Submission Date:
12 December 2025

AirGap-AI

INFORMATION SYSTEMS SECURITY
INSTRUCTOR: PYUSHI SINGH

HARVANSH ANEJA
STUDENT ID: 000905463

Contents

Abstract

Organizations in privacy-sensitive sectors such as healthcare and finance face significant risks when using cloud-based artificial intelligence systems. When employees or IT staff rely on online AI tools, sensitive organizational or personal data may be transmitted to external servers, where its storage, reuse, or exposure is not fully transparent to the user. This lack of visibility raises concerns related to regulatory compliance, particularly with frameworks such as HIPAA and GDPR, and may result in legal, financial, or reputational consequences if data is mishandled or leaked. In addition to privacy concerns, cloud-based AI systems depend on continuous internet access and external service availability, which can lead to reliability issues such as latency, outages, or loss of access during network disruptions.

To address these challenges, this project presents AirGAP-AI, a fully offline AI-based technical support assistant designed to operate entirely on a local system without requiring any network connectivity. The system can be configured to disable chat history storage, ensuring that user-provided information is removed once the session ends. This design minimizes the persistence of sensitive information and reduces the risk of unintended data exposure. AirGAP-AI is intentionally constrained and trained not to provide medical instructions; instead, it is designed for general technical troubleshooting, particularly for equipment or specialized software systems where trained support personnel may be limited.

AirGAP-AI is built using a locally deployed LLaMA 3.1 language model with prompt-based behavior control and minimal fine-tuning to guide response behavior and restrict disallowed topics. Knowledge is embedded directly into the model and supplemented through retrieval-augmented generation (RAG) using locally stored reference files. Encryption is applied separately to selected files that may be used as RAG inputs; however, encryption and logging mechanisms are only partially designed and simulated at a conceptual level, with implementation attempts explored but not integrated into the final system. Role-based access control was identified as more straightforward to implement and was partially validated at a basic level, but it is not fully enforced in the current architecture. The system operates fully offline and meets the project's minimum viable product (MVP) objectives. The primary technical success of the project was achieving stable offline operation and controlled AI behavior. Current limitations include the absence of image input support due to memory constraints.

Introduction

Artificial intelligence is increasingly used within organizations to support technical troubleshooting, automation, and decision-making. In many cases, cloud-based AI systems

are adopted due to their accessibility and ease of use. However, in security- and compliance-sensitive environments, such as healthcare, finance, and regulated enterprises, reliance on cloud-based AI introduces significant risks related to data privacy, regulatory compliance, and operational reliability.

In these environments, sensitive technical information, operational details, or user data may be inadvertently transmitted to external systems beyond organizational control. Additionally, cloud-based solutions depend on continuous internet connectivity and third-party service availability, which may not be acceptable or feasible in restricted or isolated network environments. These constraints create a need for alternative AI solutions that can operate locally while maintaining usability and effectiveness.

This capstone project explores the design and implementation of **AirGAP-AI**, a privacy-first, fully offline artificial intelligence system intended to provide technical support without reliance on cloud infrastructure. The project focuses on demonstrating the feasibility of deploying a locally hosted AI assistant that prioritizes data control, predictable behavior, and security-conscious design. By examining both technical implementation and practical limitations, this project contributes to understanding how offline AI systems can be applied within information security contexts.

Project Statement and Motivation

Organizations operating in privacy-sensitive and regulated sectors face ongoing challenges when adopting cloud-based artificial intelligence tools for technical support. While these tools can improve efficiency, they often require transmitting queries, logs, or contextual information to external servers. This lack of transparency over how data is stored, processed, or reused raises concerns related to compliance with regulations such as HIPAA and GDPR, as well as broader organizational risk management.

In addition to privacy concerns, cloud-based AI systems introduce operational dependencies on internet connectivity and third-party service availability. Network outages, latency, or service disruptions can prevent access to critical support tools when they are most needed. Furthermore, many organizations intentionally restrict internet access on internal systems to reduce attack surfaces, making cloud-based AI solutions impractical or prohibited.

The motivation for this project stems from the need to address these limitations by exploring a **fully offline AI-based technical support system**. AirGAP-AI is designed to operate entirely on local infrastructure, ensuring that sensitive information remains under organizational control while still providing meaningful assistance to users. By focusing on offline operation, constrained AI behavior, and retrieval of locally stored documentation, this project seeks to demonstrate a practical alternative to cloud-dependent AI solutions within information systems security environments.

Project Description and Overview

AirGAP-AI is an offline artificial intelligence system designed to provide technical assistance in environments where internet access is restricted, unreliable, or intentionally disabled due to privacy and security requirements. The system is intended for deployment in sensitive environments such as healthcare facilities, secure workplaces, and organizations operating on isolated or local-only networks, where access to cloud-based AI services or external technical support resources may be impractical, prohibited, or associated with unacceptable risk.

