

Sentiment-Based Alerts (SenBA)

Leslie Dees, Jayesh Gupta, Naman Parikh, Yoo Bin Shin

Table of Contents

1. Introduction.....	3
1.1 Problem.....	3
1.2 Solution.....	3
2. Current Implementation.....	4
2.1 Design Overview.....	4
2.2 Dataset.....	7
1. Selection.....	7
2. Preprocessing.....	7
3. SMS Manual Labeling.....	7
2.2 Sentiment Analysis Models.....	8
1. Selection.....	8
2. Evaluation.....	10
3. Improvement.....	11
2.4 Third-Party Integration.....	11
1. Selection.....	11
2. App Development.....	12
3. Integration with model-side.....	15
3. Planned Extensions.....	16
3.1 Sentiment Analysis Models.....	16
3.2 Third-Party Integration.....	16
4. References.....	18
Appendix A: Milestone Completions.....	19

1. Introduction

1.1 Problem

Existing alert systems on mobile devices fail to inform users of a text message's content through audio notifications. When a phone receives a text message, the device usually either vibrates if on silent mode or plays one user-selected notification sound. However, this results in homogeneous notifications where the user is required to pick up their device and check the notification screen to see a brief summary of the first words in a text.

Whilst this problem may not be significant, a potential solution to this problem could be instrumental. Today, where notification sounds have become prevalent in one's day to day, a more informative sound can benefit the productivity of a user who is working remotely or behind the steering wheel. Another potential use-case of the content-aware alert sound is if a user has their phone on the "Do Not Disturb" setting, then they can only be alerted to a text if it has an emergency sentiment. Furthermore, this may also be a helpful feature for people with developmental disabilities by providing them with audial cues that can help them understand the sender's tone conveyed in a text message.

1.2 Solution

With Machine Learning (ML) becoming more common in our everyday lives, we aim to take advantage of online resources to build an app that aims to solve this problem. In particular, we have chosen to focus on Natural Language Processing (NLP). Current applications of NLP include use cases such as Google Search for better search results, predictive text when writing emails, and Google Translate for language translation. Our focus within the field of NLP will be on sentiment analysis.

We propose an alternative solution in the form of Sentiment-Based Alerts–SenBA. This will provide smart customization based on the text message's sentiment, purpose, or importance through a model-selected sound that better conveys its content. We aim to improve a user's experience with intelligent alert systems by delivering a text message's overall sentiment or importance through seamless audial cues, helping them be more productive when working on other tasks.

2. Current Implementation

2.1 Design Overview

SenBA is a sentiment-based alerting system that processes text messages in real-time with a sentiment ensemble model to intelligently select a notification sound that provides more information for the user.

SenBA features two main components, the Third-Party API Integration and the Sentiment Analysis Model (See Figure 1).

