AI RAG System

Getting Started Guide

Install and run the AI RAG System on any devicewith a single command. No setup required.

Alexandru Stefanescu

14 February 2026

Nufringen, Germany

Table of Contents

1. What is the AI RAG System?

The AI RAG System is a local Retrieval-Augmented Generation application. It lets you ingest your own documents (PDF, TXT, Markdown) and then ask questions — answers are grounded in your data with source citations.

The system runs entirely on your machine using Ollama for the LLM and ChromaDB for vector storage. No data leaves your environment.

How it works

1. You upload documents (PDF, TXT, or Markdown files)2. Documents are split into chunks and stored as vectors3. When you ask a question, the system finds the most relevant chunks4. The LLM generates an answer using only those relevant chunks5. Sources are cited so you can verify the answer

Architecture

```
Documents -> Chunker -> ChromaDB (vectors)

|

User Query -> Retrieve top chunks -> Ollama LLM -> Answer + Sources
```

Source code and Docker image

Docker Hub: https://hub.docker.com/repository/docker/alexandrustefanescu/ai-rag-system/general

GitHub: https://github.com/alexandrustefanescu/ai-rag-system

2. Install

Run this single command on your Raspberry Pi OS Lite server. Everything is installed automatically, including Docker if needed.

```
curl -fsSL https://alexandrustefanescu.github.io/ai-rag-system/install.sh | bash
```

That's it. The installer will: - Install Docker and Docker Compose (if not already installed) - Check available disk space and memory - Create the project directory at ~/ai-rag-system - Pull and start the application containers - Download the AI models (gemma3:1b and llama3.2:1b) in the background

Warning: The first run takes a few minutes because the AI models (~1-2 GB) need to be downloaded. Subsequent starts are fast.

Once complete, open your browser:

```
https://localhost:8443
```

Tip: The app uses a self-signed SSL certificate. Your browser will show a security warning — click 'Advanced' then 'Proceed' to continue. This is expected.

System requirements

	Minimum	Recommended
RAM	4 GB	8 GB
Disk space	4 GB free	8 GB free
OS	Raspberry Pi OS Lite (ARM64)	Raspberry Pi OS Lite (ARM64, 64-bit)

3. Add Your Documents

Option A: Through the web interface

1. Open https://localhost:84432. Click the upload area or drag-and-drop your files3. Documents are automatically chunked and indexed

Option B: Copy files to the documents folder

```
cp ~/my-notes.pdf ~/ai-rag-system/documents/

cp ~/report.txt ~/ai-rag-system/documents/
```

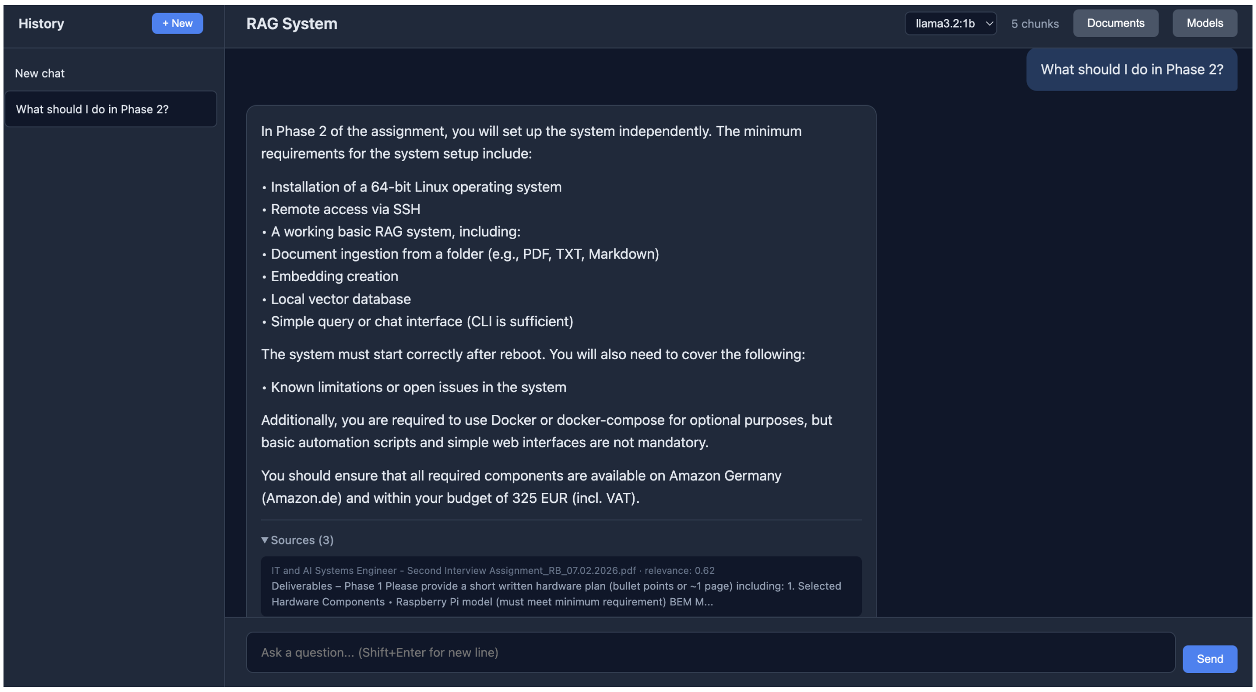
Then trigger ingestion:

```
curl -k -X POST https://localhost:8443/api/v1/ingest
```