The primary objective of AirGAP-AI is to assist users with basic to intermediate technical issues without requiring immediate access to human technical support. In many organizations, IT support teams are frequently overwhelmed or focused on high-priority incidents, while users continue to experience routine or repetitive technical problems. AirGAP-AI addresses this gap by providing locally accessible guidance that helps users resolve common issues or determine appropriate next steps. When the system does not have sufficient information to generate a reliable response, it explicitly communicates this limitation and directs users to reframe their query or contact technical support, thereby avoiding speculative or misleading guidance.

The system is designed primarily for non-technical users, including administrative staff, receptionists, and personnel who interact with specialized equipment or software but do not possess advanced technical expertise. These users often encounter operational issues related to their tools or workflows but lack the background necessary to diagnose or resolve problems independently. By delivering responses in clear, natural language, AirGAP-AI enables such users to obtain assistance without escalating every issue to IT staff. As a result, IT professionals benefit indirectly by reducing the volume of basic support requests, allowing them to focus on more complex or critical technical tasks.

A strict offline operating model was a core requirement of the project due to technical, legal, and practical considerations. From a security and compliance perspective, transmitting data to third-party cloud services introduces uncertainty regarding how information is stored, processed, or reused, which may conflict with regulatory obligations in privacy-sensitive sectors. From a reliability standpoint, local execution removes dependency on external services and network performance, ensuring consistent availability even in environments with limited connectivity. Additionally, the system can be deployed on modest local hardware without ongoing subscription costs, making it a cost-effective solution that can be maintained internally by trained IT staff. These design requirements directly informed the system architecture and technology choices described below.

At a high level, AirGAP-AI is built around a locally deployed large language model executed entirely on-device. Model deployment and runtime management are handled using Ollama, enabling fully local execution without cloud connectivity. User interaction is provided through Anything LLM, which serves as the local interface for submitting queries and receiving responses. Knowledge resources are stored locally, typically in document formats such as PDF files, which can be compressed and protected when required.

Model training and experimentation activities were conducted using machine learning frameworks such as TensorFlow and PyTorch, allowing for limited fine-tuning and behavior adjustment. LM Studio was used during development to test, validate, and refine model responses in a controlled offline environment. The system was developed and tested primarily on a Windows platform; however, the overall architecture is not operating-system specific and can be adapted for Linux or macOS systems that support local language model execution.

Methodology and Project Planning

The project was initially structured under a fixed, waterfall-style framework as defined by the course requirements, with predetermined deliverables assigned to each sprint. However, this approach proved to be poorly suited for the nature of the project, particularly due to the exploratory and unpredictable behavior of artificial intelligence systems. As a result, the project execution was adapted to follow an Agile-inspired methodology, emphasizing incremental development, continuous evaluation, and iterative refinement rather than strict adherence to predefined deliverables.

Although the course defined fixed sprint deliverables, these were treated as high-level checkpoints rather than rigid completion criteria, allowing technical decisions to evolve while still meeting academic requirements.

An Agile approach was selected because the project required frequent reassessment of technical feasibility, tooling, and model behavior. Early assumptions regarding suitable AI models, training approaches, and datasets did not consistently hold true in practice. Several ideas that appeared viable during planning stages failed during implementation, while other approaches only partially succeeded. Agile development allowed the project to progress by resolving immediate technical obstacles, learning from failures, and adjusting direction as needed, rather than forcing completion of rigid sprint objectives that no longer aligned with project realities.

This iterative approach also functioned as a risk-management strategy, allowing technical risks to be identified early and preventing excessive time investment in approaches that proved infeasible.

This flexibility was particularly important due to the unpredictable nature of AI behavior and training outcomes. Initial efforts focused on identifying an appropriate language model capable of supporting offline technical assistance. Early experimentation with pre-trained models, including a version of DeepSeek AI designed for technical support, revealed significant limitations. These included difficulty integrating document-based knowledge, limited retrieval-augmented generation (RAG) support, and challenges fine-tuning the model due to dataset conflicts and quality issues. As a result, the project required multiple shifts in tooling and architectural approach, which would not have been feasible under a strict waterfall methodology.

The project was divided into three sprints, each approximately three weeks in duration, with clearly defined but adaptable goals. Sprint 1 focused on establishing the project foundation, including environment setup, tool evaluation, and initial dataset exploration. During this sprint, machine learning frameworks such as PyTorch and TensorFlow were configured and evaluated for training feasibility under limited hardware resources. Dataset exploration involved reviewing multiple sources from platforms such as Hugging Face and GitHub and selectively extracting relevant portions rather than using complete datasets, as conflicting question–answer pairs were found to degrade model performance. This phase involved extensive debugging, frequent memory-related failures, and system crashes due to resource constraints, requiring repeated refinement of training configurations.

By Sprint 2, sufficient progress had been made to achieve a functional minimum viable product (MVP). The model was capable of responding to a limited set of technical queries, meeting the baseline project requirements. However, it became evident that training alone was insufficient for maintaining accuracy and adaptability over time. The difficulty of updating a fully trained model with new information led to the adoption of a retrieval-augmented generation approach. This shift required moving away from an LM Studio–only workflow toward the use of Ollama and Anything LLM, which provided stronger support for RAG-based knowledge integration. LM Studio continued to play a key role during this phase by enabling detailed inspection of model outputs and token-level behavior through logs, which was essential for understanding response quality and identifying hallucination risks.

