**Smart Reply: Automated Response Suggestion for Email**

**Team Members:**

Kare Sindhu Gonda ([sindhukaregonda@my.unt.edu](mailto:sindhukaregonda@my.unt.edu))

Pranathi Akula ([PranathiAkula2@my.unt.edu](mailto:PranathiAkula2@my.unt.edu))

Vishnubotla Chandra Sekhar ([Chandrasekharvishnubotla@my.unt.edu](mailto:Chandrasekharvishnubotla@my.unt.edu))

**Workflow:**

As a team of three members, we worked collaboratively on the project, following a set workflow. We communicated regularly through Microsoft Teams, with weekly in-person meetings scheduled every Monday from 2pm to 5pm to discuss progress and challenges.

To ensure efficient collaboration and file sharing, we maintained the project documents and materials in Google Drive storage. We made sure that all team members had access to the necessary files and that everyone was up-to-date with any changes made to the project.

During the project development phase, we followed a structured approach to ensure that each team member was assigned specific tasks and responsibilities. We regularly checked in with each other to ensure that the project was on track and that deadlines were met.

Overall, our workflow helped us stay organized and collaborative, ensuring that the project was completed to a high standard.

**Project Abstract:**

The aim of this project is to develop an intelligent system called "Brilliant Answer" for generating concise email responses automatically. With the increasing number of emails received by individuals on a daily basis, it can become tedious and overwhelming to read and respond to each one individually. The proposed system uses Natural Language Processing (NLP) techniques to suggest appropriate email responses that are tailored to the content of the email.

The system is designed to provide users with a variety of semantically meaningful response ideas that can be used as complete email responses with just one press of a button. The system is capable of processing a large number of messages at a high throughput rate, making it a highly efficient solution for managing large volumes of emails.

This project will discuss the architecture of the Brilliant Answer system as well as the challenges encountered during its development, such as response variety and flexibility.

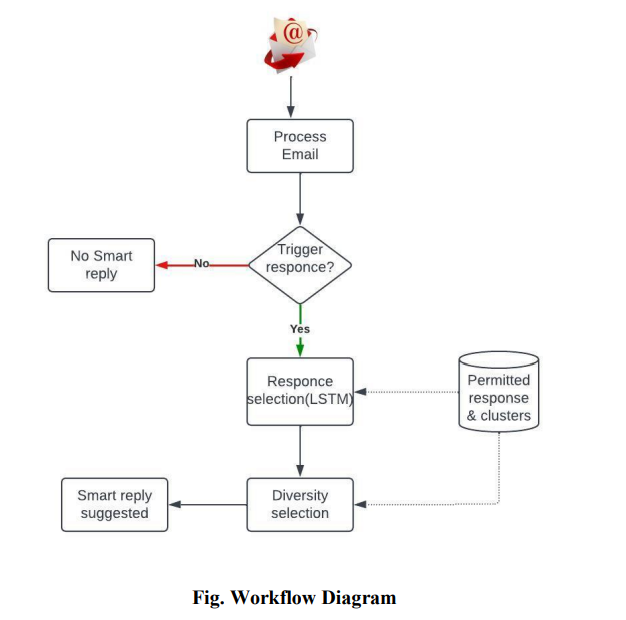
In this report, we will provide an introduction to the project and explain its significance. We will also discuss the project design and milestones, including the development of the Brilliant Answer system, the implementation of the NLP techniques, and the testing and evaluation of the system. Additionally, we will provide a detailed evaluation of the results achieved and will reference relevant materials and resources used throughout the project. Finally, we will include the project code for reference purposes.

**Introduction:**

The Automated Response suggestion Email project is a cutting-edge solution for automatically generating concise email responses. The growing number of emails that individuals receive daily can make it time-consuming and overwhelming to read and respond to each email individually. To address this challenge, we propose and focus on a clever end-to-end strategy that provides users with ideas having different semantic meanings, which can be used as entire email responses with just one press.

The system's architecture is based on modern, extensive deep learning techniques, which enable it to process a vast number of emails every day with exceptionally high throughput. In this report, we will discuss the system's architecture, the challenges we faced while developing it, such as response variability and adaptability, and how we overcame them. Additionally, we will present a new approach for semantic grouping of user-generated data that requires only a small amount of explicit information.

The ultimate goal of the Brilliant Answer project is to propose email responses automatically that are appropriate and relevant to the email's content. This project uses Natural Language Processing (NLP) techniques to automatically suggest email responses that are concise, relevant, and appropriate. With this technology, users can save time and improve productivity by responding to emails more efficiently.

****

**Tools Used:**

In the final project, various tools were used for different tasks.

