Project Report

**Capstone Project: Regional Cybersecurity Chatbot (UK & MALTA)**

Prantik Chandra

Jayesh Dhere

# INDEX

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Topic** | **Page No.** |
| 1 | Project Title Page | 1 |
| 2 | Introduction | 2 |
| 3 | Background | 3 |
| 4 | Tech Stack | 4-7 |
| 5 | Methodology | 8-9 |
| 6 | API Documentation | 10 |
| 7 | Data Sources | 11-12 |
| 8 | Features and Capabilities | 13-14 |
| 9 | Performance Metrics | 15 |
| 10 | Deployment Guide | 16-20 |
| 11 | Problems Encountered and Solutions | 21-25 |
| 12 | Screenshots | 25-30 |

# INTRODUCTION

Cybersecurity has emerged as a basic need in the current digital ecosystem, where people, businesses, and public sector entities are heavily dependent on interconnected systems and online services. The increasing pace of digital infrastructure development has led to an increased attack surface for cyber threats, and the availability of accessible cybersecurity awareness and information has become a critical need for both technical and non-technical stakeholders. Despite the increasing availability of cybersecurity information on the internet, users continue to face difficulties in understanding technical terms, regional reporting requirements, and taking appropriate actions during incidents.

The Regional Cybersecurity Chatbot – UK is developed as an integrated cyber intelligence system that brings together awareness, visual interaction, and AI-supported knowledge in a single system. Unlike being a standalone conversational system, the system is developed to offer a dashboard-based experience where users can navigate through cybersecurity updates, engage with UK-specific reporting maps, and obtain earthed explanations through a Retrieval-Augmented Generation (RAG) chatbot.

The main aim of the project is to fill the gap that exists between awareness and understanding when it comes to cybersecurity. The project uses the combination of a browser frontend, a FastAPI backend, and a locally running language model using Ollama, which makes the system have a modular and privacy-respecting design that can be used for academic proof-of-concepts, organizational prototypes, and future public-facing cyber support projects. The project uses a UK-focused interface and reporting advice.

# BACKGROUND

The typical traditional cybersecurity systems usually only address a few isolated points of the user needs. News feeds and threat dashboards are mostly about information delivery, while AI chatbots are more about conversation and less about embedding their answers in carefully curated cybersecurity sources. This division of labor leads to a fragmented user experience where a person needs to use multiple systems to stay informed, educated about threats, and find proper reporting channels for the UK’s cybersecurity landscape.

Recent developments in the Retrieval-Augmented Generation (RAG) model have opened up new avenues for improving the trustworthiness of AI-assisted systems. The RAG model’s approach of retrieving information from a structured knowledge base before generating an answer can help minimize the chances of hallucinations and ensure that the generated answer is still consistent with the verified information. The UK Cybersecurity Chatbot implements this by using ChromaDB vector storage, sentence-transformer embeddings, and a locally hosted llama3.2 model via Ollama.

Another important aspect of regional accessibility is the influence it has on the design of the system. The structure of cybersecurity reporting, guidelines, and support organizations differs greatly from country to country. Most cyber awareness systems around the globe disregard this regional aspect, leading to confusion among users seeking help. By incorporating an interactive environment specific to the UK, with maps for visual reporting and references to national guidelines, the system illustrates how localizing can improve clarity and promote active user engagement.

The project also illustrates the increasing need for privacy-friendly AI systems. Instead of solely depending on cloud-based models from other sources, the system conducts language model inference locally and maintains embedded knowledge in a persistent vector database. This approach is more responsible and follows best practices for AI system deployment, especially in cybersecurity, where data sensitivity is of utmost importance.

In summary, the context of this project originates from the requirement to integrate cybersecurity awareness, regional reporting guidelines, and AI-assisted support into a comprehensive UK-centric system. By overcoming the shortcomings of conventional dashboards and generic chatbots, the Regional Cybersecurity Chatbot – UK aims to offer a more trustworthy, interactive, and regionally aware method for delivering cyber intelligence.

# TECH STACK

The UK Cybersecurity Chatbot platform uses a layered and service-oriented architecture that is intended to facilitate grounded AI interaction, regional cybersecurity awareness, and modular deployment. Unlike traditional chatbot systems that are based solely on generative AI models, the platform combines a Retrieval-Augmented Generation (RAG) pipeline with a structured API layer and a lightweight browser-based dashboard.

