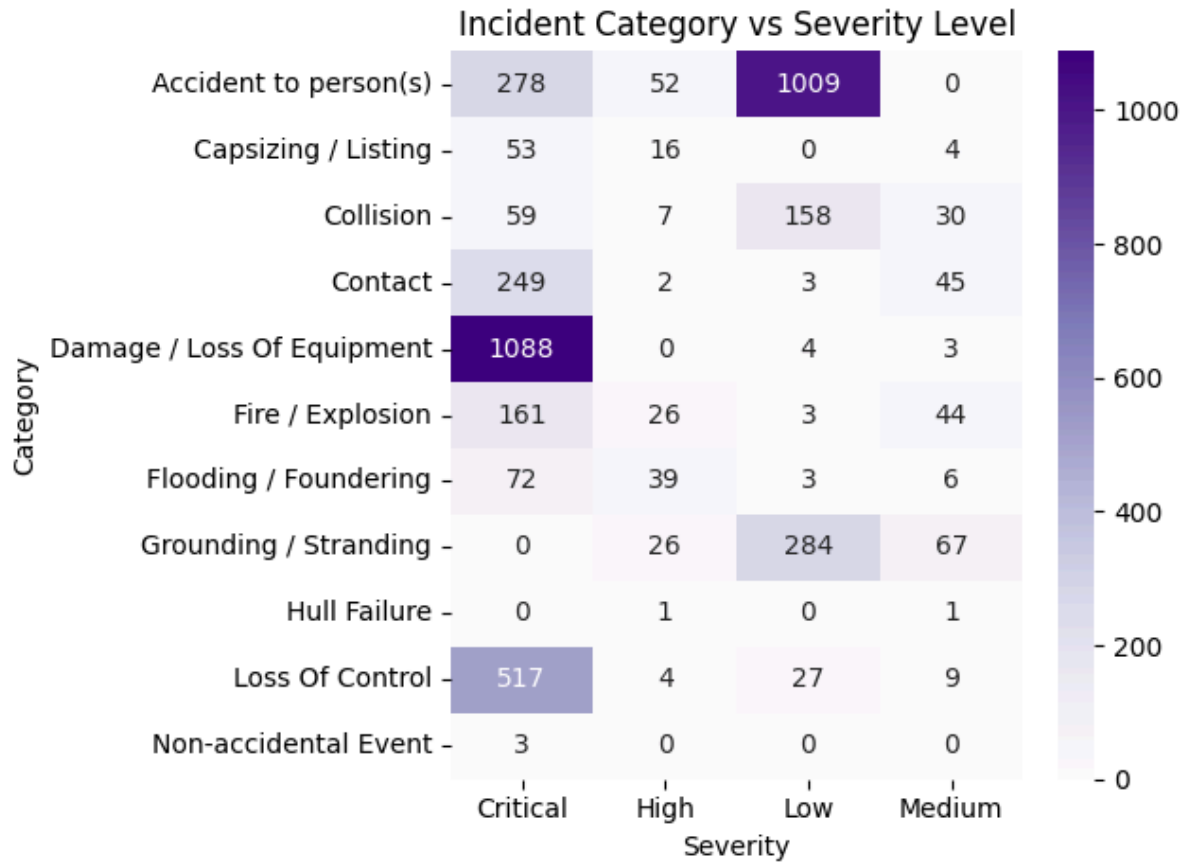
Regarding the initial dataset selection, *occurrences.csv* file contained multiple columns describing maritime incidents. For the purposes of this project, we selected:

Main event 1 which will be used as the target **Category**, **Description** which will be used as the **Report text** input for the model and **Severity** which will be the severity label to be

predicted. These columns were renamed to **category**, **report**, and **severity**, respectively. We excluded *Main event 2* and *Main event 3* as they contained limited subcategories and insufficient representation for training based on our project goal.

Descriptive Statistics Summary

Severity is moderately imbalanced, with most cases labeled as "Critical" (~2,481) and "Low" (~1,490), and fewer as "High" (174) or "Medium" (209). The most common categories include “Damage / Loss of Equipment,” “Loss of Control,” and “Accident to person(s),” while others appear less frequently. A category * severity crosstab; see ‘Plot: *Incident Category vs Severity Level*’, reveals clear patterns, with some categories strongly associated with specific severity levels. These observations informed the preprocessing and modeling strategy.