Supported file formats

Format	Extension
Plain text	.txt
PDF	.pdf
Markdown	.md

4. Using the Web Interface



What should I do in Phase 2?

In Phase 2 of the assignment, you will set up the system independently. The minimum requirements for the system setup include:

• Installation of a 64-bit Linux operating system
• Remote access via SSH
• A working basic RAG system, including:
• Document ingestion from a folder (e.g., PDF, TXT, Markdown)
• Embedding creation
• Local vector database
• Simple query or chat interface (CLI is sufficient)

The system must start correctly after reboot. You will also need to cover the following:

• Known limitations or open issues in the system

Additionally, you are required to use Docker or docker-compose for optional purposes, but basic automation scripts and simple web interfaces are not mandatory.

You should ensure that all required components are available on Amazon Germany (Amazon.de) and within your budget of 325 EUR (incl. VAT).

▾Sources (3)

IT and AI Systems Engineer - Second Interview Assignment_RB_07.02.2026.pdf · relevance: 0.62
Deliverables – Phase 1 Please provide a short written hardware plan (bullet points or ~1 page) including: 1. Selected Hardware Components • Raspberry Pi model (must meet minimum requirement) BEM M...

Ask a question... (Shift+Enter for new line)

Figure 1: AI RAG System — web interface dashboard

Chat

1. Type your question in the text box at the bottom2. Press Enter or click Send3. The AI answers based on your uploaded documents4. Each answer includes expandable source citations with relevance scores

Model selection

Use the dropdown in the header to switch between available models (e.g., gemma3:1b, llama3.2:1b).

Manage documents

The side panel shows all indexed documents with file name, size, chunk count, and a delete button.

5. Using the API

Interactive API docs are available at https://localhost:8443/api/docs

Method	Endpoint	Description

GET	/api/v1/health	Health check
GET	/api/v1/status	System status
POST	/api/v1/ask	Ask a question
POST	/api/v1/upload	Upload documents
POST	/api/v1/ingest	Re-ingest documents folder
GET	/api/v1/documents	List indexed documents
DELETE	/api/v1/documents/{name}	Delete a document

Ask a question

```
curl -k -X POST https://localhost:8443/api/v1/ask \

-H "Content-Type: application/json" \

-d '{"question": "What is machine learning?"}'
```

Upload a document

```
curl -k -X POST https://localhost:8443/api/v1/upload \

-F "files=@my_document.pdf"
```

Tip: The -k flag tells curl to accept the self-signed certificate.