From an engineering standpoint, the architecture is based on three principles:

1. **Separation of Concerns** - UI, orchestration, retrieval, and generation are separated.
2. **Local-First AI Execution** - Language model inference is done locally using Ollama to reduce data exposure.
3. **Grounded Intelligence Design** - Responses are built from retrieved cybersecurity context rather than free-form generation.

The layered architecture allows for scalability and maintainability while keeping low deployment complexity appropriate for research and prototype settings.

The system is divided into four primary layers that communicate through RESTful interfaces and structured data pipelines.

| **Layer** | **Core Components** | **Responsibilities** | **Design Rationale** |
| --- | --- | --- | --- |
| Presentation Layer | HTML, CSS, JavaScript | Dashboard UI, chatbot UX | Lightweight, framework-free deployment |
| API Layer | FastAPI, Uvicorn | Request routing, orchestration | Async processing and modular endpoints |
| Retrieval Layer | LangChain + ChromaDB | Semantic search and context selection | Grounded answer generation |
| Generation Layer | Ollama + llama3.2 | Local LLM inference | Privacy-focused AI runtime |

FRONTEND SYSTEM DESIGN

The presentation layer is implemented using vanilla web tech in order to minimize runtime overhead and improve portability. Unlike modern SPA frameworks that introduce build pipelines and dependency complexity, the dashboard is implemented using direct DOM manipulation and asynchronous API calls.

The key engineering considerations include the following:

1. **Stateless UI interaction**, where chatbot messages are processed dynamically without the need for persistent frontend storage.
2. **SVG map rendering**, which allows region selection without the need for additional mapping libraries.
3. **Thematic styling logic,** which allows for dark and light modes using CSS variables.

| **Module** | **Function** | **Implementation Detail** |
| --- | --- | --- |
| Chat Interface | Message rendering | Event-driven JS handlers |
| Threat Map | Visual simulation | Canvas/SVG animations |
| Reporting Map | Regional selection | Inline SVG interaction |
| Article Feed | News rendering | Async fetch from /api/articles |

Backend Architecture and API Orchestration

The backend serves as the orchestrator that handles data flow between the frontend, vector database, and the LLM runtime environment.

The choice of using FastAPI was informed by the following factors:

1. Asynchronous request handling
2. Lightweight deployment
3. On the internal architecture of the backend, the following tasks are performed:
4. Query preprocessing and validation
5. Retrieval of context from ChromaDB
6. Construction of the prompt
7. Formatting of the response and aggregation of sources

| **Stage** | **Description** | **Tool** |
| --- | --- | --- |
| Request Intake | Receive chatbot query | FastAPI |
| Retrieval Execution | Semantic similarity search | LangChain Retriever |
| Prompt Assembly | Inject context into template | Python Prompt Builder |
| Response Generation | LLM inference | Ollama |
| Output Formatting | Source deduplication | Backend Logic |

RETRIEVAL LAYER AND VECTOR ARCHITECHTURE

The retrieval layer is the intelligence brain of the system. Rather than querying the raw documents, cybersecurity content is mapped into dense vector embeddings using the all-MiniLM-L6-v2 model.

The design choices that affect the performance of the retrieval layer are as follows:

1. Setting n\_results to a small number to minimize token overhead.
2. Storing the vectors locally to prevent redundant computation of embeddings.
3. Preserving metadata fields for tracing purposes.

| **Field** | **Description** |
| --- | --- |
| Document Chunk | Text segment from UK cybersecurity source |
| Embedding Vector | Semantic representation |
| Metadata | Source URL, title, category |
| ID | Unique chunk identifier |

GENERATION LAYER AND LOCAL MODEL RUNTIME

The generation layer uses Ollama to execute the llama3.2 language model locally. This choice brings the following benefits over cloud-based inference:

1. Lower latency due to local inference
2. No cost for using an external API
3. Better privacy for cybersecurity search queries

The prompt engineering strategy imposes constraints to promote consistency. For instance:

1. Temperature settings are kept low to minimize randomness
2. The model is asked to not make up statistics
3. UK-specific reporting advice is given priority if applicable.

| **Parameter** | **Value** | **Purpose** |
| --- | --- | --- |
| Model | llama3.2 | Local generation |
| Temperature | 0.0 | Deterministic output |
| Context Limit | Truncated | Prevent token overflow |
| Retrieval Depth | 2 chunks | Maintain latency balance |

Architectural Design Trade-Offs

There were a number of engineering trade-offs that went into the design of the final system architecture:

1. **Local inference vs cloud scalability** - Local inference is more privacy-friendly but not horizontally scalable.
2. **Vanilla frontend vs framework abstraction** - Easier to deploy but requires manual implementation of UI logic.
3. **Small retrieval depth vs answer completeness** - Faster response times but less contextually complete answers.

