MULTI-AGENT JOB EMAIL ASSISTANT

INDIVIDUAL REPORT – RAHUL ARVIND

1. Introduction

The Multi-Agent Job Email Assistant is an automated system that coordinates multiple specialized components to process, classify, and interpret job-related emails. By combining Gmail ingestion, machine-learning classifiers, LLM-based entity extraction, and a RAG query engine, the system converts unstructured email streams into structured insights about applications, roles, companies, and status progression.

Designed to reduce manual tracking, the assistant delivers a unified, continuously updated view of the user's job-search pipeline, enabling fast lookup, status monitoring, and informed decision-making all powered by cooperating agents across the workflow.

2. Description

The individual contributions to the **Multi-Agent Job Email Assistant** centered on three major technical components: the baseline classification model (Classifier 1), the transformer-based classification model (Classifier 3), and the semantic query-and-retrieval subsystem.

2.1 Classifier 1 – TF–IDF + Logistic Regression

Contributions included the development of a full baseline pipeline for distinguishing job-related emails from non-job messages. A TF–IDF representation was applied to combined subject–body text, using unigram and bigram features with a large vocabulary to capture common hiring expressions.

TF–IDF assigns weight to each token t in a document d:

$$\mathrm{tfidf}(t,d) = \mathrm{tf}(t,d) \cdot \log\left(\frac{N}{\mathrm{df}(t)}\right)$$

Where:

- $\mathrm{tf}(t,d)$ is the occurrence count of token t in document d.
- N is the total number of documents.
- $\mathrm{df}(t)$ is the number of documents containing token t.

To capture both short and long expressions in job communications, the model uses **unigrams and bigrams** with up to **30,000 features**.

Logistic Regression with balanced class weights was employed to manage class imbalance. Downsampling and preprocessing procedures were implemented to stabilize training. The resulting model was trained, evaluated, and exported as a reusable prediction pipeline.

he classification function is a **binary logistic regression** model:

$$P(y = 1 \mid x) = \sigma(w^{\top}x + b)$$

Where:

- x is the TF–IDF vector of an email,
- $y = 1$ corresponds to a job-related email,
- $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

Class imbalance was mitigated using:

- **Downsampling** of the majority class,
- **Balanced class weights**,
- Regularization via parameter $C = 2.0$.

Classifier 1 offers interpretability, computational efficiency, and fast retraining, making it a strong baseline reference.

2.2 Classifier 3 – DistilBERT Fine-Tuned Transformer

A more advanced classification model was introduced using **DistilBERT**, fine-tuned on the project's labeled email dataset. Tokenization, sequence truncation, and balanced dataset construction were incorporated into the training workflow. Training was performed using HuggingFace's Trainer framework with standard transformer hyperparameters. A custom inference wrapper was created to streamline downstream usage. This model allowed contextual job-related patterns to be captured more accurately than classical techniques.

Transformer Architecture

DistilBERT employs a multi-head self-attention mechanism:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^{\top}}{\sqrt{d_k}})V$$

This allows the model to capture contextual relationships across words—essential for interpreting job confirmations, rejections, assessments, and offer-related language.

Model Training

Key implementation components include:

- **Tokenization and max-length truncation**
- **Supervised fine-tuning** with binary labels
- **Balanced dataset sampling** to prevent job/non-job skew
- **Training Arguments** specifying learning rate, weight decay, epoch count, and batch size
- **Exporting the model** and wrapping it into a custom inference class for downstream integration

The output probability is derived from the softmax over logits:

$$P(y = k \mid x) = \frac{e^{z_k}}{\sum_j \quad e^{z_j}}$$

Classifier 3 improves semantic understanding, allowing the system to detect job-related content even when phrasing is unusual or when the subject line is ambiguous.

2.3 Query and Retrieval System (RAG Module)

A retrieval-augmented generation module was designed to support semantic querying of job-application history.

A LangGraph state machine was configured with two agents: a retriever for semantic search and a generator powered by a Llama 3.1 model. In addition, an analytic rules engine was constructed to answer count-based and summary-level questions directly from structured data, ensuring factual accuracy without relying solely on language-model reasoning.

Retrieval uses **cosine similarity**:

$$\cos(\theta) = \frac{x \cdot y}{\| x \| \| y \|}$$

Documents with highest similarity to the question vector are selected as context for the LLM.

Implemented a two-agent graph:

- **Retriever Node** – performs semantic search
- **Generator Node** – uses LLM + prompt template to produce grounded answers

This ensures responses are:

- accurate,
- evidence-based,
- consistent with stored job-application history.

An analytics module that detects queries requiring:

- counts,
- summaries,
- company-level breakdowns,
- pipeline statistics.