6. Configuration

To customize settings, edit ~/ai-rag-system/docker-compose.yml and add environment variables under the rag service, then restart:

```
cd ~/ai-rag-system && docker compose restart rag
```

Variable	Default	Description
LLM_MODEL	gemma3:1b	Default LLM model
LLM_AVAILABLE_MODELS	gemma3:1b, llama3.2:1b	Models in dropdown
LLM_TEMPERATURE	0.3	Generation temperature
LLM_MAX_TOKENS	512	Max response tokens
CHUNK_SIZE	500	Characters per chunk
CHUNK_OVERLAP	100	Overlap between chunks
VS_QUERY_RESULTS	5	Candidates per query

7. Troubleshooting

Browser shows security warning

This is expected (self-signed certificate). Click 'Advanced' then 'Proceed to localhost'.

Models are still downloading

Run: cd ~/ai-rag-system && docker compose logs ollama-pull

App is not responding

Check if containers are running: cd ~/ai-rag-system && docker compose ps

No answer or empty response

Make sure you have documents uploaded. Check the document panel in the web UI.

Port already in use

Edit ~/ai-rag-system/docker-compose.yml, change 8443:8443 to 9443:8443, then restart.

Useful commands

```
cd ~/ai-rag-system

# View logs

docker compose logs -f

# Restart

docker compose restart

# Stop

docker compose down

# Uninstall (removes all data)

docker compose down -v
```

Warning: 'docker compose down -v' deletes all downloaded models and indexed documents. Only use for a clean reset.

8. Uninstall

To fully remove the AI RAG System from your machine, run the uninstall script. It guides you through each step interactively and never deletes anything without asking first.

```
bash -c "$(curl -fsSL
https://raw.githubusercontent.com/alexandrustefanescu/ai-rag-system/main/uninstall.sh)"
```

What the uninstaller does

The script runs in four stages, each with an interactive prompt:

- Stops and removes Docker containers (rag-app, rag-ollama).

```
Removes Docker volumes — chroma_data (indexed documents) and ollama_data (downloaded AI models).
```

- Optionally removes the Docker images (ollama/ollama, alexandrustefanescu/ai-rag-system). You are asked before any image is deleted.
- Optionally removes the install directory (~/ai-rag-system and all its contents including certs and documents). You are asked before deletion.

Warning: Removing the volumes (step 2) permanently deletes all indexed documents and downloaded AI models. Back up ~/ai-rag-system/documents/ before uninstalling if you want to keep your files.

Custom install directory

If you installed to a custom directory, set RAG_INSTALL_DIR before running:

```
RAG_INSTALL_DIR=/custom/path bash -c "$(curl -fsSL ...)"
```

Manual removal (alternative)

If you prefer to remove everything manually without the script:

```
docker rm -f rag-app rag-ollama

docker volume rm ai-rag-system_chroma_data ai-rag-system_ollama_data

docker rmi alexandrustefanescu/ai-rag-system:latest ollama/ollama:latest

rm -rf ~/ai-rag-system
```

9. Known Limitations

The AI RAG System is optimised for local, resource-constrained hardware. Understanding its boundaries helps set the right expectations.

Model size — 1 B to 3 B parameters only

The system is designed for small language models in the 1 B–3 B parameter range (e.g. gemma3:1b, llama3.2:1b, qwen2.5:3b). Larger models require more RAM than a typical Raspberry Pi or low-end server provides.

- A 7 B model needs ~6–8 GB of free RAM; a 13 B model needs ~12 GB.
- Running a model that exceeds available memory causes the Ollama container to crash or swap heavily, making responses very slow or impossible.
- Stick to models labelled 1b or 3b for reliable performance on 4–8 GB devices.

Retrieval quality degrades with large document collections

ChromaDB retrieves the top-k most similar chunks for every query. As the document collection grows, retrieval becomes less precise:

- With hundreds of documents the vector space becomes dense — unrelated chunks can score similarly to relevant ones.

- The LLM receives a fixed-size context window. If too many chunks are retrieved, the most relevant content may be diluted or cut off.
- Best results are achieved with focused collections (10–50 documents on a specific topic) rather than a broad knowledge base.

Context window and response length

Each model has a fixed context window (typically 4 096–8 192 tokens for 1 B–3 B models). The system fills this window with retrieved chunks before passing the question to the model.

- Very long questions or many retrieved chunks leave less room for the answer.
- The default LLM_MAX_TOKENS is 512, which limits response length. Increase it in docker-compose.yml if longer answers are needed.
- Documents longer than CHUNK_SIZE characters are split; very short chunks may lose context.