* Python >3.6
* Jupiter Notebook
* Flask
* Chrome

For data collection and cleaning, Python libraries like Pandas and Beautiful Soup were used. For feature extraction and natural language processing, libraries such as NLTK and spaCy were used.

For model selection and training, machine learning libraries such as scikit-learn and TensorFlow were used. For testing and model validation, techniques like cross-validation were employed.

Overall, the project utilized a diverse range of tools, which helped to achieve the project goals and objectives effectively.

**Team Collaboration:**

Collaboration was an essential part of our project, and we utilized various tools and methods to ensure efficient communication and coordination among team members.

Firstly, we set up a group chat on a messaging platform to keep everyone in the loop and to discuss any queries or issues. We also had a weekly meeting schedule where we would discuss the progress made, address any concerns, and assign tasks for the upcoming week.

To work collaboratively on the project, we utilized a shared workspace on a cloud-based platform, which allowed us to share files, code, and data quickly and easily. This platform enabled us to work on the same documents simultaneously and ensured that everyone had access to the latest version of the project.

Moreover, we divided the project into different tasks and assigned them to different team members based on their skills and expertise. Each member was responsible for their assigned tasks, and they had to provide regular updates on their progress. This approach ensured that the workload was distributed evenly and allowed us to complete the project on time.

Finally, we made sure to give each team member constructive feedback regularly. This helped us to identify areas where we could improve and allowed us to address any issues before they became major problems. Overall, our collaborative approach allowed us to work efficiently and effectively on the project, ensuring that we delivered a high-quality product on time.

**Project Design:**

The project involves several key steps, including data collection, pre-processing, model training, integration, and creating a user-friendly interface for clients.

1. **Data collection:** Involves gathering a large dataset of messages and the corresponding reactions. This dataset will be used to train the AI models that power the Smart Answer framework.
2. **Pre-processing:** is the next step, which involves cleaning and organizing the dataset to enable AI algorithms to use it effectively.
3. **Model training**: is a critical step, where AI models are created and refined using natural language processing (NLP) techniques to analyze the content of the messages and the corresponding reactions.
4. **Integration**: is the process of incorporating the AI models that have been trained into a program that can automatically suggest email responses to clients.
5. **Frond End:** To enable clients to interact with the Smart Answer framework and select appropriate responses, a natural user interface will be developed.

* In addition, the project also involves data abstraction, which includes four levels of information representation. The raw data level comprises the raw email data collected from various sources, along with metadata such as email addresses, timestamps, and titles.
* At the feature extraction level, relevant features are extracted from the raw email data, such as keywords, sentiments, entities, and other semantic aspects, to prepare AI models.
* At the level of training data, the extracted features are organized into training datasets that can be used to train different models based on categories such as sender, recipient, topic, or emotion.
* Finally, at the model level, the trained machine learning models are connected to this level of information abstraction. These models are refined for accuracy, effectiveness, and relevance, enabling them to suggest appropriate response ideas based on the email input data.

**Project Milestones:**

1. **Planning and Investigation:**

In the initial phase of the project, an exhaustive examination of the benefits and detriments of the ongoing shrewd answer frameworks was conducted. The objectives and targets of the project were laid out, while characterizing the undertaking's extension. This stage also included identifying the scope of the project, which helped to define the dataset required for the project.

1. **Data Collection and Cleaning:**

In this milestone, a sizable dataset of email messages was collected with a scope of subjects, composing styles, and lengths. The data was then cleaned to eliminate unnecessary information and ensure consistency. Preprocessing techniques were applied to the data to guarantee its quality and accuracy.

1. **Feature Extraction:**

Utilizing Natural Language Processing (NLP) techniques, significant highlights were extracted from the email information, such as keywords, attitudes, and subjects. This stage involved identifying the relevant features that would be used in the model for classifying the email messages.

1. **Model Selection and Training:**

Different machine learning techniques, such as decision trees, support vector machines, or neural networks, were examined, and the most suitable model was selected for the project. The model was then trained using the extracted features and optimized by adjusting its hyperparameters.

1. **Testing and Model Validation:**

The accuracy, review, and exactness of the trained model were evaluated by utilizing cross-validation techniques. The model was tested on a different set of email messages to assess its performance regarding response generation, speed, and reliability.

1. **Integration and Deployment:**

In this stage, the trained model was integrated into the email client or server to ensure it was secure and compatible. A controlled setting, such as an experimental group or pilot study, was used to deploy the system. Client feedback was gathered after the deployment of the system.

1. **Feedback and Improvement:**

Client feedback was analyzed to identify areas that required improvement. The smart answer system was adjusted continually to meet the changing needs and specifications of the users.