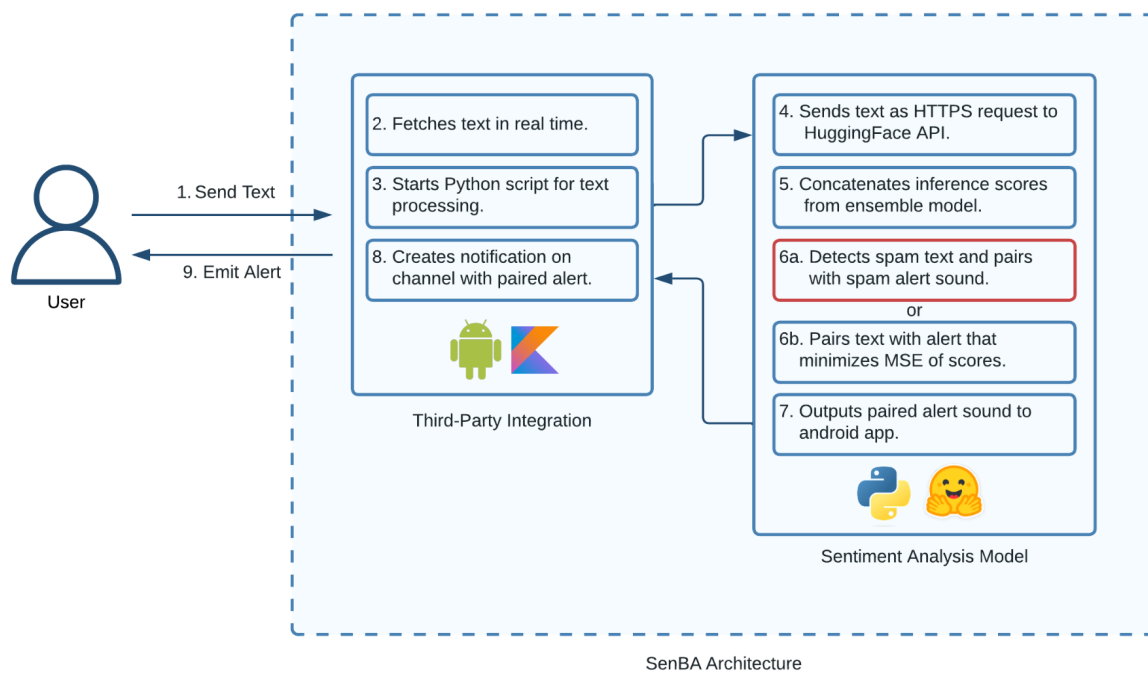


Figure 1: SenBA Flowchart

The Third-Party API Integration encompasses the front-end component that the user interacts with. Currently, in the MVP, it takes the form of an Android mobile messaging app implemented in Kotlin. The text message the user receives is fetched in real-time with the Android API and is sent as input for the sentiment ensemble model for inference. Once processed, the Android API creates a notification on the channel of the intelligently paired alert and finally emits the alert.

The Sentiment Analysis Model deals with the processing and alert-pairing of the text message. This component was implemented using Python and additional packages such as Scikit-Learn, and pre-trained models available on HuggingFace were used. The model architecture is broken down into two primary sections. The first is the ensemble model that is used to generate scores from the input text. An HTTPS request is sent to the models via the HuggingFace API, and the models perform a percentage-based classification on the text input.

The first model of the ensemble is a spam detection model (Roberta-base-finetuned-sms-spam-detection) that outputs scores for spam and not-spam. The model performs binary classification; if the spam score is over a threshold value, the text message is classified as spam, and without further sentiment pairing, the spam alert sound is emitted.

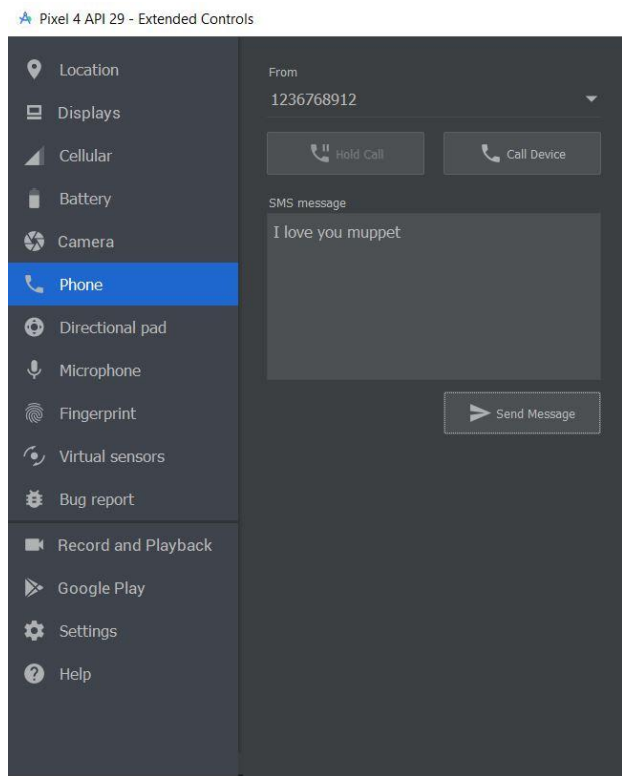
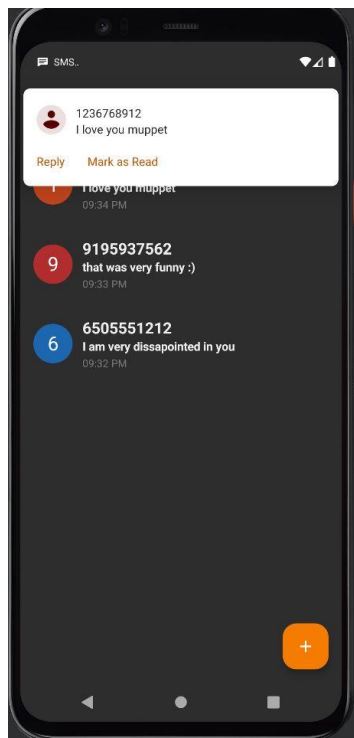
The remaining ensemble is created with two sentiment models—the RoBERTa sentiment model and the Distilbert emotion classification model. The models respectively provide a score vector of three positive, negative, and neutral sentiment labels and six emotion labels that is then used in the second stage of the model architecture.