Data Processing

Data Cleansing

1. **Null values**: Rows containing null values were removed. There was only one instance in the category column, thus the whole row was excluded from our working dataset.
2. **Non-informative entries**: Any records where the report contained the phrase *"see MAIB report when available"* were removed, as they lacked descriptive content and would not contribute meaningfully to model training. There were 43 records containing this phrase in both description and short description.

3. **Category filtering**: All 11 original category values were retained, except for extreme outliers with very low sample counts, which are listed below:
    - *non-accidental event* (3 records)
    - *hull failure* (2 records)

These were merged into larger, semantically relevant categories to ensure sufficient representation of the training data. More specifically, non-accidental event value was merged into Accident to person(s), and hull failure into Damage / Loss Of Equipment.

Data Normalization and Mapping

Severity levels originally had four distinct labels:

- *less serious*
- *serious*
- *very serious*
- *marine incident*

These were mapped to the following four standardized severity classes for normalization reasons: **low**, **medium**, **high**, and **critical**, to be easier to handle, review, and present data outcomes and ensure consistent labeling across the dataset.

Dataset Splitting

The cleansed dataset was split into:

- **Training set (70%)**
- **Validation set (15%)**
- **Test set (15%)**

as this structure allows for both model optimization and unbiased performance evaluation.

Bias and Imbalance Analysis

Exploratory data analysis revealed significant class imbalance in both category and severity. While majority classes contained over 120 samples, minority classes had significantly fewer examples, which could bias the model toward majority categories. These extreme minority categories were merged into broader classes, as described above while for the ones with less than 150 records we proceeded with the data augmentation technique, as presented below.

Data Augmentation and Oversampling

Two main approaches were used to address observed imbalance:

1. **Oversampling**: Minority class samples were repeated with replacement to match the frequency of majority classes.

2. **Data Augmentation**: Applied on the text reports to synthetically create new variations of minority class entries (e.g., synonym replacement, paraphrasing).

Model Evaluation with and without Augmentation

The training pipeline involved:

- Train_category.py, train_severity.py – model training on the original dataset.
- Evaluate_category.py, evaluate_severity.py – validation set evaluation.

For the **category classification model**, the baseline already achieved high performance (Accuracy and Weighted F1 ≈ 0.91). After applying text augmentation and fine-tuning with adjusted hyperparameters, the results remained stable, confirming the model's robustness even in low-support classes.

For the **severity classification model**, the baseline achieved a satisfying performance (Accuracy ≈ 0.83 and Weighted F1 ≈ 0.81), which is slightly worse than the category model due to the visible class imbalance. Afterwards, the weighted loss method was used to focus more on rare classes but the Accuracy & F1 results were 0.77 and 0.79, respectively. So, we didn't proceed with the weighted loss method and after applying text augmentation and fine-tuning with adjusted hyperparameters, the total results remained almost stable, but individually checking each severity class by comparing confusion matrices of each different solution, there was a clear improvement.

After applying augmentation, we repeated all actions with the new dataset:

- train_category_augm.py, train_severity_augm_2.py – re-training on augmented datasets.
- evaluate_category_augm.py, evaluate_severity_augm2.py – validation with augmented datasets.

The re-trained models showed clear improvements:

- **Category model**: Accuracy ≈ 0.91, Weighted F1 ≈ 0.91
- **Severity model**: Accuracy ≈ 0.83, Weighted F1 ≈ 0.81

These results; see also Table 1, confirm that augmentation improved classification performance for both category and severity metrics without introducing harmful bias.

Model Variant	Key Training Setup	Accuracy	Weighted F1	Macro F1
Category (baseline)	train_category.py – DistilBERT, max_len=128, batch=32/16, epochs=3, LR=5e-5	0.91	0.91	0.89
Category (augmented)	train_category_augm.py – DistilBERT, same setup, augmented data (train_augmented.csv), epochs=3	0.91	0.91	0.89
Severity (baseline)	train_severity.py – DistilBERT, original data, default LR=5e-5, epochs≈3	0.83	0.81	0.61
Severity (augmented)	train_severity_augm_2.py – DistilBERT, augmented data(train_augmented.csv), weighted loss, batch=32/32, epochs=5	0.83	0.81	0.64

Table 1: Model comparison for both metrics (Category & Severity)

Methodology

In this project we constructed an NLP classification system that can identify and classify maritime incident reports. The classification architecture is based on **fine-tuning the distilbert-base-cased pre-trained network model**, which is faster in training than BERT but has strong performance. We used Hugging faces trainer API for rapid prototyping and incorporated some training improvements like smaller batch size (32), shorter max sequence length (128), and early stopping. We fine-tuned it within 2-4 epochs with learning rate tuning and weight decay regularization. For the category & severity classification models, we also applied **text augmentation** to balance the classes and re-trained the model. All training was executed using the scripts train_category_augm.py and train_severity_augm_2.py.