Language support

Supported languages depend entirely on the chosen model. Most 1 B–3 B models are primarily English-trained.

- Multilingual models (e.g. qwen2.5) handle other languages better but may still struggle with mixed-language documents.
- Embedding quality for non-English text varies — retrieval may be less accurate.

Supported file formats

Only PDF, plain text (.txt), and Markdown (.md) files are supported. Other formats (DOCX, XLSX, HTML, images with text) are not ingested.

No persistent conversation history

Each question is answered independently. The system does not maintain a multi-turn conversation — it has no memory of previous questions in the same session.

CPU inference only (no GPU acceleration)

The Docker setup uses CPU inference by default. Responses from 1 B models take 5–30 seconds on a Raspberry Pi 5; 3 B models may take 30–120 seconds. GPU passthrough to Docker requires additional host configuration not covered by the installer.

Self-signed TLS certificate

The installer generates a self-signed certificate. Browsers will show a security warning on first access. This is expected for local use but means other devices on the network will also need to accept the warning or trust the CA certificate manually.

Potential errors and how to resolve them

- Out of memory: Container crashes on startup. Usually caused by insufficient RAM. Reduce LLM_AVAILABLE_MODELS to only 1b models and restart.
- Port conflict: Startup fails because port 8443 or 11434 is already in use. Change the host port in docker-compose.yml (e.g. 9443:8443).
- PDF ingestion failure: Uploading a corrupt or password-protected PDF causes ingestion to silently skip the file. Check container logs: docker compose logs rag.

- Model not yet loaded: If the Ollama container is not yet healthy when the app starts, the app retries until it is available. This can add up to 2 minutes on first boot.
- Embedding model OOM: The semantic chunker requires ~400 MB of RAM for the embedding model. On very low-memory devices, set CHUNK_STRATEGY=fixed in docker-compose.yml.
- Data loss on volume removal: All indexed documents and downloaded models are stored in Docker volumes. Running docker compose down -v deletes them permanently.

10. Why a Self-Developed System?

When evaluating local RAG solutions, several established open-source projects were considered (AnythingLLM, Open WebUI, LocalGPT, and others). We chose to build and maintain our own system for the following reasons.

Security transparency and full control

We know exactly what the code does because we wrote every line of it. Every dependency, every API call, and every file access is visible and auditable.

- Open-source third-party projects can include telemetry, analytics calls, or update checks that contact external servers — even unintentionally. Our system makes no outbound network calls beyond Docker image pulls and Ollama model downloads.
- When a security vulnerability is discovered (e.g. a path traversal in the upload handler, or an injection risk in a query), we patch it immediately without waiting for an upstream maintainer or a community release cycle.
- We control the entire dependency tree. Compromised or abandoned upstream packages are caught in our own audits rather than inherited silently through a transitive dependency.

No vendor lock-in or external dependencies at runtime

The system has zero runtime dependencies on external services. Everything — the LLM, the vector database, the web interface — runs in containers on your own hardware.

- No API keys required. No OpenAI, no Anthropic, no cloud.
- No accounts, no subscriptions, no usage limits.
- Works completely offline after the initial image and model download.

Data sovereignty

Your documents never leave your machine. There is no cloud sync, no telemetry endpoint, and no analytics pipeline. This makes the system suitable for sensitive or confidential documents such as internal reports, legal documents, or personal notes.

Tailored for our specific use case

General-purpose RAG platforms make trade-offs to serve a wide range of users. Our system is optimised specifically for low-power hardware (Raspberry Pi, small servers) and focused document collections, without the overhead of features we do not need.

- The installer is a single curl command that works on a fresh device — no Python, no Node.js, no manual configuration required.
- The semantic chunking strategy and retrieval parameters are tuned for the small models we target.
- The TLS setup, port configuration, and volume layout are designed for reliable single-user local deployment.

Rapid patch velocity

Because the codebase is small and fully owned, we can ship security patches and bug fixes within hours of discovery. We are not blocked by community consensus, pull request review queues, or the release cycles of a larger project.

