PART A — Email Tagging System

1.Problem Statement:

- The goal of Part A is to design an email-tagging system that automatically assigns issue tags to incoming customer emails. The system must
  - Work across multiple customers
  - Avoid cross-customer tag leakage
  - Perform well despite limited training examples
  - Use a combination of ML, rules, and guardrails
  - Provide a reliable and explainable tagging workflow

---

2. Dataset Overview:

- Two datasets were provided:
  - small_dataset.csv (very small, only a few samples per tag)
  - large_dataset.csv (more tags but still very sparse)

- Dataset characteristics:
  - ~60 unique tags
  - Most tags appear only once
  - Per-customer data is extremely small (1–3 samples)
  - Text content is extremely short and sparse

This makes training a pure machine-learning model very difficult, especially for multi-class classification.

---

3. Baseline Model (TF-IDF + Logistic Regression):

- A baseline multi-class classifier was trained using:
  - TF-IDF features
  - Logistic Regression (max_iter=2000)
  - Train/Test split or full-dataset training depending on size
- Due to extreme sparsity of labels and low sample count:

- Model predictions collapse to a few classes
- F1, precision, and recall across most classes are 0
- Confusion matrix shows most predictions along one column
- Model cannot generalize

Key Reason for Poor Performance

- Most classes have only 1 sample
- No meaningful patterns can be learned
- Vocabulary overlap is extremely low

This is expected and an important part of the assignment.

---

4. Error Analysis (Why the Model Fails)

Observations

1. Many tags have only one example → impossible to learn boundaries
2. Emails are extremely short → limited feature extraction
3. Multi-class (60 labels) with very few samples → underfitting
4. Overlap between tag semantics is high
5. Per-customer data is even smaller

Conclusion

Traditional ML cannot solve this dataset alone.
This motivates customer isolation and rule-based guardrails.

---

5. Customer Isolation

To prevent cross-customer tag leakage, a mapping of allowed tags per customer was created:

```
customer_allowed_tags = {
  "CUST_A": ["workflow_issue", "notification_bug", ...],
  "CUST_B": ["sync_bug", "analytics_issue", ...]
}
```

When predicting tags:

- If a customer-specific model exists → use it
- If not → use the global model but **filter predictions** only to that customer's allowed tags

Impact

- Reduces prediction space from ~60 tags to 3–5 tags per customer
- Increases accuracy
- Prevents invalid tag assignment
- Automatically improves reliability

---

6. Customer-Specific Models

Per-customer models were trained **only if a customer had ≥5 samples**.

Benefits:

- Much smaller number of tags
- Local patterns per customer can be learned
- Influences the final prediction pipeline

Even if accuracy remains low, customer-specific models still narrow down predictions.

---

7. Guardrails (Pattern + Anti-Pattern Rules)

Because ML alone is insufficient, guardrails were added to correct/override predictions.

Sample Guardrail Rules

Billing Issues

if text contains: charge, invoice, billed, refund

→ "billing_error"

Workflow / Rules / Automation

if text contains: rule, workflow, sla, automation

→ "workflow_issue"

Notifications

if text contains: notification, alert

→ "notification_bug"

Mobile Issues

if text contains: mobile

→ "mobile_bug"

Tagging Issues

if text contains: tag, tagging

→ "tagging_issue"

Guardrails fix many incorrect model predictions.

---

8. Final Prediction Pipeline

The final pipeline combines:

1️⃣ Customer-specific model

If available → used directly.

2️⃣ Global model with tag filtering

Only tags allowed for that customer are considered.

3️⃣ Guardrail corrections

Overrides ML predictions using rule patterns.

4️⃣ Return structured output

{

predicted: <model prediction>,

corrected: <guardrail prediction>,

confidence: <score>,

source: "customer_model" | "global_filtered"

}

---

9. Example Output

Input:

"We are unable to configure auto assignment rules."

ML predicted:

auth_issue

Guardrails corrected it to:

workflow_issue

Final output:

{

'predicted': 'auth_issue',

'corrected': 'workflow_issue',

'confidence': 1.0,

'source': 'customer_model'

}

This demonstrates how ML + rules improve reliability.

---

10. Confusion Matrix Interpretation

The confusion matrix shows:

- Most cells = zero
- A few repeated predictions
- Model collapses to a few tags
- Expected due to sparse training data
- Motivates rule-based corrections and customer isolation

---

11. System Architecture Diagram

(Describe in Word, or I can generate diagram text)

Flow:

Email → Preprocessing → Customer Model →

→ (if unavailable → Global Model + Tag Filtering) →

→ Guardrails → Final Tag Output

---

12. Production Improvements

1. **Use embeddings (BERT/SentenceTransformers) instead of TF-IDF**
   Better semantic understanding.
2. **Add contextual RAG-based retrieval**
   Fetch similar past tickets and use similarity to improve prediction.
3. **Human-in-the-loop feedback loop**
   Collect agent corrections → retrain periodically.
4. **Hybrid ML + Rule Engine**
   Rules for high-precision domains, ML for general cases.

---

13. Final Deliverables (for Word/PDF)

Include:

- Baseline model explanation
- Dataset summary
- Error analysis
- Customer isolation logic
- Guardrail rules
- Final prediction architecture
- Confusion matrix screenshot
- Sample output
- Production improvements
- Conclusion