Evaluation was performed using the evaluate_category_augm.py script on the validation dataset. The results were measured using **Accuracy**, **Macro and Weighted F1-scores**, and **Confusion Matrix**, so performance can both be checked on majority and minority classes. The final model reached about 91% accuracy and a weighted F1-score of 0.91, showing stable performance across all classes. Even low-support categories like "Capsizing / Listing" achieved F1 values higher than 87%, which confirms the robustness of the training pipeline, see ‘Figure 1’ below.

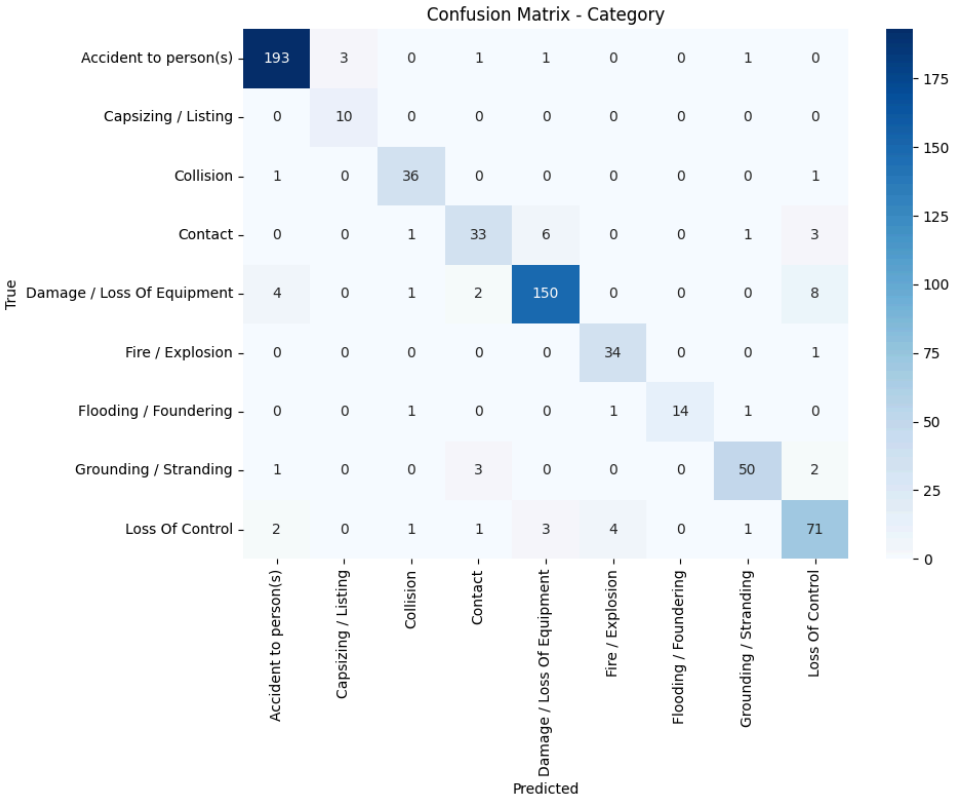


Figure 1: Confusion Matrix - Category

The **severity model**, the evaluation of which was performed using the evaluate_category_augm2.py script on the validation dataset, achieved lower performance (0.83 accuracy), likely due to more subjective labeling and fewer distinct linguistic patterns in the report text for that task; see 'Figure 2' below.