Sprint 3 focused on refinement, validation, and performance optimization. Key objectives included reducing response uncertainty, limiting undesired or irrelevant outputs, and ensuring the model could operate efficiently on a standard laptop through quantization and optimization techniques. Additional emphasis was placed on exploring security- and compliance-related features such as role-based access control, encryption, and logging. While these mechanisms were not fully implemented, conceptual designs and partial experiments were conducted to evaluate feasibility within the project's time and resource

constraints. Each sprint concluded with a retrospective-style review, during which outcomes were evaluated, failures were analyzed, and priorities for the following sprint were adjusted accordingly.

Project tracking and documentation were managed primarily through Excel-based trackers, supplemented by sprint status reports, a business canvas, a project plan, and an AI interaction log. As a solo developer, task-based tracking proved more practical than hour-based tracking, particularly during AI training, where system processes often ran unattended while requiring intermittent monitoring.

Tracking data was used directly to inform technical decision-making, helping determine when an approach was producing diminishing returns and when a change in strategy was necessary.

The project was developed entirely by a single contributor, which introduced significant constraints related to time, hardware resources, and access to high-quality datasets. No commercial datasets or compliance certification resources were available, necessitating reliance on publicly available or hobbyist-created data and simulated compliance analysis. These constraints strongly influenced prioritization decisions, with emphasis placed on achieving core functionality and MVP completion rather than full implementation of advanced security features. Continuous evaluation, informed by experimentation, peer discussion, and iterative testing, enabled progress despite these limitations and ensured that effort remained focused on feasible and high-impact outcomes.

Sprint 1 Summary: Foundation and Environment Setup

Sprint 1 Objectives

The primary goal of Sprint 1 was to establish a functional foundation for the AirGAP-AI project. This sprint focused on setting up the development environment, configuring core tools, and evaluating the feasibility of offline AI training under limited hardware and time constraints. Planned objectives included environment setup, installation of AI training frameworks, initial dataset collection and preparation, early model training attempts, implementation of basic storage security, and configuration of backup mechanisms, as defined in the project plan and Sprint 1 backlog

Planned Activities

At the beginning of Sprint 1, the following activities were planned:

- Set up the development environment, including GitHub, an Excel tracking system, a local project wiki, and a secure local repository
- Configure encrypted local storage for project documentation and datasets
- Install and configure AI training frameworks, specifically PyTorch and TensorFlow
- Collect and prepare initial datasets for AI training
- Begin training a simple AI model to evaluate feasibility
- Write initial unit tests for environment setup where applicable
- Configure regular backup mechanisms to protect project data

These activities were intended to provide a stable technical base for later sprints and reduce uncertainty around tooling and feasibility.

Actual Work Performed

During Sprint 1, significant effort was dedicated to environment setup and tool evaluation. A protected local repository was created for project documentation and datasets, along with a structured backup system to ensure data reliability. A local wiki and Excel-based tracker were established to document design decisions, progress, and issues encountered. GitHub was evaluated but not fully integrated during this sprint, resulting in a local-only version control approach at this stage.

AI training frameworks, including PyTorch and TensorFlow, were installed and configured. Multiple attempts were made to train and fine-tune an initial AI model using a pre-trained DeepSeek model designed for technical support. These attempts revealed several challenges, including limited familiarity with the training frameworks, dataset quality issues, and conflicts between existing model training and newly introduced data. Considerable time was spent exploring datasets from sources such as Hugging Face and GitHub, identifying usable subsets, and learning how to construct custom question–answer datasets from available documentation.

Sprint 1 also involved extensive experimentation to ensure training could be performed within the constraints of limited local hardware. Training runs frequently encountered memory limitations, requiring repeated adjustment of batch sizes, parameters, and configurations. Several training attempts were interrupted due to system instability, highlighting the practical challenges of offline AI development on non-specialized hardware.

In parallel, weekly backup routines were implemented, and initial outreach was conducted to industry contacts in an effort to obtain non-sensitive documentation related to medical devices for later testing and validation.

Deliverables

By the end of Sprint 1, the following deliverables were completed:

- A functional local development environment with configured AI frameworks
- Secure local storage for documents and datasets
- An operational Excel-based tracking system and project wiki
- Initial dataset exploration and partial dataset preparation
- Multiple AI training attempts to validate feasibility
- Weekly backup mechanisms for project files
- Initial industry outreach for documentation and reference materials

These deliverables aligned with the Sprint 1 plan and status report submissions

Challenges and Impediments

Sprint 1 faced several impediments that influenced project direction. As a solo developer, available debugging time was limited, and learning curves associated with PyTorch and TensorFlow slowed progress. Dataset inconsistencies and conflicting labels caused repeated training failures, while GPU and memory constraints required conservative training configurations that increased iteration time. Additionally, delays in receiving industry documentation limited access to realistic test data during this sprint.