Note: This does not mean open-source alternatives are insecure. It means that for a system specifically designed to handle private documents on private hardware, full code ownership and auditability is the most appropriate security posture for our users.

The decision

The decision to build a custom RAG system rather than adopt an existing open-source solution was not made lightly. Several platforms were evaluated, including AnythingLLM, Open WebUI, LocalGPT, and PrivateGPT. Each was installed, tested, and assessed against the same requirements.

What the alternatives lacked

Every evaluated platform had at least one of the following problems:

- Bloated feature set: Excessive complexity — dashboards, user management, and cloud sync features that are unnecessary for a single-user local deployment and increase the attack surface.
- Opaque network activity: Several solutions make outbound calls at startup (telemetry, version checks, model registries). These calls are not always documented and are difficult to fully disable.
- Slow security response: Large projects move slowly. A known security issue may sit open for weeks before a maintainer merges a fix, and users of pinned versions may never receive it.
- Host dependencies: All evaluated solutions require either a Python environment, Node.js, or both installed and maintained on the host. Our system requires only Docker.
- Data portability: Some solutions store vectors or history in formats that are difficult to inspect, migrate, or back up reliably.

What we gained by building our own

A purpose-built system let us optimise every layer for our specific constraints: low-power ARM hardware, small focused document collections, a single trusted user, and a hard requirement that nothing sensitive leaves the machine under any circumstances.

- Minimal codebase: The entire application fits in under 3 000 lines of Python. Every line is readable and auditable.
- Zero outbound calls at runtime: No analytics, no telemetry, no update pings. The only outbound traffic is the initial docker pull and model download — both under direct user control.
- Full patch control: A security bug found today can be patched, rebuilt, and redeployed within the hour. No waiting for upstream, no forking a large project.
- Clean install and removal: The installer is a single shell script. The uninstaller reverses every change. There are no leftover services, cron jobs, or systemd units.

11. Raspberry Pi Security Hardening

When running the AI RAG System on a Raspberry Pi as a home or office server, the default OS configuration is not hardened for network-exposed services. The following steps describe the security measures applied to our deployment.

SSH — private key authentication only

```
Password-based SSH login is disabled. Only connections authenticated with an SSH private key are
accepted. The following directives are set in /etc/ssh/sshd_config:

PasswordAuthentication no

PermitRootLogin no

ChallengeResponseAuthentication no
```

Apply the changes with:

```
sudo systemctl restart ssh

PasswordAuthentication no: PasswordAuthentication no — disables username/password login entirely;
only key-based auth is accepted.

PermitRootLogin no: PermitRootLogin no — the root account cannot log in over SSH even with a
valid key.

ChallengeResponseAuthentication no: ChallengeResponseAuthentication no — disables
keyboard-interactive auth methods such as OTP prompts, removing another potential brute-force
surface.
```

Static IP address

A static IP address is assigned to the Raspberry Pi so that firewall rules, router port policies, and client bookmarks remain stable across reboots.

- Router DHCP reservation: The DHCP server on the router is configured with a MAC address reservation, always assigning the same IP to the Pi's network interface.
- OS-level static IP: The Pi's /etc/dhcpcd.conf (or /etc/network/interfaces on newer Raspberry Pi OS) is also configured with a static IP as a fallback, ensuring the address is stable even if the router is replaced.

UFW firewall — subnet-restricted port access

Uncomplicated Firewall (UFW) is installed and configured to allow only the three ports required by the system, and only from within the local subnet (192.168.0.0/24). All other inbound traffic is denied by default.

```
# Install UFW

sudo apt-get install -y ufw
```

```
# Default: deny all inbound, allow all outbound

sudo ufw default deny incoming

sudo ufw default allow outgoing
```

```
# Allow only from local subnet

sudo ufw allow from 192.168.0.0/24 to any port 22

sudo ufw allow from 192.168.0.0/24 to any port 8443

sudo ufw allow from 192.168.0.0/24 to any port 11434
sudo ufw enable
```

The resulting firewall ruleset:

```
To Action From

-- ------ ----

22 ALLOW 192.168.0.0/24

8443 ALLOW 192.168.0.0/24

11434 ALLOW 192.168.0.0/24
```

- Port 22 (SSH): Remote administration. Restricted to the local network — no SSH from the internet.
- Port 8443 (RAG web interface): The HTTPS web UI and API. Accessible only from devices on the same subnet.
- Port 11434 (Ollama API): The local LLM API. Restricted to the subnet to prevent external model access.