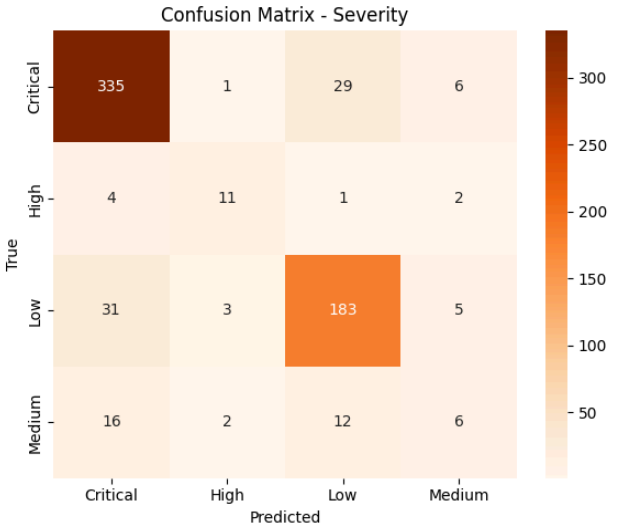


Figure 2: Confusion Matrix - Severity

Optional Local Recording (record_wav.py):

For testing purposes, we also created a small Python script using the *sounddevice* library that records short audio clips from the microphone and saves them as ‘my_report.wav’. While the main application already supports recording through the browser, this script can be used locally to quickly generate test files for the pipeline.

Speech-to-Text Pipeline (ASR)

To support voice-based incident reporting, we developed a simple speech-to-text module using Hugging Face’s whisper-small model. The **asr_pipeline.py** script loads an automatic speech recognition (ASR) pipeline that can process audio files (WAV/MP3) and return transcribed text.

This step is important because it converts verbal incident reports—such as those recorded by crew members—into text, which can then be processed by our NLP models for category and severity classification.

The model automatically detects whether a GPU is available and handles long audio files by breaking them into smaller chunks (30 seconds).

This ASR step serves as the entry point of the pipeline, converting speech into text that can then be processed by our classification models.

Final Pipeline Integration

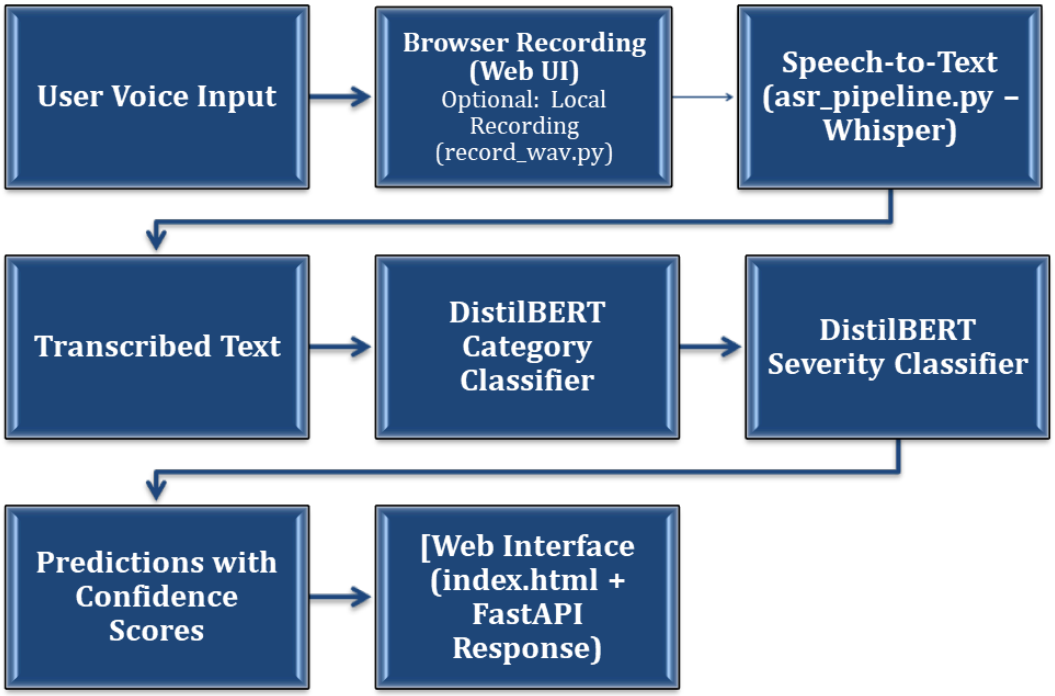
To integrate all components into a single, user-facing process, we developed the **run_pipeline.py** script. This script automates the full pipeline: it takes an input audio file, applies speech-to-text transcription using the Whisper model (asr_pipeline.py), and then feeds the resulting text into two fine-tuned transformer models to classify the incident's **category** and **severity**. The models were loaded from pre-trained checkpoints, and predictions were returned along with their confidence scores. This setup simulates a realistic use-case, where a user can submit a spoken report and immediately receive structured classification output.