Key Outcomes and Lessons Learned

Sprint 1 provided critical insight into the complexity of offline AI development. Key lessons included the importance of dataset quality, the need for careful resource management when training models locally, and the value of early experimentation to expose feasibility issues. The sprint also demonstrated that environment stability, tooling familiarity, and documentation practices were essential prerequisites for meaningful progress in later sprints.

Although a fully functional AI model was not achieved during Sprint 1, the sprint successfully reduced uncertainty around tooling and constraints, enabling more informed technical decisions in subsequent sprints and directly shaping the project's transition toward a minimum viable product in Sprint 2.

6. Sprint 2 Summary: Minimum Viable Product Development

Sprint 2 Objectives and Planned Activities

The primary objective of Sprint 2 was to transition from foundational experimentation to a functional system by achieving a **minimum viable product (MVP)**. This sprint focused on integrating the trained model into an offline assistant, improving dataset quality, and validating that the system could operate reliably without network connectivity. Planned activities included addressing dataset formatting issues identified in Sprint 1, deploying the model within a local execution environment, implementing a basic interface for user interaction, exploring encrypted knowledge storage and role-based access control, and conducting functional tests to verify response accuracy and offline operation.

These objectives were designed to confirm that the core concept—an offline AI-based technical support assistant—was feasible and demonstrable within the project's constraints.

Actual Work Performed

During Sprint 2, previously trained models and lessons learned from Sprint 1 were consolidated into a functional offline assistant using **LM Studio** as the primary runtime and testing environment. Dataset formatting issues were partially resolved, resulting in noticeable improvements in response consistency and relevance. Additional fine-tuning was performed using limited, non-sensitive industry-provided information to improve domain alignment; however, due to usage restrictions, this data was not incorporated into the final system artifacts.

A local encrypted data repository for troubleshooting and support information was designed and initially implemented during this sprint. While this component appeared viable at the time, it was ultimately not retained in the final system **due to later architectural changes rather than technical infeasibility**. This outcome reflects the iterative nature of the project and the need to adapt architectural decisions as system requirements evolved.

LM Studio provided an effective environment for testing natural-language interactions, inspecting token-level output, and evaluating model behavior. Its use removed the immediate need to develop a custom user interface, allowing focus to remain on validating offline functionality and response quality. Offline operation was verified by disabling network connectivity during execution, and functional testing was conducted using representative technical queries to assess system behavior.

Deliverables

By the end of Sprint 2, the following deliverables were achieved:

- A working offline AI assistant operating locally within LM Studio
- Improved dataset structure and retraining compared to Sprint 1
- Verified offline operation with network connectivity disabled
- Functional natural-language response capability
- Completion of the project's **minimum viable product (MVP)**
- Partial exploration of encrypted knowledge storage and role-based access control

Although some components did not persist into later sprints, the MVP was complete and suitable for demonstration at the conclusion of Sprint 2.

Challenges and Limitations

Sprint 2 faced several challenges that influenced subsequent design decisions. Feedback opportunities were limited due to the solo-developer structure and the absence of formal peer or industry review during this phase. While LM Studio proved valuable for testing and inspection, it lacked robust support for retrieval-augmented generation (RAG), limiting the system's ability to scale knowledge without retraining the model.

Additionally, reliance on model-embedded knowledge raised maintainability concerns, as updates would require retraining rather than the addition of external reference material. Partial implementations of role-based access control and encrypted storage were constrained by platform limitations and were not fully viable within the chosen environment.

Key Outcomes and Lessons Learned

Sprint 2 marked a critical milestone by demonstrating that a fully offline AI-based technical support assistant could function at a basic but usable level. Improvements in dataset quality and iterative testing significantly enhanced response accuracy and reduced uncertainty. The sprint also revealed key architectural limitations, particularly related to extensibility, security customization, and long-term maintainability.

These findings directly informed the architectural transition and refinement efforts undertaken in Sprint 3.

7. Sprint 3 Summary: Refinement, Optimization, and Finalization

Sprint 3 Objectives

Sprint 3 functioned primarily as a **stabilization and refinement phase**, focusing on improving reliability, performance, safety, and usability rather than introducing new core features. The planned goals for this sprint included completing AI training and refinement, simulating **compliance-oriented behavioral checks**, optimizing model performance for execution on local devices, finalizing documentation, and exploring the feasibility of feedback mechanisms and additional security controls.

During this sprint, it became clear that several assumptions made earlier—particularly regarding long-term use of LM Studio—would not hold for the final system. As a result, Sprint 3 involved significant architectural reassessment alongside refinement efforts.

Architectural Transition and Implementation Work

Early in Sprint 3, limitations in LM Studio became increasingly restrictive, particularly with respect to extensibility, retrieval-augmented generation (RAG) support, and long-term deployment flexibility. To address these limitations, the project transitioned to **Ollama** as the backend runtime and **Anything LLM** as the frontend interface. Ollama was selected due to its flexibility in managing locally hosted models and its support for custom model files, while Anything LLM was chosen for its usability, configurable controls, and built-in support for RAG functionality.