Note: The Ollama API (port 11434) has no authentication layer of its own. Restricting it to the local subnet via UFW is essential to prevent external callers from querying the LLM without any access control.

fail2ban — brute-force protection

fail2ban monitors system logs and automatically bans IP addresses that show signs of brute-force activity (e.g. repeated failed SSH login attempts).

```
sudo apt-get install -y fail2ban

sudo systemctl enable --now fail2ban
```

The default jail configuration monitors /var/log/auth.log and bans an IP for 10 minutes after 5 failed login attempts within a 10-minute window. This is effective even though password auth is disabled — it also catches malformed key attempts and scanning traffic.

- Check banned IPs: sudo fail2ban-client status sshd
- Unban an IP: sudo fail2ban-client set sshd unbanip <IP>
- View logs: sudo journalctl -u fail2ban -f

Security measures summary

Measure	Tool / Config	Purpose
SSH key-only auth	/etc/ssh/sshd_config	Eliminates password brute-force attacks
Root login disabled	PermitRootLogin no	Removes the highest-privilege attack target
Static IP	DHCP reservation + OS config	Stable addressing for firewall and routing rules
Subnet firewall	UFW (ports 22, 8443, 11434)	Blocks all internet-facing access to services

Brute-force banning	fail2ban	Auto-bans IPs after repeated failed auth attempts

12. Managing the System

All day-to-day management is done with Docker Compose from the install directory. Run these commands on the machine where the system is installed (locally or over SSH).

```
cd ~/ai-rag-system
```

Start

Start all services in the background:

```
docker compose up -d
```

On first start, the RAG app waits for Ollama to pass its health check before accepting connections — this can take up to 2 minutes. Check readiness with:

```
docker compose ps
```

The rag service status shows healthy once the app is ready.

Stop

Stop all containers without removing any data:

```
docker compose down
```

Note: This stops and removes the containers but leaves all volumes intact — your indexed documents and downloaded models are preserved.

Restart

Full restart (e.g. after editing docker-compose.yml):

```
docker compose down && docker compose up -d
```

Restart only the RAG app without touching Ollama:

```
docker compose restart rag
```

View logs

```
# All services, live

docker compose logs -f
```

```
# RAG app only

docker compose logs -f rag
```

```
# Ollama only
```

```
docker compose logs -f ollama
```

Press Ctrl+C to stop following logs.

Check status

```
# Container status and health

docker compose ps
```

```
# CPU and memory usage

docker stats
```

Update to a newer version

Pull the latest image and recreate the container:

```
docker compose pull

docker compose up -d
```

Note: Your data volumes are not touched during an update. The new container starts automatically with docker compose up -d.

Remove completely

There are two options for complete removal.

Option A — uninstall script (recommended): prompts before each destructive step.

```
bash -c "$(curl -fsSL
https://raw.githubusercontent.com/alexandrustefanescu/ai-rag-system/main/uninstall.sh)"
```

Option B — manual removal: run these commands in order.

- Stop containers

```
docker compose down
```

- Remove data volumes (deletes all indexed documents and downloaded AI models)

```
docker volume rm ai-rag-system_chroma_data ai-rag-system_ollama_data
```

- Remove Docker images

```
docker rmi alexandrustefanescu/ai-rag-system:latest ollama/ollama:latest
```

- Remove the install directory

```
rm -rf ~/ai-rag-system
```

Warning: Step 2 permanently deletes all your indexed documents and downloaded AI models. Back up ~/ai-rag-system/documents/ before proceeding if you want to keep those files.

Quick reference

Task	Command
Start	docker compose up -d
Stop (keep data)	docker compose down
Restart all	docker compose down && docker compose up -d
Restart app only	docker compose restart rag
View live logs	docker compose logs -f
Check container status	docker compose ps
Remove everything	bash -c "$(curl -fsSL .../uninstall.sh)"