These trade-offs represent the project’s emphasis on controllability and educational transparency over enterprise optimization.

# METHODOLOGY

The development methodology is based on a Retrieval-Augmented Generation (RAG) model, where the cybersecurity knowledge is converted into embeddings before runtime interaction. The methodology does not rely on web scraping or the knowledge of the pretrained models but uses a controlled ingestion pipeline to ensure consistency and predictability.

The development methodology is based on five major steps: data acquisition, preprocessing, embedding, retrieval, and generation.

Data Acquisition Strategy

Cybersecurity content is collected from curated UK governmental and advisory sources. The acquisition process emphasises reliability and source authority to reduce misinformation risk within generated responses.

| **Source Type** | **Example Providers** | **Content Role** |
| --- | --- | --- |
| Government Guidance | GOV.UK | Policy and reporting information |
| National Security Advisory | NCSC | Technical cybersecurity advice |
| Fraud Reporting | Action Fraud | Incident response guidance |
| Regulatory Information | ICO | Data protection awareness |

Knowledge Preparation Pipeline

The ingestion process converts raw cybersecurity documents into semantically searchable units. Key stages include:

* Token-aware text splitting
* Metadata tagging
* Embedding generation
* Persistent storage in ChromaDB

| **Operation** | **Input** | **Output** |
| --- | --- | --- |
| Text Extraction | HTML/PDF | Cleaned Text |
| Chunking | Raw Text | Context Segments |
| Embedding | Text Segments | Dense Vectors |
| Indexing | Vectors | ChromaDB Collection |

Update Monitoring Workflow

A background update checker monitors configured cybersecurity pages and detects content changes through hashing mechanisms. When new or modified content is detected, the system flags datasets for re-ingestion.

| **Step** | **Method** | **Purpose** |
| --- | --- | --- |
| Fetch Page | HTTP Request | Retrieve latest content |
| Generate Hash | SHA-based checksum | Detect changes |
| Compare State | Stored vs New Hash | Identify updates |
| Trigger Action | Flag ingestion | Maintain freshness |

This methodology ensures that the chatbot knowledge base remains aligned with evolving cybersecurity guidance.

# API DOCUMENTATION

The API layer provides structured communication between the frontend dashboard and backend intelligence services. FastAPI enables asynchronous processing, allowing the system to manage chatbot interactions alongside update monitoring and article delivery.

Endpoint Architecture

| **Endpoint** | **Method** | **Payload** | **Description** |
| --- | --- | --- | --- |
| / | GET | None | Service availability check |
| /api/health | GET | None | System diagnostics |
| /api/query | POST | Query JSON | RAG chatbot interaction |
| /api/articles | GET | Limit param | Article retrieval |
| /api/updates | GET | Limit param | Source monitoring |

Query Payload Structure

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| query | String | User question |
| region | String | UK context parameter |

Key engineering decisions include:

* Stateless request handling to improve scalability.
* Explicit health endpoints for system observability.
* Separation of update monitoring from chatbot endpoints to avoid blocking operations.

# DATA SOURCES

The UK Cybersecurity Chatbot depends entirely on the quality and organization of its knowledge sources. Unlike open-domain chatbots, the system uses a carefully curated ingestion approach that targets reputable UK cybersecurity sources. This limits the potential for misinformation while enhancing contextual relevance during the Retrieval-Augmented Generation (RAG) processing stage.

The data is harvested using carefully organized scraping pipelines and stored locally in JSON datasets before being embedded. Each dataset is linked to metadata variables that retain traceability, such as URLs, document types, and publication context.