The second section consists of the alert-pairing algorithm. This is constructed with an MSE minimization function between the current input's ensemble model sentiment score and a list of forty pre-labeled alert sounds' sentiment scores. The notification sound whose sentiment score has the minimum MSE with the input's score is selected, then provided back to the app-end to use as its audible notification sound.

The final MVP of SenBA can be deployed on an Android mobile or demonstrated on the Android Studio Emulator (See Figure 2). Users can fully interact with the app by sending texts of different sentiments and hearing each sentiment-based alert sound emit in real-time. Our implementation can be found here: <https://github.com/jg435/Senba>.

The demo will involve iterations of texts that express different sentiments. The desired steps will be as follows:

1. Text message is sent in real-time.
2. Text message is processed by pre-trained sentiment analysis model ensemble.
3. Sentiment analysis model predicts sentiment of text correctly in reasonable time.
4. Outputted sentiment is paired with well-fitting pre-labeled alert sound.
5. The selected alert sound is played on the device that receives the text message.



```
I/Playing channel: happy2
```

Figure 2: Screenshot of app receiving a message (left), Screenshot of sending a text message to the emulator (right), Screenshot of log print of alert channel played (bottom)

2.2 Sentiment Analysis Models

1. Selection

The main objective of the model selection was to create an ensemble of sentiment-related models that in conjunction with each other would create a holistic prediction of a text message's sentiment. Publicly available pretrained models obtained from sources like HuggingFace and Kaggle were evaluated for this purpose. Since these pretrained models contained large quantities of data that are rich in diversity of corpus sources, along with being large transformer models well-suited for zero-shot tasks, we decided to focus on finding an ideal model rather than training our own from scratch.

The two models that were ultimately chosen were twitter-XML-roBERTa-base-sentiment and distilbert-base-uncased-emotion. These were chosen for three separate reasons. Firstly, they produce two separate sentiment results that we require. These are base sentiment scores (positive, negative, neutral) and emotion scores (sadness, joy, love, anger, fear, surprise) respectively. By utilizing these models in tandem, we can create a wider feature base to select our notifications and theoretically increase performance. Secondly, they utilize information obtained from separate sources. The roBERTa model utilizes Twitter data finetuned for sentiment analysis. Andriotis et al. found that current methods used for Twitter sentiment analysis are very useful for depicting the emotional polarity of SMS messages, which validates the usage of this fine-tuned model for our purpose (Smartphone Message Sentiment Analysis). The distilbert model similarly uses the Twitter-Sentiment-Analysis dataset provided by HuggingFace. Finally, as described in the Evaluation section, these two models performed best on our custom MSE detection.

The BERT model used as the architecture basis is a pre-trained language model that utilizes Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). During MLM, BERT learns to predict missing words while in NSP it attempts to pair sentences based on continuations of the first sentence. The RoBERTa variation removes this NSP training objective and trains with much larger mini-batches and learning rates. Distilbert, on the other hand, attempts to distill BERT by training a base with 40% fewer parameters, making it faster while still retaining the majority of its performance.