This architectural shift enabled the system to separate core language model execution from knowledge retrieval, addressing scalability and maintainability issues identified in Sprint 2. RAG functionality was implemented through Anything LLM, allowing reference documents to be accessed dynamically without requiring retraining of the base model. This change represented a major improvement over the earlier LM Studio–only approach, which relied heavily on model-embedded knowledge.

Model Refinement, Image-Based Exploration, and Performance Optimization

Significant effort during Sprint 3 was dedicated to refining model behavior and optimizing performance for local execution. Dataset integrity was reviewed and corrected, enabling final rounds of fine-tuning and stability testing. Extensive manual testing was conducted by submitting a wide range of queries to evaluate response accuracy, detect hallucinations, and

verify that the model remained within defined behavioral constraints, particularly regarding medical and sensitive information.

In parallel, image-based input capabilities were explored to evaluate whether visual data could be incorporated into the system. This included experimentation with image interpretation models and illumination-sensitive inputs, as well as attempts to integrate image processing alongside the primary language model and RAG pipeline. These experiments required substantial time investment due to the need for additional datasets, model fine-tuning, and coordination between multiple concurrently running models.

Ultimately, running three models simultaneously—one for the primary language model, one for RAG retrieval, and one for image interpretation—exceeded available system resources. In addition, limitations in training data quality for image-based tasks resulted in unstable behavior, including increased hallucination risk and inconsistent outputs. Due to these constraints, and given the project's emphasis on reliability and safety in privacy-sensitive environments, image-based input was intentionally removed from the final system.

Model quantization was explored extensively to balance accuracy and hardware constraints. Multiple quantized variants were generated, including 2-bit, 3-bit, 4-bit, 5-bit, and 6-bit versions. Lower-bit models were found to be overly unstable, while higher-bit models exceeded practical memory limits. The final configuration selected for the project was an **8-billion-parameter model using 4-bit quantization**, representing the best balance between performance, accuracy, and feasibility on local hardware.

Security and Compliance Considerations

Sprint 3 included focused exploration of security and compliance-related concerns. Model behavior was constrained to prevent the generation of medical advice or the handling of patient-specific information. Compliance-oriented behavioral checks were conducted by manually evaluating system responses against known regulatory expectations and identifying obvious policy violations. These checks were **behavioral and functional in nature** and were not intended to represent legal or regulatory certification.

Role-based access control and encrypted logging mechanisms were explored but not fully implemented due to time and complexity constraints. While Anything LLM provides foundational support for access control and local log storage, full enforcement and encryption of logs were not completed within the project timeline. These features remain feasible future enhancements rather than completed components of the final system.

Challenges and Limitations

Sprint 3 faced several challenges that constrained implementation scope. The late-stage architectural transition from LM Studio to Ollama and Anything LLM required rapid

adaptation and limited the time available for completing advanced features such as formal feedback mechanisms and comprehensive role-based access enforcement. Dataset preparation and validation continued to require substantial effort, and compliance evaluation remained largely manual and time-intensive.

As with earlier sprints, the project was constrained by solo development, limited hardware resources, and reliance on non-commercial datasets. While Anything LLM improved flexibility and usability, it also imposed limits on how deeply security mechanisms could be customized within the available timeframe.

Final Outcomes and Lessons Learned

Sprint 3 successfully transformed a functional MVP into a **stable, optimized, and demonstrable offline AI system**. Performance optimization and behavioral refinement significantly reduced hallucination risk and improved response reliability. The sprint reinforced the importance of prioritizing dataset quality, system constraints, and safety controls over feature expansion, particularly in privacy-sensitive environments.

By the end of Sprint 3, the system consisted of a **fully offline AI assistant using an Ollama backend and Anything LLM frontend**, with retrieval-augmented generation enabled, a quantized model optimized for local execution, and constrained behavior appropriate for security- and privacy-conscious use cases. Although some planned features were not completed, the final system clearly demonstrated the feasibility and practicality of privacy-first, offline AI support systems. These outcomes provide the basis for the final system overview and evaluation presented in the following sections.

8. Final System Overview: Consolidated Architecture and Data Flow

System Architecture Overview

At the conclusion of Sprint 3, AirGAP-AI exists as a fully offline, locally deployed AI-based technical support system composed of a modular backend, frontend, and retrieval pipeline. The final architecture emphasizes privacy, reliability, and local execution while remaining lightweight enough to operate on standard modern hardware.

The backend runtime is provided by **Ollama**, which is responsible for hosting and executing the language model locally. The frontend interface is implemented using **Anything LLM**, which provides the user-facing interaction layer and manages retrieval-augmented generation (RAG) workflows. Together, these components enable natural-language interaction without any dependency on external cloud services.

The core language model used in the final system is a **fine-tuned LLaMA 3.1 model with 8 billion parameters**, optimized for local execution through forward quantization. This model was selected to balance response quality and hardware feasibility. Retrieval-augmented generation is supported using **Nomic text embeddings**, which were found to provide stable performance and efficient semantic retrieval in an offline environment.

Supporting resources, such as documentation and reference materials, are stored locally. These resources reside in a separated, encrypted folder structure and can be selectively integrated into the RAG pipeline through Anything LLM as needed. This separation allows future updates to documentation without requiring changes to the core model.