Source Selection Criteria

1. The sources were chosen based on three engineering constraints:
2. Authority – This refers to official UK cybersecurity or regulatory sources.
3. Consistency – This refers to well-structured content amenable to automated ingestion.
4. Actionability – This refers to guidance related to reporting, awareness, or cyber hygiene.

| **Organisation** | **Role** | **Content Type** |
| --- | --- | --- |
| National Cyber Security Centre (NCSC) | Technical advisory authority | Threat guidance, best practices |
| Action Fraud | National reporting centre | Fraud reporting instructions |
| Information Commissioner’s Office (ICO) | Regulatory authority | Data protection policies |
| GOV.UK Cyber Security Pages | Government information hub | Public awareness resources |

Data Storage Structure

| **Storage Element** | **Location** | **Purpose** |
| --- | --- | --- |
| Scraped JSON Files | Scraped files/ | Raw knowledge base |
| Primary Dataset | cyber\_chatbot\_UK1.json | Backend default source |
| Vector Database | UK/backend/chroma\_db | Semantic retrieval index |

# SYSTEM FEATURES AND CAPABILITIES

The UK Cybersecurity Chatbot integrates multiple cybersecurity assistance functions into a unified environment. The design philosophy prioritises user accessibility while maintaining engineering modularity between visualisation, retrieval, and AI generation components.

**A. Functional Overview**

Key system capabilities include:

* Retrieval-grounded AI chatbot
* Interactive UK reporting-centre map
* Cyber threat visualisation dashboard
* Curated cybersecurity article feed
* Dark/light interface themes

| **Feature** | **Presentation Layer** | **API Layer** | **Retrieval Layer** | **Generation Layer** |
| --- | --- | --- | --- | --- |
| AI Chatbot | ✓ | ✓ | ✓ | ✓ |
| Threat Map Visualisation | ✓ | — | — | — |
| Reporting Centre Map | ✓ | — | — | — |
| Article Feed | ✓ | ✓ | — | — |
| Update Monitoring | — | ✓ | — | — |

**B. Engineering Characteristics**

The system emphasises:

* **Grounded intelligence** through contextual retrieval.
* **Regional relevance** via UK-focused visual components.
* **Minimal frontend dependencies** to support lightweight deployment.

Threat visualisation is intentionally implemented as simulated activity rather than real-time telemetry, reducing infrastructure complexity while preserving educational value.

# PERFORMANCE METRICS

The current implementation prioritises functional reliability and architectural validation over large-scale benchmarking. However, several performance considerations influence system behaviour, including retrieval depth, prompt size, and local inference latency.

1. Dashboard Metrics

| **Metric** | **Value** | **Context** |
| --- | --- | --- |
| UK Cyber Attacks | 7,780,000 | Awareness indicator |
| Fraud Losses Prevented | £2.1B | Economic impact |
| Major Incidents | 204 | NCSC reference |
| Businesses Affected | 43% | Risk representation |

1. Recommended Evaluation Metrics

Future evaluation should include structured measurement frameworks aligned with AI system research practices.

| **Metric Category** | **Measurement Goal** |
| --- | --- |
| API Latency | Measure /api/query response time |
| Retrieval Quality | Precision and recall of vector search |
| Answer Faithfulness | Alignment between response and sources |
| Update Freshness | Time between source change and ingestion |

# DEPLOYMENT GUIDE

The deployment process for the UK Cybersecurity Chatbot follows a structured local-first architecture designed to ensure reproducibility, minimal infrastructure dependency, and privacy-preserving AI execution. Unlike cloud-based chatbot deployments, the system relies on a local runtime environment consisting of Python services, a vector database, and an Ollama-hosted language model.

The installation workflow is divided into repository preparation, environment configuration, dependency installation, model provisioning, and knowledge ingestion.