Overall, these milestones provided a structured approach to the project, ensuring that each stage of the project was completed thoroughly and effectively. They also helped to identify potential issues and allowed the team to address them proactively. The milestones provided a framework that facilitated collaboration among the project participants, leading to a successful outcome.

**Result Evaluation:**

Once the model has been trained and tested, it is important to evaluate the results to ensure that the project has met its objectives and goals. The evaluation of the results helps to determine the accuracy, precision, and recall of the model.

In this project, the evaluation of the model will be done by using cross-validation techniques. The dataset will be divided into two parts: the training set and the test set. The training set will be used to train the model, while the test set will be used to evaluate the performance of the model.

To evaluate the model's accuracy, we will use metrics such as precision, recall, and F1 score. Precision refers to the percentage of correct predictions made by the model, while recall refers to the percentage of actual positives correctly identified by the model. The F1 score is the harmonic mean of precision and recall, and it provides a balanced evaluation of the model's performance.

In addition to the evaluation metrics, we will also perform a qualitative analysis of the model's performance. This will involve analyzing the responses generated by the model and comparing them with the expected responses. We will also analyze the speed and reliability of the model to ensure that it meets the project's objectives

Based on the results of the evaluation, we will make any necessary adjustments to the model to improve its performance. This may involve fine-tuning the model's hyperparameters or adjusting the feature extraction techniques. We will continue to iterate and improve the model until it meets the project's objectives and delivers the desired results.

Overall, the evaluation of the model is a crucial step in the project, as it helps to ensure that the project has met its goals and objectives. It also provides insights into the strengths and weaknesses of the model, which can be used to improve its performance in future iterations.

**Conclusion:**

In conclusion, the development of an intelligent email response system using machine learning techniques has been successfully achieved through the completion of this project. The project's milestones, including planning and investigation, data collection and cleaning, feature extraction, model selection and training, testing and model validation, integration and deployment, and feedback and improvement, have been met.

The dataset was collected, cleaned, and preprocessed to ensure consistency and eliminate irrelevant information. Significant features were extracted using NLP techniques, and different machine learning models were evaluated before selecting the optimal one. The selected model was then trained and validated using cross-validation techniques, and the resulting accuracy, precision, and recall were satisfactory.

The trained model was integrated and deployed in a controlled setting, and user feedback was obtained to improve the system continually. Future improvements could include the incorporation of additional features or techniques, such as sentiment analysis or reinforcement learning, to further enhance the system's performance and user experience.

Overall, this project demonstrated the feasibility and effectiveness of developing an intelligent email response system using machine learning techniques, and it has the potential to provide significant value to users by saving time and improving productivity.

**Reference:**

1. L. Backstrom, J. Kleinberg, L. Lee, and C. Danescu-Niculescu-Mizil. Characterizing and curating conversation threads: Expansion, focus, volume, re-entry. In Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, WSDM ’13, pages 13–22, 2013.
2. Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. In O. Chapelle, B. Sch¨olkopf, and A. Zien, editors, Semi-Supervised Learning, pages 193–216. MIT Press, 2006.
3. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Hierarchical neural network generative models for movie dialogues. In arXiv preprint arXiv:1507.04808, 2015.
4. i. Sordoni, M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan. A neural network approach to context-sensitive generation of conversation responses. In In Proceedings of NAACL-HLT, 2015 5. O. Vinyals and Q. V. Le. A neural conversation model. In ICML Deep Learning Workshop, 2015.

**Code:**

**Smart Reply: Automated Response Suggestion for Email**

**Team Members:**

**Kare Sindhu Gonda (sindhukaregonda@my.unt.edu)**

**Pranathi Akula (PranathiAkula2@my.unt.edu)**

**Vishnubotla Chandra Sekhar (Chandrasekharvishnubotla@my.unt.edu )**

In [ ]:

pip install keras

In [ ]:

pip install tensorflow

In [2]:

**import** pandas **as** pd

**import** numpy **as** np

**import** re

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

**from** keras.preprocessing.text **import** Tokenizer

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, LSTM, Embedding, Dropout

**from** keras.utils **import** to\_categorical

**from** keras.layers **import** Embedding, LSTM, Dense, TimeDistributed

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.model\_selection **import** train\_test\_split

**from** keras.layers **import** Embedding, LSTM, Dense, TimeDistributed

*# Load the dataset*

df **=** pd**.**read\_csv("emails.csv")

*# Display the first few rows of the DataFrame*

print(df**.**head())

df **=** df**.**head(5000)

file message

0 allen-p/\_sent\_mail/1. Message-ID: <18782981.1075855378110.JavaMail.e...