End-to-End Data Flow

The system follows a clear and deterministic data flow when handling user queries. When a user submits a question through the **Anything LLM** interface, the request is forwarded to the **Ollama** backend for processing. The backend analyzes the query to determine whether additional contextual information is required.

If relevant reference material is needed, the system issues a retrieval request to the RAG component managed by Anything LLM. Using Nomic text embeddings, the RAG system searches the locally stored documentation and returns the most relevant excerpts. These retrieved references are then passed back to the language model as contextual input.

The model generates a response by combining its internal knowledge with the retrieved documentation, prioritizing information sourced directly from reference materials when available. If insufficient information is available to produce a reliable answer, the system explicitly indicates this limitation to the user rather than generating speculative output. The final response is returned to the user through the frontend interface.

Once the interaction concludes, no data is transmitted externally, and all processing remains confined to the local system.

Offline Operation and Deployment Model

AirGAP-AI operates **entirely offline** and does not require any internet connectivity for normal operation. All model execution, retrieval, and response generation occur locally. Optional network access may be used only for administrative purposes, such as manually updating documentation used by the RAG system, depending on how the system is deployed on-site.

The system is designed to run either on a centralized local server or directly on individual workstations within an organization’s internal network. Deployment can be tailored to

organizational needs without altering the core architecture. In all cases, external connectivity is not required for system functionality.

Security and Data Handling Posture

The final system enforces a conservative and privacy-focused data handling approach. Chat persistence is configurable: conversations may be automatically deleted at the end of a session or saved locally for internal compliance review, depending on organizational policy. Saved chats are stored locally and can be encrypted separately if required.

Document handling is restricted to non-sensitive reference materials intended for technical support purposes. Sensitive personal or patient-specific data is not required for system operation and is intentionally excluded from training and retrieval workflows.

Logging is minimal by design. Aside from optional chat persistence, only basic operational logs are generated to support troubleshooting or system diagnostics. Role-based access control (RBAC) was explored but is **not implemented** in the current system. While Anything LLM provides mechanisms that could support RBAC, full enforcement was deferred due to project scope and time constraints.

System Constraints and Limitations

The final system has several clearly defined constraints. Image-based input and processing are not supported, as prior experimentation demonstrated that integrating vision models alongside the language model and RAG pipeline exceeded available system resources and reduced overall reliability. As a result, the system is intentionally limited to text-based interaction.

From a hardware perspective, the system requires a **modern CPU and a minimum of 16 GB of RAM** to operate reliably. Systems with Intel processors from the 10th generation or newer were found to provide acceptable performance. GPU acceleration is beneficial but not strictly required.

Compliance evaluation within the system is conducted manually. There is no automated compliance testing or formal legal certification. Instead, the system has been behaviorally validated to avoid obvious regulatory violations, such as generating medical advice or handling protected personal information. These checks are intended to reduce risk rather than to guarantee regulatory compliance.

Summary

The final AirGAP-AI architecture demonstrates that a privacy-first, offline AI assistant can be implemented using locally deployed tools without reliance on cloud infrastructure. By combining a fine-tuned and 4-bit quantized large language model, retrieval-augmented generation, and a modular frontend-backend design, the system achieves reliable offline operation while remaining adaptable to future enhancements.

Most importantly, the final system validates the feasibility of deploying practical, privacy-conscious AI support tools in environments where cloud-based solutions are unsuitable or prohibited. This outcome directly supports the project's core objective and reinforces the applicability of offline AI architectures within security- and compliance-sensitive domains.

9. Challenges and Lessons Learned

Technical Challenges

One of the most significant challenges throughout the project was the **unpredictable nature of AI behavior**, particularly during training and fine-tuning. Early assumptions that a pre-trained model could be easily adapted for offline technical support proved incorrect. Initial attempts to fine-tune a pre-trained technical support model resulted in conflicting behaviors, inconsistent answers, and difficulty controlling output, largely due to dataset incompatibilities and limited visibility into how prior training influenced responses. These issues required repeated experimentation and ultimately contributed to a change in model selection and training strategy.

Dataset quality and preparation emerged as a persistent challenge across all sprints. Publicly available datasets varied widely in structure, relevance, and accuracy, and combining data from multiple sources often introduced contradictions that degraded model performance. Significant time was spent identifying usable subsets, cleaning data, and creating custom question–answer pairs from documentation. The project demonstrated that dataset quality had a greater impact on response reliability than model size alone, and that insufficient or poorly aligned data increased hallucination risk even when the model appeared to be functioning correctly.

Hardware and resource limitations also strongly influenced development decisions. Training and experimentation were conducted on non-specialized hardware, which resulted in frequent memory constraints, system instability, and long iteration cycles. Tasks such as fine-tuning, quantization, and multi-model experimentation—particularly when image-based processing was explored—often exceeded available resources. These constraints forced careful prioritization and ultimately led to the removal of image-based input from the final system in favor of stability and reliability.

