**Deployment of a Domain-Adaptive RAG Pipeline for Secure Offline Knowledge Assistance**

**TABLE OF CONTENTS**

1. Introduction
2. Literature Survey
3. Prerequisites for Running Application
4. Design
5. Evaluation
6. Working and Output
7. Conclusion
8. Improvements
9. References

**1. INTRODUCTION**

**1.1 Project Background**

In today’s rapidly evolving technological ecosystem, Artificial Intelligence, especially large language models (LLMs), has enabled organizations to harness their accumulated knowledge very efficiently. Defence Research & Development Organisation (DRDO), given its sensitivity and classified research, necessitates secure, domain-adaptive platforms that remain offline and impervious to external networks. This project, undertaken at DLRL (Defence Electronics Research Laboratory), aimed to deploy a Retrieval-Augmented Generation (RAG) pipeline using offline LLMs for knowledge assistance—ensuring security, customizability, and operational utility for DRDO’s internal stakeholders.

**1.2 Objectives**

The core objective was to deploy a domain-adaptive RAG pipeline, operating fully offline, that allows DRDO researchers and engineers to interactively query organizational knowledge securely. The focus was on:

* End-to-end offline operation, with no external data flow
* Flexible and efficient information retrieval and generation
* Multi-system deployment (server-client architecture)
* Alignment with DRDO’s compliance & cybersecurity policies
* Exploration of cutting-edge open AI agent and LLM tools

**2. LITERATURE SURVEY**

**2.1 Retrieval-Augmented Generation (RAG)**

RAG combines the strengths of information retrieval systems and generative language models. Unlike traditional LLMs, which rely solely on pre-trained weights, RAG approaches augment input queries by retrieving relevant context from a knowledge base (Lewis et al., 2020). This allows answers to be both accurate and grounded in up-to-date information.

**Equation:**  
Given query *q*, RAG retrieves documents *D = {d1, ..., dn}* and generates answer *a* = LLM(q, D).

**Key Applications:**

* Enterprise QA bots
* Technical support
* Scientific document search

**2.2 Offline LLM Solutions**

Traditional LLMs (GPT, Bard, Claude) require cloud APIs, unsuitable for defense. Open-source alternatives—**OLAMA**, **LM Studio**, **Mistral**—allow running LLMs locally:

* OLAMA: CLI/server tool to serve models like Llama, Mistral
* LM Studio: GUI for hosting and chatting with LLMs
* Mistral: Efficient, open LLMs with high accuracy

These enable secure, air-gapped operation.

**2.3 AI Agents and Workflow Orchestration**

AI agents such as **Manus**, **N8n**, and **Suno** orchestrate multi-step tasks, combining LLMs with automation logic (scheduling, chaining, etc.). Manus, for instance, manages agent workflows in offline settings; N8n offers open-source workflow automation.

**2.4 Secure Deployment Practices**

**Security** is vital for DRDO:

* **Docker:** Isolates components
* **Kubernetes:** Orchestrates container clusters
* **Network isolation:** Prevents internet access
* **Authentication:** Local credentialing (OS-level, application-level)

**2.5 Related Work**

* RAG for enterprise: Facebook AI, 2020
* Offline LLM evaluation: OLAMA documentation, 2023
* Defence AI practices: MoD, 2023

**3. PREREQUISITES FOR RUNNING APPLICATION**

**3.1 Hardware & Network**

* Minimum 2 modern systems (Quad-Core CPU, 16 GB+ RAM, SSD)
* Operating System: Ubuntu 20.04+ / Windows 10+
* Secure internal LAN, with static IPs (no external access)

**3.2 Software Stack**

* Python 3.8+
* Required libraries: transformers, langchain, sentence-transformers, Flask/FastAPI
* Docker (v20+)
* OLAMA/LM Studio/Mistral LLM binaries and models
* Optional: Kubernetes (k8s), for scalable deployments

**Sample requirements.txt:**

transformers  
langchain  
sentence-transformers  
Flask  
gunicorn  
torch  
faiss-cpu

**3.3 Knowledge Corpus**

* Locally curated DRDO documentation
* Processed into searchable chunks

**3.4 Security**

* Air-gap system (no external LAN/WiFi)
* User authentication (local OS or application-level)
* DRDO compliance: auditing, logging

**4. DESIGN**

**4.1 Architecture Overview**

The system is a **client-server application** on a secure LAN:

* **Server:** Hosts RAG pipeline including retriever and LLM (in Docker)
* **Client:** Lightweight interface to send queries and display responses
* **Networking:** Only whitelisted IPs/ports on LAN

*[Insert Block Diagram]*

**4.2 Key Components**

* **Document Retriever**: Uses vector search (e.g., FAISS) to fetch relevant chunks
* **LLM Module**: Receives query + retrieved text; generates answer
* **Knowledge Base**: Local, processed document store
* **API Layer**: Handles requests from client, enforces authentication
* **Docker Containers**: Package each module for reproducibility and isolation

**4.3 Data Flow**

1. Client submits query
2. Server's retriever fetches relevant docs
3. LLM combines query & retrieved docs to generate answer
4. API returns answer to client