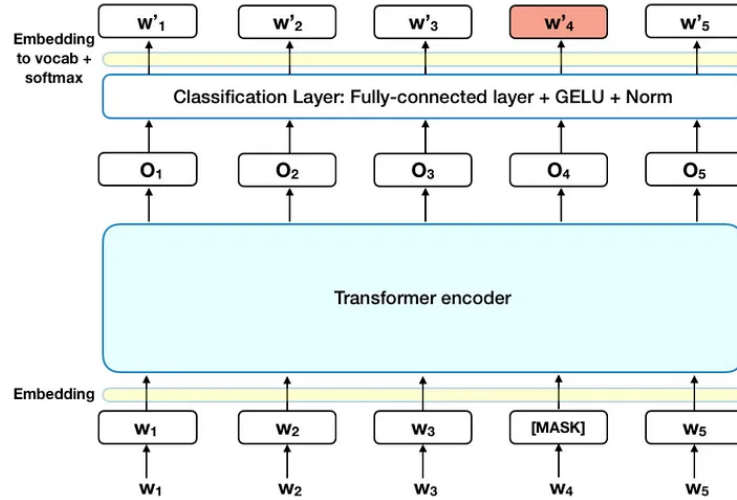


Figure 3: Masked Language Model used to predict maskings for BERT

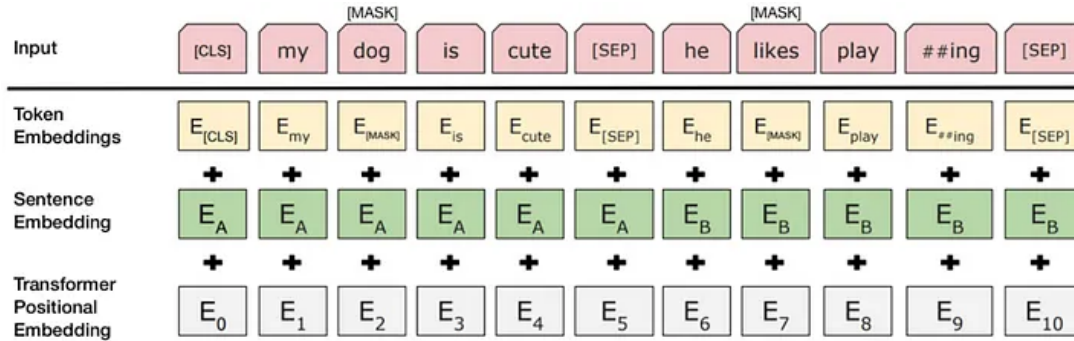


Figure 4: Next Sentence Prediction within BERT training process

These two models are combined in our ensemble, which feeds our input text into both models simultaneously. Therefore, we receive the sentiment scores from the RoBERTa model and also the emotion scores from Distilbert, providing us with 9 total features we can use to determine matching between texts and notification sounds. This model ensemble is formed from a single pipeline containing these models in parallel.

2. Evaluation

One of the objectives of the project pipeline was to optimize for model selection. To maximize accuracy, the metric used in the process was to select the model with the relatively least MSE (Mean Squared Error). Since picking only one

model based on this metric could increase the chances of overfitting, two models were picked and used in a customized pipeline.

Hugging Face was used to research sentiment analysis models: it not only has a wide assortment of models (with seamless deployment options) but also the models are open-source (with high credibility as evident from the number of downloads, likes, etc.). Using this context, five models were shortlisted, and were fed through the codebase. There, the Hugging Face API was used to connect them with the text messages dataset and compute the MSE score, The details have been tabulated below. The top two models with the relatively lowest MSE (highlighted in green in Table 1 below).

Table 1: Different Sentiment models found and their corresponding performance on the text message dataset

Model Name	MSE (Mean Squared Error)
<i>twitter-XLM-roBERTa-base for Sentiment Analysis</i>	6.957308178697062
<i>distilbert-base-uncased-emotion</i>	10.055352351115552
<i>SiEBERT - English-Language Sentiment Classification</i>	18.28619101346972
<i>RoBERTa-based model can classify the sentiment of English language text</i>	19.724432414665674
<i>bertweet-sentiment-analysis</i>	22.489619979215668

3. Improvement

Model improvement was a difficult task given the pre-trained regressive nature of the selected models. Fine-tuning on SMS data was an impossibility due to our limited corpus of 100 manually labeled text messages for emotion and sentiment. We attempted to research other methods to improve the models. Text style

transfer was explored, but corpus style transfer for text is not a well-researched area and attempts were unfruitful. Therefore, we relied on large quantity model testing for optimal model selection. Performance was also optimized through notification selection MSE minimization. By scaling the MSE of the roBERTa model, which dominated the pipeline's minimized MSE, we were able to drastically improve qualitative performance.