The project also encountered challenges related to **tooling limitations and architectural fit**. While LM Studio proved valuable for early testing, token-level inspection, and rapid experimentation, it lacked the extensibility required for long-term deployment, particularly for retrieval-augmented generation and security customization. The late-stage transition to Ollama and Anything LLM introduced additional complexity and time pressure but was necessary to achieve the project's final architectural goals.

Security and Compliance Challenges

Security and compliance considerations introduced both technical and conceptual challenges. Ensuring that the AI did not generate medical advice, handle patient-specific information, or violate regulatory expectations required careful prompt design, constrained response behavior, and extensive manual testing. Compliance-oriented behavioral checks were time-intensive and required familiarity with regulatory principles rather than reliance on automated tooling.

Implementing advanced security features such as encrypted logging and role-based access control proved more complex than anticipated within the project's timeframe. While these features were explored conceptually and partially tested, full enforcement was not achieved due to architectural transitions, tooling constraints, and prioritization of core system stability. These limitations reinforced the reality that security features must be planned alongside architecture from the outset to avoid late-stage integration challenges.

Project and Process Challenges

Developing the project as a **solo contributor** introduced significant scope and time-management challenges. The breadth of responsibilities—including model training, dataset preparation, architecture design, security analysis, documentation, and testing—required frequent trade-offs between depth and completeness. Without additional team members, time spent troubleshooting or experimenting with non-viable approaches directly reduced time available for feature completion.

The initial course-imposed waterfall-style structure also presented challenges for an AI-driven project. Fixed deliverables did not align well with the experimental and iterative nature of AI development, where feasibility often cannot be determined until implementation is attempted. Transitioning to an Agile-inspired approach allowed the project to adapt more effectively but required ongoing effort to balance academic requirements with technical reality.

Tracking effort also proved non-trivial. Traditional hour-based tracking was difficult to apply to AI training workloads, where processes often ran unattended but still required

periodic supervision and intervention. As a result, task-based tracking became the primary mechanism for monitoring progress and controlling scope.

Key Lessons Learned

A central lesson of the project was that **stability and reliability outweigh feature completeness** in privacy-sensitive environments. Several technically interesting features—such as image-based input, advanced ranking mechanisms, and deeper security enforcement—were intentionally excluded from the final system when they threatened reliability, accuracy, or maintainability. This prioritization proved critical to delivering a usable and defensible final product.

The project also reinforced the importance of **iterative validation and early failure detection**. Regular self-review and sprint retrospectives helped identify when approaches were not yielding meaningful progress, enabling timely pivots rather than prolonged investment in ineffective solutions. This approach functioned as an informal but effective risk-management strategy.

Another key takeaway was the value of **architectural flexibility**. The transition from LM Studio to Ollama and Anything LLM, while disruptive, ultimately enabled retrieval-augmented generation, improved maintainability, and better alignment with offline deployment goals. This highlighted the need to select tools not only for immediate convenience but also for long-term extensibility.

Finally, the project demonstrated that **privacy-first, offline AI systems are feasible but require deliberate design choices**. Constraints related to data handling, compliance, and safety must be addressed at every stage of development, not retrofitted at the end. While the final system is not production-certified, it provides a realistic and practical foundation for deploying AI support tools in environments where cloud-based solutions are unsuitable or prohibited. These challenges and lessons highlight clear opportunities for future enhancement, which are outlined in the following section.

10. Future Work and Improvements

Overview

While the final AirGAP-AI system demonstrates the feasibility of a privacy-first, offline AI support assistant, several areas for improvement and extension were identified during development. These opportunities primarily reflect limitations imposed by time, tooling maturity, hardware constraints, and the scope of a solo-developer capstone project.

Addressing these areas should significantly enhance the system's robustness, scalability, and suitability for real-world deployment.

Security and Access Control Enhancements

One of the most important areas for future work is the **full implementation of role-based access control (RBAC)**. Although RBAC concepts were explored and partially validated, enforcement was not completed in the final system. Future development **should** implement RBAC to clearly distinguish between employees, IT staff, and administrators, ensuring that access to sensitive functionality and stored data is appropriately restricted.

Similarly, **encrypted logging and audit trail support** should be fully implemented. While chat persistence is configurable and logs can be stored locally, comprehensive encryption and controlled export of logs were not completed within the project timeline. Future work **should** introduce encrypted log storage, secure export mechanisms, and configurable retention policies to better support internal audits and compliance workflows.

Automated Compliance Validation

Compliance evaluation in the current system relies on **manual, behavior-based review**. Future iterations **could** explore automated compliance validation mechanisms, such as rule-based response filters, policy enforcement layers, or automated test suites designed to detect prohibited outputs. These mechanisms would not replace formal legal certification but could reduce manual effort and help identify compliance risks earlier in the development lifecycle.

Model and Dataset Expansion

Future work **should focus on expanding and refining datasets** used for both training and retrieval. Access to higher-quality, domain-specific datasets—particularly curated or professionally reviewed documentation—would improve response accuracy and reduce ambiguity. Separating general troubleshooting knowledge from organization-specific documentation would also improve maintainability and allow tailored deployments without retraining the core model.