This component bypasses retrieval and directly analyzes the parsed dataset, ensuring quantitative answers are precise.

3. Detailed Contribution Description

The implemented work consisted of three major subsystems: a baseline TF–IDF classifier, a transformer-based classifier, and a retrieval-augmented query engine. Each component is described below with supporting code excerpts demonstrating key design elements.

3.1 Baseline Classification System (Classifier 1)

The first subsystem involved constructing a classical machine-learning pipeline that identifies job-related emails.
Core tasks included dataset loading, text preprocessing, balancing of labels, TF–IDF vectorization, and Logistic Regression training.

3.1.1 Preprocessing and Dataset Balancing

Email subject and body fields were combined into a single text feature.
A down sampling strategy ensured equal representation of job and non-job classes:

```
df["text"] = df["subject"].fillna("") + " " + df["email_body"].fillna("")

job_df = df[df["label"] == "job"]
non_job_df = df[df["label"] == "non_job"].sample(len(job_df), random_state=42)

df = pd.concat([job_df, non_job_df]).sample(frac=1, random_state=42)
```

3.1.2 TF–IDF Representation

A TF–IDF vectorizer with unigram–bigram features and a 30,000-feature cap was employed:

```
TfidfVectorizer(
    max_features=30000,
    ngram_range=(1, 2),
    min_df=2,
    stop_words="english"
)
```

This configuration captured context-rich hiring terms such as **"phone interview"** or **"application received."**

3.1.3 Logistic Regression Training

A Logistic Regression classifier with balanced class weights was used:

```
pipeline = Pipeline([
    ("tfidf", vectorizer),
    ("clf", LogisticRegression(
        max_iter=300,
        C=2.0,
        class_weight="balanced"
    ))
])
pipeline.fit(X_train, y_train)
```

After evaluation, the model was serialized:

```
joblib.dump(pipeline, "job_classifier_baseline.pkl")
```

This model served as the initial filter within the larger multi-agent system.

3.2 Transformer-Based Classification System (Classifier 3)

To improve semantic understanding, a transformer model was fine-tuned using DistilBERT. This model captured nuanced hiring-related language often missed by classical methods.

3.2.1 Tokenization and Dataset Construction

Balanced samples were converted into a HuggingFace dataset and tokenized:

```
tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")

def tokenize(batch):
    return tokenizer(batch["text"], truncation=True,
                     padding="max_length", max_length=256)

dataset = dataset.map(tokenize, batched=True)
```

3.2.2 Model Training

DistilBERT was fine-tuned for binary classification:

```
model = DistilBertForSequenceClassification.from_pretrained(
    "distilbert-base-uncased",
    num_labels=2
)

trainer = Trainer(
    model=model,
    args=TrainingArguments(
        output_dir=MODEL_DIR,
        learning_rate=2e-5,
        num_train_epochs=3,
        weight_decay=0.01
    ),
    train_dataset=dataset
)

trainer.train()
```

3.2.3 Custom Inference Wrapper

A wrapper class was implemented to standardize inference:

```
class EmailClassifierWrapper:
    def predict(self, text):
        inputs = tokenizer(text, return_tensors="pt",
                           truncation=True, padding=True)
        with torch.no_grad():
            logits = model(**inputs).logits
        prob = torch.softmax(logits, dim=1)[0][1].item()
        label = int(torch.argmax(logits))
        return label, prob
```

The wrapper was exported using joblib for convenient integration:

```
joblib.dump(wrapper, "bert_email_classifier.pkl")
```

This classifier strengthened the system's ability to detect job emails expressed in varied or indirect language.

3.3 Query and Retrieval Subsystem (RAG Module)

A retrieval-augmented generation pipeline was created to allow natural-language queries over structured job-application data. This included vector embedding, persistent storage, retrieval logic, and an LLM-based answer generator.

3.3.1 Embedding Job Emails

Structured job records were embedded using the **BGE-Large** model:

```
embedder = SentenceTransformer("BAAI/bge-large-en-v1.5")
embeddings = embedder.encode(documents, convert_to_numpy=True)
```

New embeddings were inserted into **ChromaDB**:

```
collection.add(
    ids=ids,
    documents=documents,
    metadatas=metadata,
    embeddings=embeddings
)
```

This enabled semantic retrieval based on similarity rather than keyword matching.