2.3 Dataset

Since SenBA primarily deals with text messages as input, it was paramount to use a dataset comprising this data form. The pipeline around this dataset is laid out as follows:

1. Selection

While other sources were browsed, Kaggle was primarily used to conduct the dataset selection search. In addition to providing a wide range of datasets on its platform, Kaggle also enables users to obtain the data and process it in a straightforward manner. SenBA makes use of UCI Machine Learning's "SMS Spam Collection Dataset" on Kaggle. Referring to its online description, this dataset is a collection of SMS tagged messages that have been collected for SMS Spam research. It contains a set of 5,574 English SMS messages, classified into either ham (meaning legitimate) or spam. In relation to the datasets that were used within the decided upon models, we focused on ensuring that there was a diverse and large corpus included in the pretrained models. This led to us choosing models that also contained 198 million tweets as their training data.

2. Preprocessing


Preprocessing was performed on the SMS spam dataset. We divided the data into only the "Ham" messages, which were the ones that were not labeled as spam. This provided us with 4827 text messages we could use for training. These text messages were subdivided as being obtained from the Grumbletext website, NUS SMS Corpus, Caroline Tag's PhD Thesis, and finally SMS Spam Corpus v.0.1 Big.

3. SMS Manual Labeling


It was unanimously agreed upon to select a model that would offer the lowest MSE (Mean Squared Error) when comparing its sentiment predictions to manually annotated data points. Hence, out of the 5,574 text messages in the above-mentioned Kaggle Dataset, after filtering out the spam messages, a random sample of 100 text messages was selected for human annotation. Every member in the team manually annotated each of those 100 messages i.e. the

message was given a score (between 0 and 1) representing the probability of that message corresponding to the given sentiment. The image below shows a snapshot of that process, wherein manual annotation was conducted for two of the shortlisted models. Moreover, for each of those two models, the probability values would sum to 1. The values displayed below are averaged across the scores given by the 4 team members - accounting for each member's choices evenly.


Text Message	Manual or Machine	positive	negative	neutral	love	joy	sadness	anger	surprise	fear
Life spend with someone for a lifetime may be meaningless but a few moments spent with someone who really love you means more than life itself...	Manual	0.7	0.2	0.1	0.5	0.25	0.1	0.05	0	0.1
U can call me now...	Manual	0.25	0.35	0.4	0.25	0.1	0.1	0.3	0.1	0.15
Boo I'm on my way to my moms. She's making tortilla soup. Yummmm	Manual	0.4	0.35	0.25	0.3	0.2	0.25	0.1	0.05	0.1
I wasn't well babe, i have swollen glands at my throat ... What did you end up doing ?	Manual	0.15	0.65	0.2	0.2	0	0.5	0.1	0.1	0.1
HEY DAS COOL... IKNOW ALL 2 WELDA PERIL OF STUDENTFINANCIAL CRISISISPK 2 U L&R!™	Manual	0.1	0.7	0.2	0	0.1	0.3	0.45	0.05	0.1
How much you got for cleaning	Manual	0.1	0.25	0.65	0	0.1	0.25	0.3	0.25	0.1
LmaoNice 1	Manual	0.65	0.05	0.3	0.15	0.5	0	0.05	0.3	0




**Non-Spam
Sampled Text**



**Human
Annotation**



**Roberta
Labels (1 dp)**



**Distillbert
Labels (1 dp)**

Figure 5: CSV file of manually labeled sentiment scores for text message data

2.4 Third-Party Integration

1. Selection

To determine which third-party messaging API to build SenBA on, we conducted preliminary research on several widely used messaging platforms. These included platforms like iMessage, Android SMS, Facebook Messenger, and WhatsApp, along with more business-catered platforms like Slack and Microsoft Teams.