In addition, improved dataset management practices and selective retraining workflows should be explored to allow updates as systems, devices, or procedures change, without requiring full model retraining.

Interface and Usability Improvements

While Anything LLM provided a practical frontend for this project, future work should prioritize the development of a **custom user interface** tailored to organizational workflows and security requirements. A custom interface would enable features not fully supported in the current setup, including stronger role-based access control enforcement, improved encrypted logging, and more granular administrative controls.

Developing a custom interface would also enable tighter integration with internal systems and allow more precise control over how users interact with the AI assistant.

Performance and Deployment Considerations

The current system operates reliably on modern consumer hardware, and performance optimization is not a primary limitation. While future optimizations could further reduce resource usage or improve response times, the system already functions effectively on lower-end hardware, albeit with slower performance. As such, performance optimization remains a secondary consideration compared to security, reliability, and correctness.

Reintroducing Image-Based Capabilities

Image-based input was intentionally excluded from the final system due to stability, resource, and training-data constraints. However, future work should reintroduce image-based capabilities once the system is deployed on stronger hardware or centralized servers and sufficient development time is available.

With improved multimodal datasets and targeted training to exclude sensitive information during optical character recognition (OCR), image-based support could significantly enhance the system's usefulness. Potential applications include analyzing screenshots, error dialogs, or device indicators, provided these features can be implemented without compromising privacy, security, or reliability.

11. Conclusion

This project set out to explore whether a practical, privacy-first artificial intelligence system could be deployed entirely offline to support technical troubleshooting in security- and compliance-sensitive environments. Organizations in sectors such as healthcare and finance face increasing risks when relying on cloud-based AI tools, including data exposure, regulatory uncertainty, and operational dependency on external services. AirGAP-AI was developed in response to these challenges, with the goal of demonstrating a locally deployed alternative that prioritizes data control, reliability, and safety.

Through an iterative, Agile-inspired development process, the project progressed from foundational experimentation to a functional minimum viable product and ultimately to a stable, optimized offline system. The final implementation combines a fine-tuned and 4-bit quantized LLaMA 3.1 language model with retrieval-augmented generation, delivered through a modular architecture using Ollama and Anything LLM. This design enables natural-language technical assistance without requiring internet connectivity, while maintaining clear constraints on behavior to avoid unsafe or non-compliant outputs.

The project demonstrated that offline AI systems are not only technically feasible but also practical when developed with careful attention to dataset quality, architectural flexibility, and system limitations. Key technical challenges—including dataset inconsistencies, hardware constraints, tooling limitations, and late-stage architectural changes—were addressed through iterative refinement and informed prioritization. Several advanced features, such as image-based input, automated compliance validation, and full security enforcement, were intentionally deferred in favor of stability, accuracy, and demonstrability.

Beyond the technical outcome, the project provided valuable insight into the realities of developing AI systems under real-world **operational** constraints. It highlighted the importance of early experimentation, continuous validation, and the willingness to abandon non-viable approaches. The experience reinforced that in privacy-sensitive contexts, reliable behavior and predictable operation are more critical than feature completeness.

In conclusion, AirGAP-AI validates the feasibility of deploying offline, privacy-conscious AI support tools in environments where cloud-based solutions are unsuitable or prohibited. While the system is not production-certified, it establishes a strong foundation for future development and demonstrates how local AI architectures can be responsibly applied within security-focused domains. **The lessons learned from this project contribute to a broader understanding of how artificial intelligence can be adapted to meet stringent privacy, reliability, and compliance requirements without reliance on cloud infrastructure.**

Appendix A: Project Documentation

The following documents were created and used throughout the AirGAP-AI capstone project to support planning, tracking, implementation, and evaluation activities.

- Business Model Canvas – AirGAP-AI

- Team / Individual Project Contract

- Project Plan and Schedule

- Sprint 1 Status Report

- Sprint 2 Status Report

- Sprint 3 Status Report

- Sprint Presentations and Demonstrations

- AI Training and Interaction Logs

- Final Showcase and Peer Review Documentation

Appendix B: Tools, Frameworks, and Platforms

- Ollama – Local LLM runtime and model management

- Anything LLM – Frontend interface and RAG orchestration

- LLaMA 3.1 (8B) – Base language model

- Nomic Text Embeddings – Retrieval-augmented generation

- LM Studio – Model testing and early experimentation

- PyTorch – Model training and fine-tuning framework

- TensorFlow – Model experimentation and training

- Microsoft Excel – Task tracking and project management

- Microsoft Word – Documentation and reporting

Appendix C: Reference Resources and Websites

The following publicly available resources were consulted during the project for datasets, documentation, tooling guidance, and regulatory understanding.

- Hugging Face – https://huggingface.co

- GitHub – https://github.com/harvansh-aneja/AirGap-AI

- Ollama Documentation

- Anything LLM Documentation

- Nomic AI Documentation

- HIPAA Overview (health information privacy principles)

- GDPR Overview (data protection and privacy principles)

- Vendor and manufacturer technical manuals (non-sensitive excerpts)