**1) Repository Initialization**

Deployment begins by cloning the project repository and navigating to the root directory. Version-controlled installation ensures consistent directory structure and predictable runtime paths.

| **Step** | **Command** | **Purpose** |
| --- | --- | --- |
| Clone Repository | git clone <repo-url> | Obtain source code |
| Enter Directory | cd Regional\_Cybersec\_Chatbot\_UK\_Malta | Prepare environment |

**2) Python Virtual Environment Configuration**

A dedicated Python virtual environment isolates dependencies from the host system and prevents version conflicts. This approach aligns with best practices for reproducible AI system deployment.

| **Platform** | **Command** |
| --- | --- |
| Windows PowerShell | .venv\Scripts\Activate.ps1 |
| Windows CMD | .venv\Scripts\activate.bat |
| macOS/Linux | source .venv/bin/activate |

**3) Backend Dependency Installation**

Backend dependencies are installed from the requirements.txt file located within the UK backend directory. These packages support API orchestration, retrieval pipelines, embedding generation, and document parsing.

| **Package** | **Role in System** |
| --- | --- |
| FastAPI | REST API framework |
| Uvicorn | ASGI server runtime |
| ChromaDB | Vector database |
| Sentence-Transformers | Embedding generation |
| LangChain | RAG orchestration |
| LangChain-Ollama | LLM integration layer |
| BeautifulSoup4 | HTML parsing |
| pdfplumber / PyPDF2 | PDF text extraction |
| aiofiles | Async file operations |

Dependencies are installed using:

pip install -r requirements.txt

**4) Local LLM Runtime Configuration (Ollama)**

The system relies on a locally hosted language model to maintain privacy and reduce reliance on external APIs. Ollama acts as the runtime interface for executing the llama3.2 model.

| **Step** | **Command** | **Description** |
| --- | --- | --- |
| Install Ollama | Download from official site | Runtime installation |
| Pull Model | ollama pull llama3.2 | Retrieve model locally |
| Verify Runtime | ollama list | Confirm model availability |

By default, Ollama runs at:

<http://localhost:11434>

**5) Vector Database Ingestion**

Before runtime interaction, cybersecurity datasets must be embedded and indexed into the ChromaDB vector store. The ingestion process transforms structured JSON knowledge into semantic embeddings using the all-MiniLM-L6-v2 model.

| **Input** | **Process** | **Output** |
| --- | --- | --- |
| JSON Dataset | Chunking + Embedding | Vector Index |
| Metadata | Tagging | Source Traceability |

Execution command:

python ingest.py

Resulting vectors are stored in:

UK/backend/chroma\_db/

**Application Startup Procedures**

Once installation and ingestion are complete, the application can be launched using either automated or manual startup workflows.

**1) Automated Startup (Windows Batch Execution)**

A preconfigured batch script simplifies deployment by orchestrating backend initialization and frontend launch.

| **Step** | **Action** |
| --- | --- |
| Launch Backend | Starts FastAPI server on port 8001 |
| Initialization Delay | Allows RAG pipeline loading |
| Frontend Launch | Opens UK dashboard in browser |

**2) Manual Startup Workflow**

Manual startup provides greater visibility into system logs and runtime status.

| **Step** | **Command** |
| --- | --- |
| Start Backend | python main.py |
| Access Frontend | Open UK/index.html |

**Deployment Verification and Health Validation**

Following startup, several validation steps confirm that each subsystem is operational.

| **Component** | **Verification Method** | **Expected Result** |
| --- | --- | --- |
| Backend Service | Visit / endpoint | Status JSON returned |
| API Documentation | /docs endpoint | Swagger UI visible |
| RAG Availability | /api/health | rag\_available = true |
| Dashboard UI | Open index.html | Stats rendered |
| Chatbot | Submit query | Generated response |
| UK Map | Click region | Interactive feedback |

The deployment architecture prioritises simplicity and reproducibility over distributed scalability. Key design motivations include:

* Eliminating external API dependencies through local inference.
* Allowing offline operation after ingestion.
* Supporting educational demonstration environments without container orchestration.

Future production deployments may introduce containerisation, authentication layers, and stricter CORS policies to enhance security and scalability.

# PROBLEMS ENCOUNTERED AND SOLUTIONS

**Data Acquisition and Scraping Challenges**

During the development of the UK Cybersecurity Chatbot, several technical challenges emerged within the data acquisition pipeline. These challenges were primarily associated with automated web scraping, anti-bot detection mechanisms, and content filtering constraints. Since the platform relies on curated cybersecurity sources for Retrieval-Augmented Generation (RAG), ensuring reliable and ethical scraping behaviour became a critical engineering consideration.

The following subsections outline the primary issues encountered, their underlying causes, and the strategies adopted to mitigate their impact on system functionality.