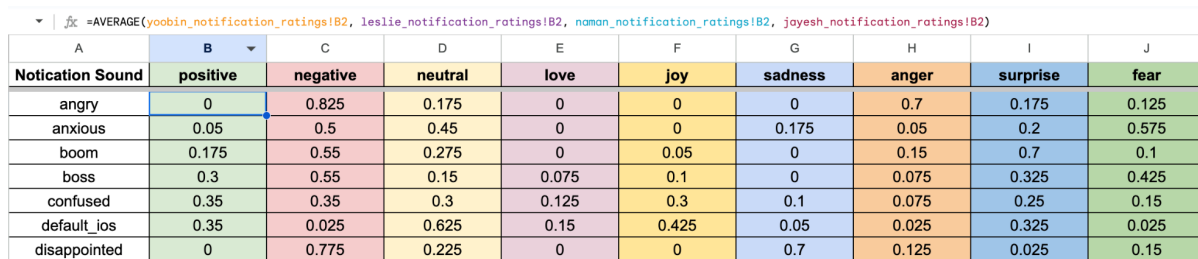
Although Android SMS may not be the most widely used messaging platform, the Android OS, being open-source, provided the most developer flexibility, crucial for the rapid implementation of our ML-powered alert system. The Android API allowed access to read a text message before the notification is emitted and allowed programmatic modification of alert sound. Additionally, as Android is open-sourced, more resources were accessible online. Thus, we deemed that integration with the Android API would be a reasonable and sufficient proof-of-concept for SenBA.

2. App Development

The next step was to build SenBA's alert system on the Android API. Instead of implementing a messenger app from scratch, we worked off of an open-source implementation of a simple Android messaging App on GitHub and integrated our additional features.

The first feature implemented was the ability to modify the alert sound in real-time depending on the sentiment processed from the content of the text message. The Android API gives the user final control over the settings of a notification channel such as whether it is silenced or which alert sound is used. Thus, the workaround that we used to programmatically emit a custom alert sound per text was creating a notification channel for each distinct alert sound.

A total of 40 distinct alert sound mp3 files were downloaded from zedge.com, zapsplat.com, and notificationsounds.com. They were selected consciously to cover diverse sentiments of ranging intensities. Next, the alert sounds were manually labeled with sentiment scores by our four team members, and the mean values were used as their final sentiment scores (See Figure 4).



=AVERAGE(yoobin_notification_ratings!B2, leslie_notification_ratings!B2, naman_notification_ratings!B2, jayesh_notification_ratings!B2)									
A	B	C	D	E	F	G	H	I	J
Notication Sound	positive	negative	neutral	love	joy	sadness	anger	surprise	fear
angry	0	0.825	0.175	0	0	0	0.7	0.175	0.125
anxious	0.05	0.5	0.45	0	0	0.175	0.05	0.2	0.575
boom	0.175	0.55	0.275	0	0.05	0	0.15	0.7	0.1
boss	0.3	0.55	0.15	0.075	0.1	0	0.075	0.325	0.425
confused	0.35	0.35	0.3	0.125	0.3	0.1	0.075	0.25	0.15
default_ios	0.35	0.025	0.625	0.15	0.425	0.05	0.025	0.325	0.025
disappointed	0	0.775	0.225	0	0	0.7	0.125	0.025	0.15

Figure 6: CSV file of manually labeled sentiment scores for each alert sound

Another important feature was the ability to emit an alert sound that reasonably resembles the sentiment of the text message. This involved determining a pairing algorithm that selects one alert sound from the forty. First, we tested a pairing algorithm that calculates the total MSE, or the average squared distance, between the model-outputted sentiment score of the input text message and the manually labeled sentiment score of each alert sound. Simply, the alert sound whose sentiment score is closest to the input text's score is selected.

The statistical evaluation using MSE demonstrated considerable theoretical performance. However, when we conducted human evaluation of whether the alert-pairing was reasonable or not, we determined the practical performance to be lacking. For instance, informative texts with a neutral tone were emitting alert

sounds conveying a clear sentiment. This can be seen in Table 2 as when the “eating lunch rn” text is received, the quick alert sound is played rather than default ios alert sound. Since there were more emotion labels (6) than neutrality labels (3), we hypothesized that by calculating the MSE together, the emotion scores were dominating the neutrality scores in the pairing process.