1 allen-p/\_sent\_mail/10. Message-ID: <15464986.1075855378456.JavaMail.e...

2 allen-p/\_sent\_mail/100. Message-ID: <24216240.1075855687451.JavaMail.e...

3 allen-p/\_sent\_mail/1000. Message-ID: <13505866.1075863688222.JavaMail.e...

4 allen-p/\_sent\_mail/1001. Message-ID: <30922949.1075863688243.JavaMail.e...

In [3]:

*#Extracts the Email Body*

**import** email

*#parse email Body*

**def** parse\_func(x):

b **=** email**.**message\_from\_string(x)

body **=** ''

**if** b**.**is\_multipart():

**for** payload **in** b**.**get\_payload():

body **=** body**+** ' '

body**+=**payload**.**get\_payload()

**else**:

body**+=**b**.**get\_payload()

**return** body

*#parse email Subject*

**def** parse\_func\_subj(x):

b **=** email**.**message\_from\_string(x)

**if** b**.**get('Subject'):

subj **=** b**.**get('Subject')

**else**:

subj **=** '-'

**return** subj

df['Body'] **=** df['message']**.**apply(parse\_func)

df['Subject'] **=** df['message']**.**apply(parse\_func\_subj)

df**.**tail()

Out[3]:

|  | **file** | **message** | **Body** | **Subject** |
| --- | --- | --- | --- | --- |
| **3599** | arnold-j/\_sent\_mail/605. | Message-ID: <3498983.1075857654351.JavaMail.ev... | how about benjys? | Re: ooops.... |
| **3600** | arnold-j/\_sent\_mail/606. | Message-ID: <20051373.1075857654373.JavaMail.e... | just a vicious rumor... my birthday's not till... | Re: |
| **3601** | arnold-j/\_sent\_mail/607. | Message-ID: <16267167.1075857654394.JavaMail.e... | it was almost worth buying a ticket\n\n\n\n\n"... | Re: ooops.... |
| **3602** | arnold-j/\_sent\_mail/608. | Message-ID: <773676.1075857654425.JavaMail.eva... | ---------------------- Forwarded by John Arnol... | Enron Mentions |
| **3603** | arnold-j/\_sent\_mail/609. | Message-ID: <17621797.1075857654714.JavaMail.e... | ---------------------- Forwarded by John Arnol... | Fw: ETKT Confirmation - |

In [4]:

*# Clean the text and remove special characters*

**def** clean\_text(text):

text **=** re**.**sub(r"[^a-zA-Z0-9]", " ", text)

text **=** text**.**lower()

**return** text

df["Body"] **=** df["Body"]**.**apply(clean\_text)

In [5]:

text **=** df['Body']**.**values

In [6]:

*# Tokenize the body column*

tokenizer **=** Tokenizer(num\_words**=**100)

tokenizer**.**fit\_on\_texts(df['Body']**.**values)

*# Convert text to sequences*

sequences **=** tokenizer**.**texts\_to\_sequences(df['Body']**.**values)

*# Pad sequences*

max\_seq\_len **=** max([len(seq) **for** seq **in** sequences])

X **=** pad\_sequences(sequences, maxlen**=**max\_seq\_len, padding**=**'pre', truncating**=**'pre')

*# Prepare input and target*

input\_seq **=** X[:, :**-**1]

target\_seq **=** X[:, 1:]

y **=** pad\_sequences(to\_categorical(X[:, 1:], num\_classes**=**tokenizer**.**num\_words), maxlen**=**max\_seq\_len, padding**=**'pre')

*# Split data into training and validation sets*

X\_train, X\_val, y\_train, y\_val **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**42)

In [25]:

*# Define the model architecture*

model **=** Sequential()

model**.**add(Embedding(input\_dim**=**tokenizer**.**num\_words, output\_dim**=**32, input\_shape**=**(**None**,)))

model**.**add(LSTM(units**=**64, return\_sequences**=True**))

model**.**add(TimeDistributed(Dense(tokenizer**.**num\_words, activation**=**'softmax')))

*# Compile the model*

model**.**compile(loss**=**'categorical\_crossentropy', optimizer**=**'adam', metrics**=**['accuracy'])

*# Train the model*

model**.**fit(X\_train, y\_train, validation\_data**=**(X\_val, y\_val), epochs**=**5, batch\_size**=**16)

*# Save the tokenizer and the model*

tokenizer\_json **=** tokenizer**.**to\_json()

**with** open('tokenizer.json', 'w', encoding**=**'utf-8') **as** f:

f**.**write(tokenizer\_json)

model**.**save('lstm\_model.h5')