Application & User Interface - NautiVoice

To make the pipeline usable in real-world cases, we developed a voice based web application named NautiVoice. The system allows users such as crew members or inspectors to record incident reports, automatically transcribe them into text using ASR, and receive immediate classification of both incident category and severity level.

The main application is implemented in **FastAPI**, and is extended with a responsive **HTML/CSS/JavaScript** front-end that runs locally in the browser. The backend handles audio input, speech recognition using **Whisper**, and classification using the fine-tuned **DistilBERT** models trained earlier in the project.

Functional Overview

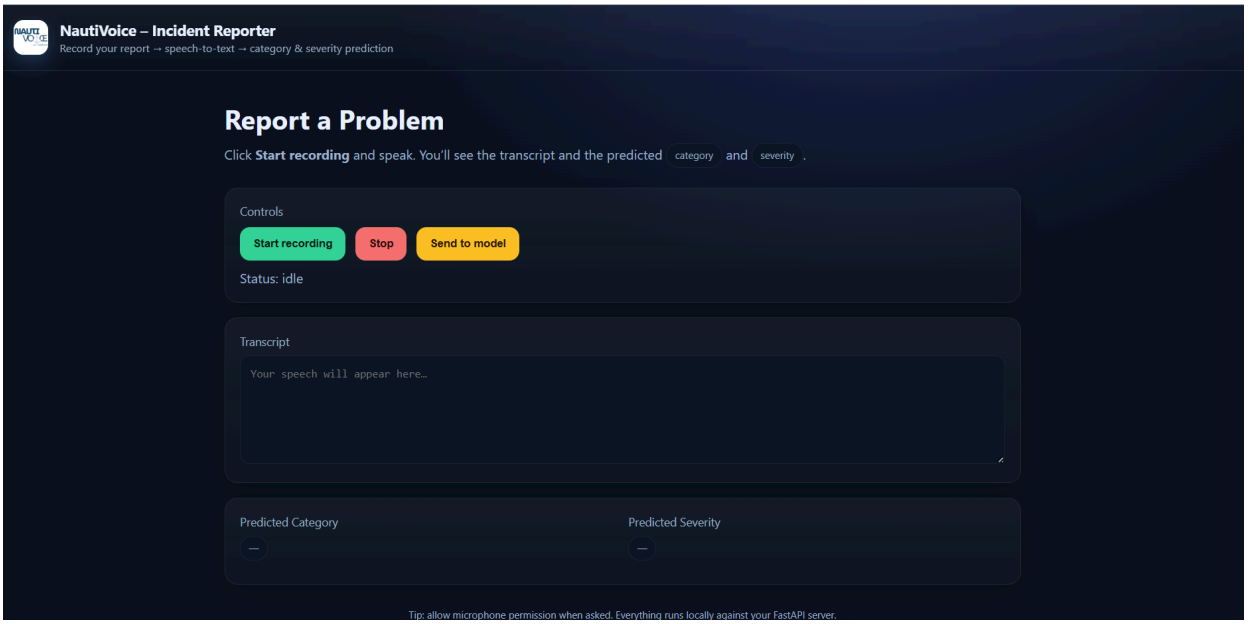


Audio Recording (Web UI): Users can record short voice reports through their microphone directly in the browser. Audio is sampled at 16 kHz and encoded as .wav format before being sent to the backend.

Backend (FastAPI): Receives the .wav file via a POST /predict request and then transcribes the audio using the openai/whisper-small ASR model. Applies text classification using the two fine-tuned Transformer models for Category and Severity and returns structured results in JSON format, including predicted labels and confidence scores of the rest labels as well.

Frontend (index.html): Displays the transcribed text, predicted labels, and a bar chart with probability scores using animated HTML elements.