To address this issue, we tested a pairing algorithm that calculated two separate MSE values for each alert sound—an MSE for the 3 positive, negative, and neutral labels and an MSE for the 6 emotion labels. This creates a combination of 78 MSEs (2 sentiment models x 39 alert scores excluding spam). To account for distilBERT’s naturally higher occurring MSE due to a wider range of sentiments being observed (as seen in Table 1), we multiplied a scaling factor of 0.4 to the MSE value of distilBERT to account for this. This scaling factor was a hyperparameter determined by testing a variety of different inputs with the outputted alert noise suggestions as well as comparing the final adjusted MSE values of distilBERT with RoBERTa’s.

The alert sound that minimizes one of the model’s MSE is selected. As alert sound and sentiment are ultimately subjective metrics, we determined this pairing algorithm performed more reasonably in the human evaluation and deployed it in our MVP. We can see the results of the two different MSE methods in Table 2 below.

Table 2: Demonstration of texts received and alert sounds generated by different MSE methods

Text Input	Pairing Algorithm 1 (Total MSE)	Pairing Algorithm 2 (Separate MSE)	Ideal classification
Eating lunch rn	quick	default_ios	default_ios
I am having a bad day	sad	sad	disappointed5
holy shit there is a fire	confused	negative2	emergency3
I love pancakes	positive	positive6	positive2
Hey this needs to be done ASAP	positive6	emergency	emergency
I fought with my girlfriend	angry	anxious	solemn
hahahaha	serious	positive6	funny
GO DUKE!	quick	positive6	positive2

3. Integration with model-side

We explored two options for integrating the Android app with the model-side. The first involved serving the models on our end, and the second involved sending requests to models already deployed on the HuggingFace server.

Initially, we tried to connect our app with Google Firebase where the models would be served by uploading TensorFlow Lite (tflite) files. When connected to wifi, the app would ping the server when it needs a model output. When the device is offline, since the latest version of the model would be locally deployed on the device, a model output would still be generated. Although this worked for lighter NLP models, we had issues trying to convert the pre-trained HuggingFace models we were using into tflite files. In addition, serving the models on Firebase meant the raw sentiment score vectors were to be dealt with on the Kotlin app-end, requiring the pairing algorithm to be implemented in Kotlin. However, as opposed to Python which offers more packages with built-in ML methods like MSE computation, Kotlin did not. Thus, we decided to explore an alternative route.

For the final product, we used the Chaquopy library, a Python SDK for Android, which allows users to run Python scripts in their Android apps (which would typically run Kotlin or Java instead). By doing so, instead of serving the models ourselves, we could simply send HTTPS requests to the HuggingFace API for the model inference results. We could also implement the pairing algorithm in Python using the sk-learn package. Having the entire pipeline—model inference and pairing—handled in Python improved productivity in testing, as it did not require building and running code on the app-end. However, one issue with this approach is that it requires constant internet access as we are calling the HuggingFace API for model outputs. Although a user may not always be connected to the internet, we can make this assumption as most people are either connected to wifi or have cellular data.

3. Planned Extensions

3.1 Sentiment Analysis Models

One large improvement that could be made to improve the model performance is to further develop textual style transfer approaches. Since the models that we selected were trained on social media data rather than SMS data, we assume that the sentiment fit is not perfect to our domain. Therefore, the idea of style transfer came up as a potential solution. Style transfer is widely available for political, sentiment, formality, and many other approaches. However, there are no style transfer approaches explored for changing corpus styles for broader approaches, such as SMS-to-Twitter. We could attempt to explore a non-parallel approach which would allow us to provide our entire text message corpus and a large twitter corpus to attempt to allow us to style transfer our inputs prior to model predictions to better fit the training data of the models.

We also could potentially add in sarcasm detection as a separate pipeline addition to the ensemble. Sarcasm detection models are already widely available and well explored, so implementation of a pipeline addition would be simple. This additional feature was not explored due to time constraints but could add breadth to the detection ability.

3.2 Third-Party Integration

A future feature that would be crucial to have is integration with other widely used messaging platforms such as iMessage, Whatsapp, Messenger, and Slack. This offers greater accessibility to SenBA by reaching a larger group of users.