**A. Cloudflare and Anti-Bot Protection Mechanisms**

One of the most significant challenges involved websites protected by Cloudflare or similar traffic filtering services. These platforms employ behavioural analysis, JavaScript challenges, and request fingerprinting to detect automated crawlers.

When the scraper attempted to retrieve content from certain cybersecurity portals, requests were either blocked or redirected to verification pages, preventing the extraction pipeline from accessing the desired data.

**Engineering Impact**

* Incomplete dataset ingestion.
* Increased request latency due to repeated retries.
* Potential IP-based rate limiting.

| **Issue** | **Root Cause** | **Observed Behaviour** |
| --- | --- | --- |
| Access Denied Responses | Anti-bot filtering | HTTP 403/503 status codes |
| Verification Pages | JS challenge detection | Non-parsable HTML returned |
| Rate Limiting | Request pattern analysis | Temporary connection blocks |

**Mitigation Approach**

* Restrict scraping to publicly accessible UK government sources.
* Implement controlled request intervals to reduce detection probability.
* Avoid bypassing security measures to maintain ethical scraping practices.

**B. Immediate Bot Detection by Target Websites**

Several cybersecurity portals identified crawler activity shortly after initial requests. This behaviour was likely triggered by repeated requests, predictable request headers, or absence of browser-like interaction patterns.

**Engineering Impact**

* Early termination of scraping sessions.
* Reduced coverage of desired datasets.
* Increased maintenance overhead for data acquisition scripts.

| **Detection Trigger** | **Technical Explanation** |
| --- | --- |
| Rapid Sequential Requests | High-frequency HTTP calls flagged as automated behaviour |
| Static User-Agent Headers | Lack of variability in request signatures |
| Missing Browser Context | No JavaScript execution environment |

**Mitigation Approach**

* Introduced request throttling and asynchronous delays.
* Reduced scraping scope to essential text-based pages.
* Prioritised structured sources with predictable layouts

1. **Unintended Media Scraping (Images, Videos, and Non-Text Assets)**

Initial scraping attempts resulted in crawlers attempting to download multimedia content, including images, videos, and embedded media elements. This behaviour increased bandwidth usage and introduced unnecessary processing overhead, while providing minimal benefit to the RAG knowledge base.

**Engineering Impact**

* Slower ingestion times.
* Increased storage consumption.
* Noise introduced into parsing pipeline.

| **Content Type** | **Issue Introduced** |
| --- | --- |
| Images | Irrelevant for semantic retrieval |
| Videos | Large file sizes impacting performance |
| Embedded Media | Parsing errors during extraction |

**Mitigation Approach**

* Modified scraping logic to target text nodes only.
* Implemented HTML filtering to ignore <img>, <video>, and media containers.
* Focused ingestion pipeline on structured textual guidance.

**D. Malta-Specific Source Acquisition Limitations**

Although the project initially aimed to include region-specific datasets beyond the UK, several Malta-based cybersecurity websites either restricted automated access or presented structural inconsistencies that prevented reliable scraping.

**Engineering Impact**

* Incomplete dataset collection for Malta region.
* Increased preprocessing complexity due to inconsistent page formats.
* Reduced reliability of automated ingestion workflows.

| **Challenge** | **Description** |
| --- | --- |
| Access Restrictions | Automated requests blocked or limited |
| Structural Variability | HTML layouts inconsistent across pages |
| Content Availability | Limited machine-readable cybersecurity guidance |

**Mitigation Approach**

* Prioritised UK-focused datasets for stable system operation.
* Documented unsupported sources for potential future manual ingestion.
* Maintained modular ingestion pipeline to allow future expansion.

## **Scraping Restrictions and Access Limitations**

Certain Malta government and safety portals employ security mechanisms designed to limit automated crawling. These protections include request filtering, behavioural detection, and dynamic page rendering, which interfered with the scraper’s ability to extract consistent text content.

| **Website** | **Issue Observed** | **Technical Cause** |
| --- | --- | --- |
| <https://mitasupport.gov.mt/> | Partial content retrieval | Navigation-based dynamic loading |
| [https://www.besmartonline.info](https://www.besmartonline.info/) | Requests blocked | Bot-detection filtering |
| <https://childwebalert.gov.mt/> | Access limitations | Security protections on reporting pages |

# SCREENSHOTS



     