Example Use Case

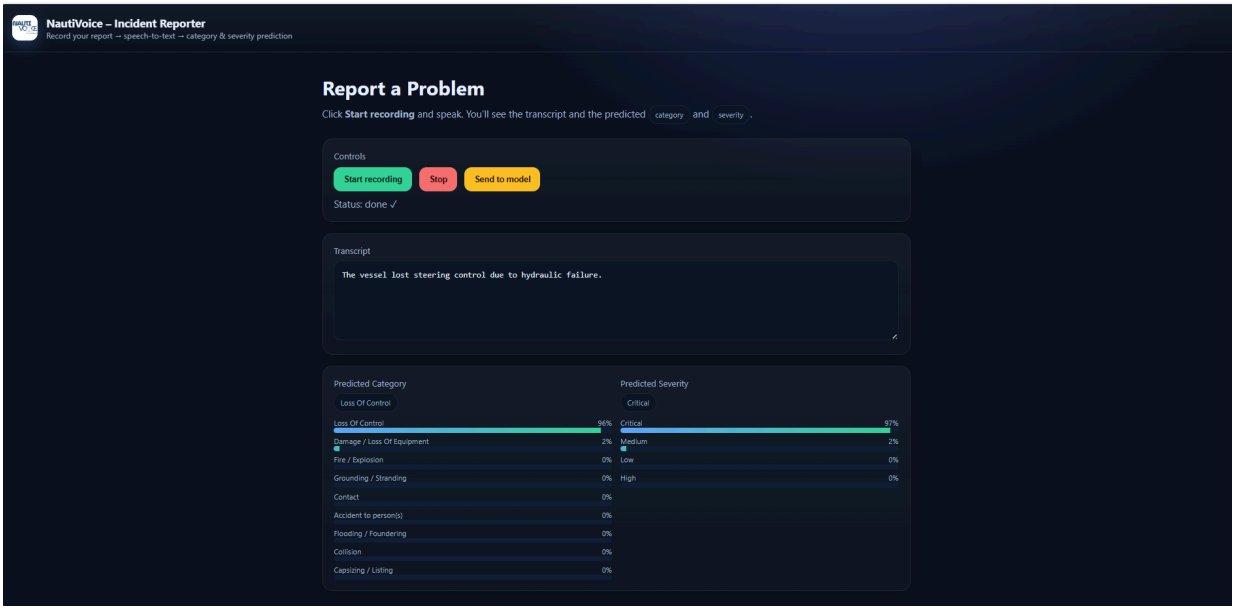


The user clicks on “Start Recording” option and when this option becomes active (indicated by a blinking icon), they start describing the incident:

e.g.:**“The vessel lost steering control due to hydraulic failure.”**

Then the user chooses “Stop” and “Send to model”

The app sends the audio to the FastAPI server and the system responds with the predicted outcomes. The user is able to review and even correct the recorded text written on the screen.



Qualitative & Error analysis

There was a manual testing conducted of 30 specific records included in the validation dataset (val.csv) with different combinations of categories & severities. Below (Figure 3 & Figure 4), there are attached two barcharts of both category and severity of the records selected for the error analysis, where we can see that there is a relative balance of the category levels and a relative imbalance of the severity levels since critical class has almost 3 times more records than the rest severity levels based on the class portions of the whole 'validation' dataset, as well.