Additional features can be implemented to make SenBA more informative. One method is to make SenBA account for other non-sentiment parameters, such as text length, contact's importance, and attached content. For instance, the pitch of an alert sound can be consistent for important contacts such as the user's family member or work manager. That way, the user is able to infer both the sender and the sentiment of their text. Another method is customizing an alert sound per text message. While the MVP is limited to the 39 manually collected and labeled alert sounds, an algorithm or a Conditional-GAN model can be implemented to generate a unique alert sound for each text message. Since alert sounds are essentially audio signals characterized by distinct features like timbre, pitch, loudness, etc., these features can be modified accordingly based on the sentiment scores.

Another feature is enabling sentiment-based alerts in silent mode. This can be achieved by emitting a different sequence or intensity of vibrations for different emotions. Another feature is allowing users to prioritize notifications by the classified sentiment. For instance, a user may choose to only hear notifications for texts when certain emotions are conveyed, such as happiness or emergency.

4. References

- [1] Almeida, T. (2012). SMS Spam Collection. UCI Machine Learning Repository. Retrieved from <https://archive.ics.uci.edu/ml/datasets/SMS%2BSpam%2BCollection>.
- [2] Cardiffnlp. (2022). Twitter-roBERTa-base for Sentiment Analysis. Hugging Face. Retrieved from <https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest>.
- [3] Bhadresh-savani. (2021). Distilbert-base-uncased-emotion. Hugging Face. Retrieved from <https://huggingface.co/bhadresh-savani/distilbert-base-uncased-emotion>.
- [4] Tibor Kaputa, Android SMS/MMS Sending Library. (2022). GitHub Repository. Retrieved from <https://github.com/tibbi/android-smsmms>.
- [5] Chaquopy: the Python SDK for Android. (2022). GitHub Repository. Retrieved from <https://github.com/chaquo/chaquopy>.
- [6] Maria Grandury. (2022). Roberta-base-finetuned-sms-spam-detection. Hugging Face. Retrieved from <https://huggingface.co/mariagrandury/roberta-base-finetuned-sms-spam-detection>.

Appendix A: Milestone Completions

- Preliminary Research & Choice of frameworks
 - Found pre-trained HuggingFace sentiment analysis models (*Yoo Bin*)
 - Found Sentiment140 raw dataset and explored Kaggle regression trainings (*Leslie*)
 - Found Spam Text Message Dataset (*Jayesh*)
 - Conducted research on different third-party messenger platforms (*Yoo Bin, Jayesh*)
- Model Selection
 - Explored ease of training our own model versus deploying model from HuggingFace (*Leslie*)
- Dataset Classification
 - Conducted manual labeling of text message data with sentiment scores (*Leslie, Naman*)
 - Manually collected alert sound database (40 sounds) from royalty-free websites (*Yoo Bin*)
 - Conducted manual labeling of collected alert sounds with sentiment scores (*All*)
- Mobile App Implementation
 - Found simple Android messenger Github Repo implemented in Kotlin (*Jayesh*)
 - Implemented feature to modify alert sound per text by creating notification channel for each alert sound (*Yoo Bin, Jayesh*)
- Model Pipeline Implementation
 - Model Testing Framework Development (*Leslie*)
 - Model Ensemble Pipeline Development (*Leslie*)
 - Individual Model Pipeline Development (*Leslie*)
 - Model Testing for Sentiment and Emotion Models (*Naman*)
 - Model Ensemble Pipeline Integration with Selected Evaluated Individual Models (*Leslie*)
 - Implemented pairing algorithm with total MSE that outputs paired alert sound file in Python (*Yoo Bin, Jayesh*)
- Mobile App Integration with Model Pipeline
 - Found Chaquopy as alternative solution, integrated package to app configuration, and wrote code that calls Python script to run model pipeline on Kotlin (*Yoo Bin*)
- Model Enhancement
 - Testing different pairing algorithms (total MSE vs separate MSE) (*Naman, Yoo Bin*)
 - Fine-tuning on subset of our labeled dataset (*Naman, Leslie*)
 - Implemented second iteration of pairing algorithm using separate MSEs (*Yoo Bin*)
 - Added spam detection model to pipeline (*Yoo Bin*)