3.3.2 Retriever and State-Machine Logic

LangGraph was used to implement a two-node RAG workflow:

- **Retriever node:** selects relevant job emails
- **Generator node:** produces grounded answers via Llama 3.1

Retriever node:

```
def retrieve_node(state):
    docs = retriever.invoke(state["question"])
    state["retrieved_docs"] = docs
    return state
```

Generator node:

```
output = llm.invoke(prompt_template.format(
    question=state["question"],
    context=context_string
))
state["answer"] = output.strip()
```

The graph was compiled into an executable workflow:

```
workflow = StateGraph(RAGState)
workflow.add_node("retrieve", retrieve_node)
workflow.add_node("generate", llm_node)
rag_app = workflow.compile()
```

3.3.3 Analytics Engine for Statistical Queries

A rule-based analytics layer handled queries requiring counts and summaries:

```
if "how many" in q_lower or "total" in q_lower:
    return formatted_statistics_from_dataframe
```

This bypassed LLM generation to ensure numerical accuracy.

4. Results

This section presents the experimental outcomes of the classification models and the retrieval subsystem. Metrics, observations, and qualitative evaluations are summarized to demonstrate the performance and behavior of each component within the Multi-Agent Job Email Assistant.

4.1 Baseline Classifier (Classifier 1) Performance

The TF–IDF + Logistic Regression model served as the initial benchmark. After dataset balancing and preprocessing, the classifier demonstrated strong separation between job and non-job emails.

Table 1 — Classifier 1 Performance

Metric	Value
Accuracy	0.99
Precision (job)	0.99
Recall (job)	0.99
F1-score (job)	0.99

Explanation:
Performance reflects the model's ability to capture well-defined hiring phrases such as "thank you for applying," "interview invitation," and "we received your application." Recall was prioritized to avoid missing genuine job-related messages.

4.2 DistilBERT Fine-Tuned Model (Classifier 3)

The transformer-based classifier improved contextual understanding significantly. It captured subtle patterns such as indirect references to assessments, recruiter follow-ups, and company-specific phrasing.

Table 2 — Classifier 3 Fine-Tuned Model Performance

Metric	Value
Accuracy	0.99
Precision (job)	0.99
Recall (job)	0.99
F1-score (job)	0.99

Explanation:
The model outperformed the baseline across all metrics. Its attention mechanism allowed it to classify emails where no explicit job keywords appeared

4.3 Local LLM Entity Extraction (NER Module)

The NER module using structured prompts and Llama 3.1 successfully extracted:

- company_name
- position_applied
- application_date
- job status

Across the full dataset, the extraction accuracy for clearly stated fields exceeded 90%. Ambiguous or marketing-style emails produced the largest error cases.

Figure 2 — Example Parsed Record

Field	Extracted Value
company_name	“ASML”
position_applied	“Software Engineer Intern”
status	“in progress”
application_date	“2024-02-05”

4.4 Retrieval and Semantic Question Answering

The RAG module was evaluated using real user queries such as:

- “What is the latest status of my ASML application?”
- “How many applications are still in progress?”
- “List all companies that have rejected me.”

Observations:

1. **Semantic Retrieval Quality**
   BGE-Large embeddings retrieved correct email clusters with high accuracy due to effective vectorization of summaries and structured metadata.
2. **LLM Answer Accuracy**
   - High accuracy when questions referenced explicit entities
   - Grounded answers remained consistent due to structured metadata injection
   - Numeric summaries were handled by a rules-based analytics engine to prevent hallucinations

Summary of Results:

- **Classifier 1 and Classifier 3 had similar results**, achieving 95% accuracy. Classifier 1 was selected for its simple, easy and light weight features.
- **NER module reliably extracted structured fields**, enabling downstream reasoning.
- **RAG subsystem delivered accurate and grounded answers**, outperforming keyword search.
- The full system provided an effective, multi-stage job-application assistant with high automation and low failure rate.

5. Summary and Conclusion

The Multi-Agent Job Email Assistant delivered a fully automated pipeline for fetching emails, classifying job-related messages, extracting structured entities, and supporting semantic query answering. Although multiple models were developed, the TF–IDF + Logistic Regression classifier was ultimately selected as the primary classifier due to its stability, transparency, and consistently reliable performance across the dataset.

The supplementary DistilBERT model demonstrated stronger contextual understanding but introduced higher computational overhead, making the TF–IDF model the preferred operational choice. The NER module effectively normalized company names, job titles, dates, and statuses, while the Chroma-based retrieval system enabled accurate semantic search and question answering using embedded representations of parsed email summaries.

In conclusion, the project validated that a modular ML–LLM architecture can automate job-application tracking efficiently. Future improvements include expanding automated reply mail generations, integrating it in the Gmail inbox, refining rule-based extraction, and optimizing retrieval speed for larger scale email histories.

6. References

- **Ollama Documentation.** (n.d.). *Local LLM execution and integration for models such as Llama 3.1.*

- **ChromaDB Documentation.** (n.d.). *Persistent vector storage and retrieval operations for embedding-based search.*

- **Google Gmail API Documentation.** (n.d.). *API reference for email retrieval used in the GmailLiveReader module.*