B”H

**Email Classification GenAI Solution**

****

Below is a concise overview addressing the first two deliverables. Further on this document, you will find a comprehensive, in-depth explanation of the selected solution and the methods employed to fulfill the task.

Deliverable 1: Output of the Required Development

What the Output Looks like:

1. Trained Model(s):

* A multilingual model (English and Hebrew, for example bert-base-multilingual-cased or xlm-roberta-base) fine-tuned on the company’s email dataset.
* Model artifacts include:
  + - 1. pytorch\_model.bin: model’s weights file.
      2. config.json
      3. tokeniker file (for example vocab.txt)

1. Web-based (or CLI-based) Classifier Application:

* A local Flask (or Streamlit) application that can:

1. Accept Subject and Body of an email (in English or Hebrew).
2. Output the Category (for example: HR, Finance, IT Support).
3. Output the Priority level (High, Medium, Low)
4. Optionally allow feedback from the user to confirm or correct the classification.
5. (Optional) RAG / Agentic AI Extension:

* If LangChain /LangGraph or any other retrieval mechanism have been integrated, the user would have:

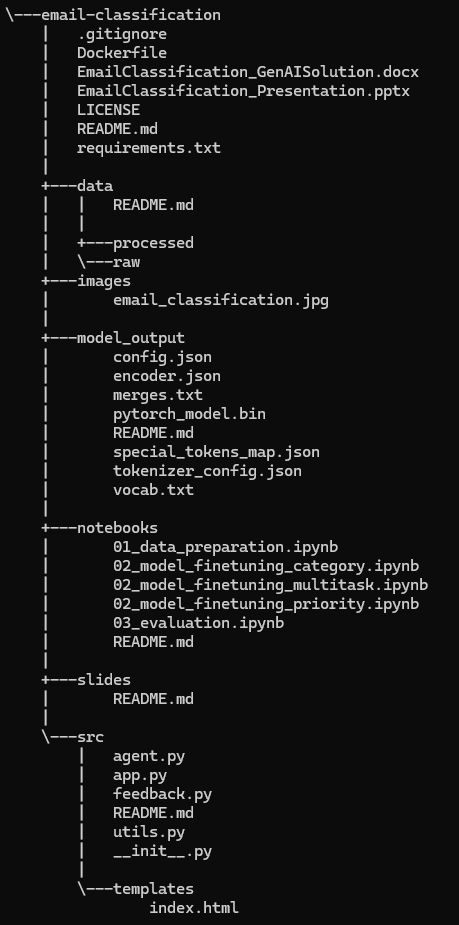
1. A basic retrieval system (using a Vector DB, for example FAISS or Pinecone) loaded with relevant domain docs.
2. A script or function (an agent) in the code to handle RAG-based lookups.
3. Evaluation Report:

* Performance metrics (accuracy, precision, recall, F1) on a held-out test set.

In short, the main deliverable is a working pipeline + interface that shows how an email’s subject and body get automatically classified and prioritized with optional user feedback.

Deliverable 2: Code for the Development, Organized and Versioned by Stages

Below is the repository structure, you can find the repository at <https://www.github.com/ShomoVolosky/email-classification>:



Stage 1: Data Collection and Preparation:

Steps:

1. Load raw email CSV/JSON datasets into a Pandas DataFrame,
2. Clean the text (remove HTML, URLs, special characters),
3. (Optional) Identify language (if for example both Hebrew and English are in the same dataset and the user wants to label them),
4. Split into train/test sets in a balanced manner.

Stage 2: Model Training and Fine-Tunning:

Steps:

1. Install and import Hugging Face transformers and datasets,
2. Load the multilingual model and tokenizer,
3. Prepare the dataset for tokenization and label alignment,
4. Fine-tune for category classification; optionally multi-task for priority or train a separate model.
5. Save the trained model artifacts in model\_output/.

Stage 3: Evaluation:

Steps:

1. Evaluate on the test set.
2. Display accuracy, precision, recall, F1.
3. (Optional) Show a confusion matrix.

Stage 4: UI/Chatbot and Feedback Loop:

Steps:

1. Create src/app.py:

* A Flask/Streamlit app that loads the saved model and provides a web form for email text.
* Returns the predicted category and/or priority.
* Incorporates a feedback function (thumbs up/down or correct labels).

1. Create src/feedback.py:

* This is meant to handle storing user feedback in a local SQLite DB or a cloud-based database.

1. (Optional) Create src/agent.py:

* If implementing RAG or agentic AI via LangChain or LangGraph.