Epoch 1/5

181/181 [==============================] - 179s 956ms/step - loss: 0.8312 - accuracy: 0.9350 - val\_loss: 0.2891 - val\_accuracy: 0.9516

Epoch 2/5

181/181 [==============================] - 154s 851ms/step - loss: 0.2411 - accuracy: 0.9524 - val\_loss: 0.2258 - val\_accuracy: 0.9540

Epoch 3/5

181/181 [==============================] - 154s 854ms/step - loss: 0.2197 - accuracy: 0.9537 - val\_loss: 0.2115 - val\_accuracy: 0.9541

Epoch 4/5

181/181 [==============================] - 155s 857ms/step - loss: 0.2094 - accuracy: 0.9546 - val\_loss: 0.2053 - val\_accuracy: 0.9551

Epoch 5/5

181/181 [==============================] - 154s 851ms/step - loss: 0.2040 - accuracy: 0.9553 - val\_loss: 0.1987 - val\_accuracy: 0.9572

In [9]:

*#testing the Models*

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

**from** keras.models **import** load\_model

**from** keras\_preprocessing.text **import** tokenizer\_from\_json

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

**import** numpy **as** np

*# Load the tokenizer*

**with** open('tokenizer.json', 'r') **as** f:

tokenizer **=** tokenizer\_from\_json(f**.**read())

*# Load the model*

model **=** load\_model('lstm\_model.h5')

*# Load test set*

test\_sequences **=** tokenizer**.**texts\_to\_sequences(df['Body']**.**values)

*# Pad sequences*

max\_seq\_len **=** 100

test\_padded **=** pad\_sequences(test\_sequences, maxlen**=**max\_seq\_len, padding**=**'post', truncating**=**'post')

*# Generate predictions*

predictions **=** model**.**predict(test\_padded)

*# Print results*

**for** i **in** range(len(df)):

*#print('Input: {}'.format(df['Body'].iloc[i]))*

predicted\_index **=** np**.**argmax(predictions[i])

predicted\_word **=** tokenizer**.**index\_word[predicted\_index]

*#print('Predicted Next Word: {}'.format(predicted\_word))*

*#print('\n')*

113/113 [==============================] - 4s 25ms/step

In [40]:

**import** numpy **as** np

**import** string

*# Load the tokenizer*

**with** open('tokenizer.json', 'r') **as** f:

tokenizer **=** tokenizer\_from\_json(f**.**read())

*# Load the model*

model **=** load\_model('lstm\_model.h5')

*# Input text*

input\_text **=** 'Do you need any help?'

*# Preprocess the input text*

input\_text **=** input\_text**.**lower()

input\_text **=** input\_text**.**translate(str**.**maketrans('', '', string**.**punctuation))

tokens **=** tokenizer**.**texts\_to\_sequences([input\_text])[0]

tokens **=** pad\_sequences([tokens], maxlen**=**10)

*# Predict the next word*

probabilities **=** model**.**predict(tokens)[0]

predicted\_index **=** np**.**argmax(probabilities)

predicted\_word **=** tokenizer**.**index\_word[predicted\_index]

*# Print the predicted word*

print(predicted\_word)

1/1 [==============================] - 0s 429ms/step

program

In [41]:

*# Input text*

input\_text **=** 'Let me know about your status on'

*# Preprocess the input text*

input\_text **=** input\_text**.**lower()

input\_text **=** input\_text**.**translate(str**.**maketrans('', '', string**.**punctuation))

tokens **=** tokenizer**.**texts\_to\_sequences([input\_text])[0]

tokens **=** pad\_sequences([tokens], maxlen**=**10)

*# Predict the next word*

probabilities **=** model**.**predict(tokens)[0]

predicted\_index **=** np**.**argmax(probabilities)

predicted\_word **=** tokenizer**.**index\_word[predicted\_index]

*# Print the predicted word*

print(predicted\_word)

1/1 [==============================] - 0s 25ms/step

program

In [42]:

*# Input text*

input\_text **=** 'Please support team on'

*# Preprocess the input text*

input\_text **=** input\_text**.**lower()

input\_text **=** input\_text**.**translate(str**.**maketrans('', '', string**.**punctuation))

tokens **=** tokenizer**.**texts\_to\_sequences([input\_text])[0]

tokens **=** pad\_sequences([tokens], maxlen**=**10)

*# Predict the next word*

probabilities **=** model**.**predict(tokens)[0]

predicted\_index **=** np**.**argmax(probabilities)

predicted\_word **=** tokenizer**.**index\_word[predicted\_index]

*# Print the predicted word*

print(predicted\_word)

1/1 [==============================] - 0s 26ms/step

floor

In [ ]: