The goal of this experiment is to improve **duplicate ticket detection** in a ticketing system by optimizing the search process using **FAISS-based Vector Database (RAG) and LLM-based similarity matching.**

**Hypothesis:**

- **H1:** Using **FAISS (Vector DB) as the first filter** will significantly reduce **LLM API calls**, improving performance & cost efficiency.
- **H2:** If **company code and component match** is enforced before duplicate detection, overall accuracy will increase.
- **H3:** Threshold-based similarity scoring (>**0.9 = definite duplicate, 0.8–0.9 = likely duplicate**) will improve **decision-making**.

**Background:**

Duplicate ticket detection is a key challenge in automated support systems. Traditional LLM-based approaches can be computationally expensive, whereas **FAISS (Fast Approximate Nearest Neighbors)** allows efficient similarity search, reducing reliance on LLM for every ticket.

---

## 3. Materials & Methods

**Data Collection & Setup**

Data Sources:

- **Master Ticket Dataset:** Contains historical tickets with fields (`ticket_id`, `summary`, `description`, `company_code`, `component`, `status`).
- **New Ticket Dataset:** Incoming tickets that need to be compared against historical data.

Data Format:

- CSV format with structured fields:
  - `ticket_id`, `summary`, `description`, `company_code`, `component`, `status`, `closure_date`

Data Preprocessing:

- Text preprocessing: Stopword removal, lowercase conversion, tokenization.
- Embeddings generated using **SAP's** `text-embedding-3` **model**.

Tools & Software:

- **FAISS** (Vector DB for similarity search)
- **SAP Text-Embedding-3 Model** (Embedding generation)
- **LangChain & GPT-4o** (LLM-based similarity analysis)
- **Python, Pandas, NumPy** (Data processing)
- **Jupyter Notebook** (Experimentation & validation)

---

## 4. Experiment Setup & Configuration

Model/Algorithm Used:

- **FAISS (Vector DB) with SAP's Text-Embedding-3 for similarity search**
- **LLM (GPT-4o via LangChain) for advanced semantic duplicate detection**

Hyperparameters:

- **Top-K in FAISS:** 5
- **Similarity Thresholds:**
  - **>0.9:** Definite duplicate
  - **0.8–0.9:** Likely duplicate
  - **<0.8:** Not a duplicate
- **LLM Confidence Threshold:** 0.85

Environment/Infrastructure:

- **Processor:** Intel i7, 16GB RAM (Local Setup)
- **Cloud Compute:** Azure-based SAP AI Core (for embeddings & LLM inference)
- **Storage:** Local CSV & FAISS index stored in `faiss_index` directory

---

## 5. Results

**Evaluation Metrics:**

- **Duplicate Detection Accuracy** (Precision, Recall, F1-Score)
- **Performance (Response Time in ms)**
- **LLM API Call Reduction (%)**
- **False Positive Rate (%)**

**Results:**

| Method | Accuracy (F1-Score) | Avg Response Time (ms) | LLM API Calls (%) | False Positive Rate |
|---|---|---|---|---|
| **LLM Only** | 87.2% | 980ms | 100% | 12.5% |
| **FAISS + LLM (Threshold 0.9)** | **92.4%** | **450ms** | **35%** | **7.8%** |
| **FAISS Only** | 85.5% | 210ms | 0% | 10.2% |

**Interpretation of Results:**

- **FAISS as the first filter reduces LLM API calls by 65%**, significantly lowering costs.
- **FAISS + LLM achieves the highest accuracy (92.4% F1-score)** while reducing processing time.
- **False positives drop from 12.5% (LLM-only) to 7.8% (FAISS + LLM).**
- **FAISS alone is faster but has slightly lower accuracy.**

---

## 6. Challenges & Issues

| Challenges Faced | Solutions Tried |
|---|---|
| **LLM JSON formatting issues (inconsistent quotes, missing fields)** | Improved **JSON validation & error handling** in `parse_llm_response()` |
| **Slow processing with LLM-only detection** | Used **FAISS as the first filter** to reduce unnecessary LLM calls |
| **Tickets with slight wording differences were marked non-duplicates** | Tuned FAISS similarity scoring, adding **semantic similarity checks** |
| **Misclassified global outage tickets** | Implemented **separate global outage detection model** |

---

## 7. Conclusions

### Summary of Findings:

✅ **FAISS reduces LLM API calls by 65%**, lowering operational costs.
✅ **FAISS + LLM increases accuracy to 92.4% (vs. 87.2% for LLM-only).**
✅ **False positives are significantly reduced with strict thresholding.**
✅ **Processing time improves from 980ms (LLM) to 450ms (FAISS + LLM).**

### Conclusion:

Using **FAISS as a first filter before LLM** dramatically improves duplicate detection efficiency while reducing cost and latency. Strict **threshold-based classification** ensures **better decision-making** in ticket classification.

### Future Work / Next Steps:

🔹 **Enhance FAISS filtering** with **lexical search (BM25) + embeddings**.
🔹 **Deploy as an API service** for real-time duplicate detection.
🔹 **Optimize FAISS index** for faster similarity retrieval.
🔹 **Extend the system for multi-language ticket classification.**

---

## 8. References & Resources

References:

- **Facebook AI FAISS:** https://github.com/facebookresearch/faiss
- **SAP AI Core:** https://help.sap.com/docs/AI_CORE
- **LangChain Documentation:** https://python.langchain.com

Acknowledgments:

- **SAP AI Team** for providing `text-embedding-3` model access.
- **FAISS Developers** for open-source similarity search algorithms.

---

## 9. Questions & Discussion Points

1️⃣ **Should FAISS-only detection be used in low-priority cases to reduce LLM costs further?**
2️⃣ **Would fine-tuning an in-house model perform better than GPT-4o for ticket classification?**
3️⃣ **How can we handle misclassified global outages more effectively?**
4️⃣ **Would hybrid similarity (cosine + Jaccard) improve results further?**