**4.4 Security Considerations**

* All data remains on-premises
* Network firewalls: restrict server-client communication
* Access-control: Only authorized users allowed
* System logs all queries and responses for audit

**5. EVALUATION**

**5.1 Objectives & Success Criteria**

* End-to-end offline operation
* Correctness & relevance of answers
* Response time (within 2s for standard queries)
* No external network calls

**5.2 Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | System | Query | Result | Status |
| 1 | Client | "Explain ECCM techniques" | Accurate, context-rich | Pass |
| 2 | Security | Remove LAN | Service unreachable | Pass |
| 3 | Server | "Who is project PI?" | Correct, per docs | Pass |

**5.3 Results & Observations**

**5.4 Challenges & Resolutions**

* **Model size:** Optimized with quantized models
* **Corpus formatting:** Developed Python scripts for preprocessing
* **Security:** Regular logs checked, system monitoring in place

**6. WORKING AND OUTPUT**

**6.1 System Setup**

1. **Prepare Knowledge Base:**
   * Process all documents into text chunks
2. **Start Server:**

docker-compose up -d

1. **Launch Client App:**
   * Connects to server's API endpoint

**6.2 Usage Example**

1. Open client interface
2. Enter: *"Explain principles of radar signal processing"*
3. Server processes & responds: "Radar signal processing involves ... [context] ..."
4. User views response on client system

**6.3 System Screenshots**

*Insert CLI logs, API outputs, or sample UI screenshots*

**6.4 Logs & Validation**

* All queries logged for security
* Output verified by project guide

**7. CONCLUSION**

The successful deployment of a domain-adaptive Retrieval-Augmented Generation (RAG) pipeline for secure offline knowledge assistance at DLRL (DRDO) represents a significant milestone in the pursuit of cutting-edge, AI-enabled solutions tailored for sensitive government and defense environments. The project addressed the unique challenges associated with creating an interactive knowledge assistance platform that functions wholly offline, thereby satisfying strict regulatory, security, and organizational requirements.

Throughout this endeavor, the design and implementation of the system brought together several advanced technologies—open-source large language models (LLMs), vector-based information retrieval, Docker containerization, and network security practices. By exploring and leveraging tools like OLAMA, LM Studio, and Mistral, the team demonstrated that state-of-the-art language models can be run entirely within local infrastructure, eschewing dependence on external APIs and minimizing the threat surface for data leakage.

**8. IMPROVEMENTS**

**8.1 Enhancing Retriever Accuracy and Customization**

* Domain-Specific Embeddings: Integrate embedding models fine-tuned on DRDO-specific data to improve retrieval relevance, rather than relying solely on general-purpose pre-trained models.
* Semantic Search Upgrades: Implement advanced search techniques such as hybrid search (combining keyword and vector search), or leverage re-ranking strategies for better answer precision.
* Context Window Optimization: Experiment with dynamic context window sizes to optimize the quantity and quality of retrieved information per query type.

**8.2 User Experience and Accessibility**

* GUI Development: Build a user-friendly graphical interface for the client-side, including features like query history, auto-suggestion, and feedback on answer quality.
* Multilingual Support: Add the ability to process queries and return answers in multiple languages prevalent within DRDO, enabling wider usability across departments.
* Accessibility Features: Ensure the interface supports screen readers and high-contrast modes for users with disabilities.

**8.3 Knowledge Base Maintenance and Expansion**

* Automated Ingestion Pipeline: Develop scripts or pipelines that periodically check for updates to DRDO documentation (from approved internal sources) and refresh the knowledge base while maintaining security protocols.
* Incremental Indexing: Adopt strategies for updating the vector store or retriever indexes efficiently as new data is added, avoiding full reprocessing.

**8.4 Robust Security and Compliance**

* Two-Factor Authentication: Upgrade user authentication mechanisms to include 2FA, especially for administrative or sensitive tasks.
* Zero Trust Architecture: Adopt a “zero trust” model for intra-network communications, ensuring every component verifies identity and encryption.
* Detailed Auditing: Enhance logging to track not just queries and responses but also access times, query intent (if categorized), and system resource usage for compliance reporting.

**8.5 Scalability and High Availability**

* Kubernetes Deployment: Fully migrate to a Kubernetes-based architecture for easier scaling, automated failover, and load balancing as the number of users grows.
* Resource Optimization: Profile system performance and introduce hardware accelerators (e.g., GPUs, TPUs) where available for faster LLM inference.
* Distributed Architecture: Consider sharding the knowledge base and/or deploying multiple retriever/LLM nodes to handle larger query volumes.

**9. REFERENCES**

1. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", NeurIPS 2020
2. OLAMA Documentation, 2023
3. LM Studio: <https://lmstudio.ai/>
4. [Mistral.ai](http://Mistral.ai) Documentation
5. DRDO Cybersecurity Policy Manual, 2021
6. Docker Documentation: <https://docs.docker.com/>
7. LangChain Documentation: <https://python.langchain.com/>
8. Manus Automation Agent Whitepaper