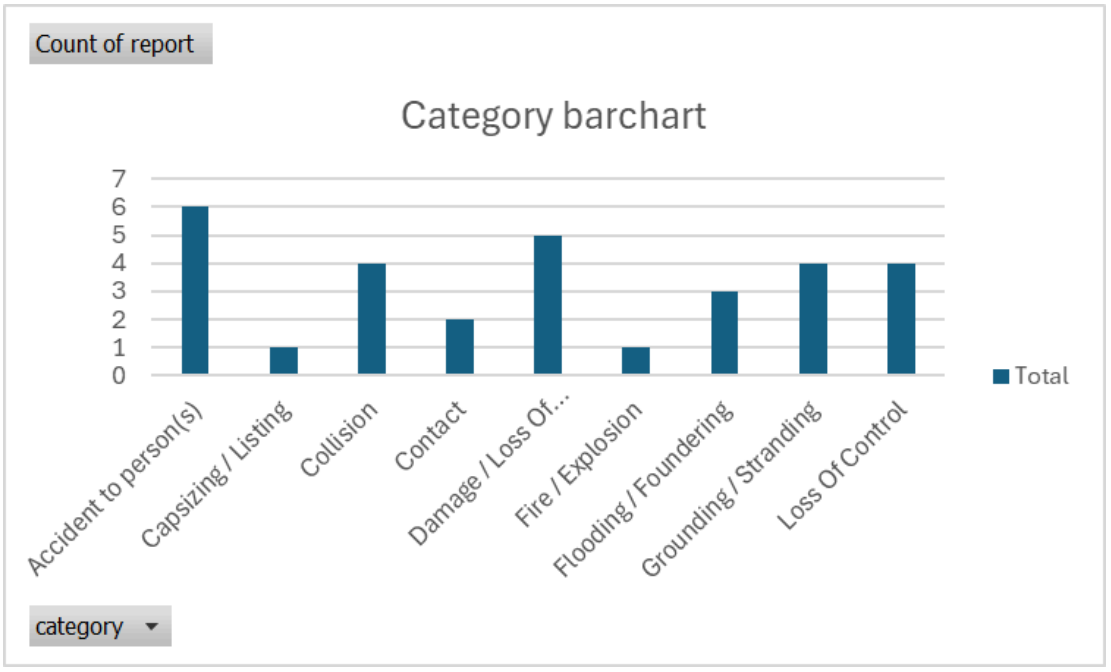


Figure 3: Category Barchart

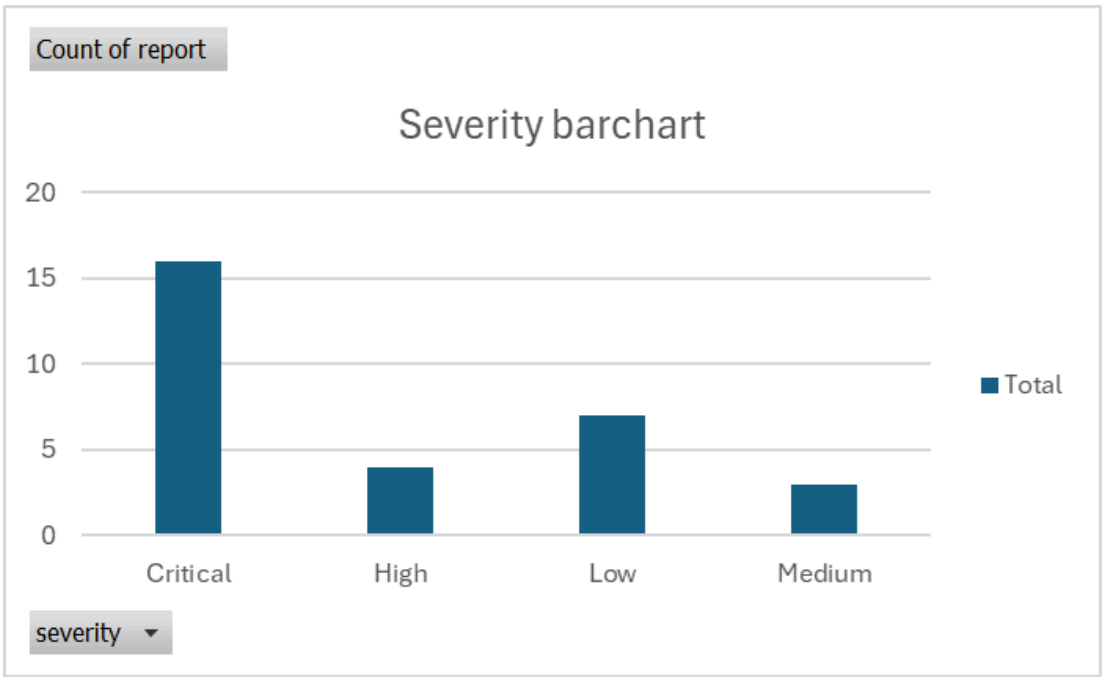


Figure 4: Severity Barchart

Regarding the results, out of 30 evaluated records, 24 were correctly predicted in both category and severity. In 3 cases, the category was correctly predicted but the severity was misclassified, while in another 3 cases, the severity was correct but the category was misclassified. Importantly, no records were found with both category and severity misclassified, which indicates that the model consistently captures at least one of the two dimensions correctly and results in a high performance outcome.

