**Documentation of the GEN AI Orchestrator for Email and Document Triage/Routing**

**1. Create Email Files with Required Attachments for Each Category**

The first step in training a BERT model for email classification is **data collection**. Realistic emails are generated for different categories, each containing structured content and attachments relevant to their classification.

📌 **Example Categories:**

* **Fee Payment** – Emails related to payment transactions and invoices.
* **Commitment Charge** – Notifications regarding charges applied to accounts.
* **Closing Notice** – Notifications about account closure with associated fees.

**All the categories which are supporting are below**:

✅ **AU Transfer**  
✅ **Adjustment**  
✅ **Closing Notice**  
✅ **Commitment Change**  
✅ **Money Movement (Inbound & Outbound)**  
✅ **Fee Payment**

🔹 **Attachments:**  
Each email should include **relevant attachments** (e.g., PDFs, Excel files, invoices, or receipts) to mimic real-world data and improve classification accuracy.

**2. Save Them in a Folder Called generated\_emails**

Once emails are generated, they need to be stored systematically. All emails are saved in a **dedicated folder** (generated\_emails) in **.eml** or **.msg** format.

📂 **Folder Structure:**

generated\_emails/

│── fee\_payment\_01.eml

│── commitment\_charge\_02.eml

│── closing\_notice\_03.eml

This structured storage ensures easy access when training and parsing emails.

**3. Train Models Using BERT Pre-Trained Data Model**

BERT (Bidirectional Encoder Representations from Transformers) is used to **understand and classify emails**. Instead of training from scratch, we use **pre-trained BERT models** (such as bert-base-uncased) and fine-tune them for email classification.

⚙️ **Steps in Model Training:**

* Convert emails into a machine-readable format.
* Tokenize text using **WordPiece Tokenization**.
* Input tokens into **BERT’s transformer layers**.
* Train a classification head to predict categories.

**4. Use BERT Data Model to Process Emails for Classification**

After training, the BERT model is used to **process incoming emails** and predict their categories. The emails are:

1. **Tokenized** → Converted into input IDs for BERT.
2. **Processed** → BERT analyzes the context using self-attention.
3. **Classified** → The output layer assigns a category label to each email.

**5. Save Parsed Emails into a CSV File for Training**

To ensure structured data, the parsed email contents are stored in a **CSV file**.

📄 **Example parsed\_emails.csv:**

| **Email ID** | **Subject** | **Content** | **Category** |
| --- | --- | --- | --- |
| 001 | Payment Confirmation | Your invoice has been paid | Fee Payment |
| 002 | Interest Charges | New charges applied to your account | Commitment Charge |

This CSV file serves as a **training dataset** for model fine-tuning.

**6. Load Parsed Email Data from parsed\_emails.csv**

Once the CSV file is generated, it is loaded back into the training pipeline. The data is read using **Pandas** or other data-processing libraries to extract email content and labels.

python

import pandas as pd

data = pd.read\_csv('parsed\_emails.csv')

print(data.head())

This step ensures that emails are correctly formatted before NLP processing.

**7. Clean Email Text Using NLP Techniques**

Email text often contains **noise** like special characters, HTML tags, and stopwords. Using **Natural Language Processing (NLP)** techniques, we clean the data to improve model performance.

✔ **NLP Cleaning Steps:**  
✅ Remove HTML tags & special characters  
✅ Convert text to lowercase  
✅ Remove stopwords (e.g., "the," "is," "and")  
✅ Perform stemming & lemmatization

python

from nltk.corpus import stopwords

import re

def clean\_text(text):

text = re.sub(r'<.\*?>', '', text) # Remove HTML tags

text = text.lower() # Convert to lowercase

text = ' '.join(word for word in text.split() if word not in stopwords.words('english'))

return text

**8. Save the Processed Dataset for BERT Training**

After cleaning, the dataset is saved as a **preprocessed CSV file** (processed\_emails.csv) for BERT training.

python

data['cleaned\_text'] = data['Content'].apply(clean\_text)

data.to\_csv('processed\_emails.csv', index=False)

This processed dataset ensures that **BERT receives optimized input** for fine-tuning.

**9. Fine-Tune BERT Using Hugging Face’s Transformers**

Fine-tuning involves adjusting the **pre-trained BERT model** to classify emails into specific categories.

✔ **Steps for Fine-Tuning:**

* Load the bert-base-uncased model.
* Replace the final classification layer with a custom **softmax layer**.
* Train the model using **supervised learning** with processed\_emails.csv.

python

from transformers import BertTokenizer, BertForSequenceClassification

model = BertForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=5)

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

The model is then trained using an **Adam optimizer** and **cross-entropy loss function**.

**10. Load the Fine-Tuned BERT Model**

After training, the fine-tuned BERT model is loaded for **email classification**.

python

from transformers import pipeline

classifier = pipeline("text-classification", model="fine\_tuned\_bert")

This classifier is now ready to predict **email categories** in real-time.

**11. Tokenize New Email Text**

New incoming emails are tokenized before classification.

python

new\_email = "Your payment of $500 has been processed successfully."

tokens = tokenizer(new\_email, return\_tensors="pt")

Tokenized text ensures that **BERT can interpret and classify** new emails correctly.

**12. Use the Trained BERT to Predict the Category**

The final step is using the **fine-tuned BERT model** to classify emails based on their content.

python

category = classifier(new\_email)

print(category)

📌 **Example Output:**

bash

{'label': 'Fee Payment', 'score': 0.98}

The email is correctly classified as a **Fee Payment** with **98% confidence**.