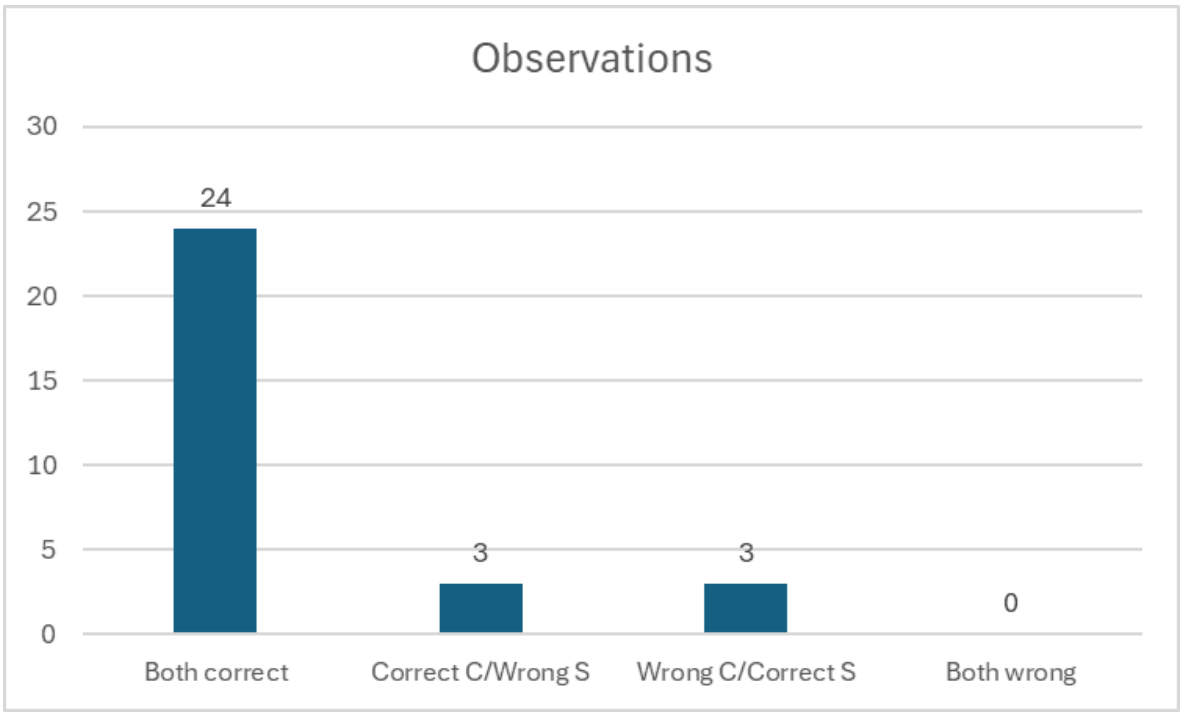


Figure 5: Predictive Results Barchart

Discussion and Future Work

Based on the work completed for this project, we would consider some supplementary actions that could optimize the general functionality of *NautiVoice* App. First of all, Category and Severity classification models may be better trained with a focus on the minority classes of each different metric. Another extra step might be the construction of a dashboard which will retain and present classification outcomes coming from the *NautiVoice* records. This dashboard would provide aggregated views of incidents by type and severity, enabling managers to track trends, identify recurring issues, and support data-driven decision-making. Finally, it would be useful to add Named Entity Recognition (NER), so the system could automatically retrieve information like vessel names and locations from the reports. That way, users could search or filter incidents by ship or area, and gain more detailed insights.

Members/Roles

The project was carried out collaboratively by two team members, Elena Lyssea and Natalia Tsamtsouri, with a clear division of tasks but also continuous interaction and communication.

Elena was primarily responsible for the **Category Classification** component of the system, including dataset preprocessing (preprocess_occurrences.ipynb), train/validation/test splits, and the implementation of data augmentation for minority categories. She also developed the evaluation pipeline (evaluate_category_augm.py) and built the integration scripts (run_pipeline.py, app.py) that connected the ASR, classification models, and web interface. She also contributed to the writing of the final project report.

Natalia on the other side focused on the **Severity Classification** task, which was the second metric to predict. She worked on training and fine-tuning the BERT model for severity prediction, and applied weighted loss and data augmentation techniques to enrich the dataset. In addition, she took the lead in preparing the documentation, including the README file and the user guide.

Beyond our individual responsibilities, both of us collaborated closely on the annotation of the dataset, discussed the selection of models and hyperparameters, and jointly tested and refined the end-to-end pipeline. We also contributed to the design and structure of the upcoming presentation.

TimePlan

The development of the project was structured into several phases, spread over approximately two months. The preliminary timeline is outlined below:

Week	Activity
1–2	Data collection, cleaning, and preprocessing (annotation, augmentation, preparation of input features) & business case definition
3–4	Training and fine-tuning of the BERT-based models for both category and severity classification.
5	Integration of the ASR (Whisper pipeline) and initial implementation of the FastAPI backend.
6	API refinement, frontend/dashboard development, and connection between components.
7	Testing, error analysis, and evaluation of model performance.
8	Final report writing, documentation preparation, and presentation preprocess.