Vectoria: Supercomputer-driven vector database for private LLM retrieval

User Manual, version 1.1

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

This page intentionally blank

Summary

Table of Figures

Acronyms and Terminology

AI	Artificial Intelligence	Field of study in computer science that develops and studies intelligent machines.
LLM	Large Language Model	A type of AI model trained on vast amounts of text to perform natural language understanding and generation.
RAG	Retrieval-Augmented Generation	A technique that combines information retrieval and generative AI for producing contextually relevant outputs.
CLI	Command-Line Interface	A text-based interface used to interact with software or operating systems.
FAISS	Facebook AI Similarity Search	A library for fast nearest-neighbor search and vector similarity.
API	Application Programming Interface	A set of rules and tools for building and interacting with software applications.
GPU	Graphics Processing Unit	Specialized hardware designed for accelerating computational tasks, especially in AI and ML.
MMR	Maximal Marginal Relevance	A ranking technique that optimizes information relevance and diversity in retrieved results.
RAGAS	Retrieval-Augmented Generation Assessment	A tool or metric for evaluating the performance of RAG-based systems.
HPC	High-Performance Computing	Computing systems with high-speed processing capabilities used for complex computations.
Embedding	-	A vector representation of data (e.g., text) used for similarity searches and machine learning tasks.
Inference	-	The process of generating predictions or outputs from a trained model.

1. Introduction

Within the framework of the European Project EuroCC2, the national competence center EuroCC Italy developed "Vectoria: Supercomputer-driven vector database for private LLM retrieval". This project aims to equip small and medium enterprises with a service to streamline access to private, internal documentation through intuitive chatbot-like interactions. This initiative targets the challenges of navigating unstructured and dispersed knowledge stored across diverse organizational repositories, offering an efficient solution for users seeking concise, contextually relevant answers.

By employing state-of-the-art retrieval and generative AI technologies, the system ensures privacy preservation while delivering high-quality responses. This document aims to outline the requirements, core functionalities, and operational guidance for deploying and utilizing the system effectively.

The service is designed to integrate with High Performance Computing (HPC) systems, leveraging the computational power of supercomputers to enhance the speed and precision of a generative model inference. This approach enables scalable, private, and high-performance information retrieval, tailored to meet the needs of users in a data-rich environment.

2. EuroCC: Objectives

EuroCC 2 is a project which works to identify and address the skills gaps in the European High-Performance Computing (HPC) ecosystem and coordinate cooperation across Europe to ensure a consistent skills base. The mission of EuroCC is to improve the technological readiness of Europe. In particular, the role of EuroCC2 is to establish and run a network of more than 30 NCCs across the EuroHPC Participating States. The NCCs act as single points of access in each country between stakeholders and national and EuroHPC systems. They operate on a regional and national level to liaise with local communities, in particular SMEs, map HPC competencies and facilitate access to European HPC resources for users from the private and public sector.

Figure 1: NCCs across Europe

EuroCC2 delivers training, interacts with industry, develops competence mapping and communication materials and activities, and supports the adoption of HPC services in other related fields, such as quantum computing, artificial intelligence (AI), high performance data analytics (HPDA) to expand the HPC user base.

3. Service Description

The "Vectoria: Supercomputer-driven vector database for private LLM retrieval" project introduces a tailored approach to optimize knowledge access within organizations. By focusing on the integration of advanced AI-driven methods, it enables users to interact seamlessly with internal documentation via chatbot-like interfaces. The project emphasizes creating a unified, efficient framework to navigate unstructured data by delivering concise, contextually relevant answers to users while upholding stringent privacy standards.

Key Objectives:

- **Efficient Knowledge Management**: scattered internal documentation is unified into a knowledge source accessible through querying.
- **High Precision and Relevance**: Deliver results that precisely match user queries by using vector-based retrieval techniques.
- **Data Privacy**: Operate entirely within the organization's infrastructure to ensure compliance with internal security protocols.

4. Architectural Diagram

The RAG pipeline consists of several tightly integrated components, each playing a critical role in ensuring the system's accuracy. Below is a comprehensive breakdown of its architecture and workflow.

System Overview

The architecture comprises the following core modules:

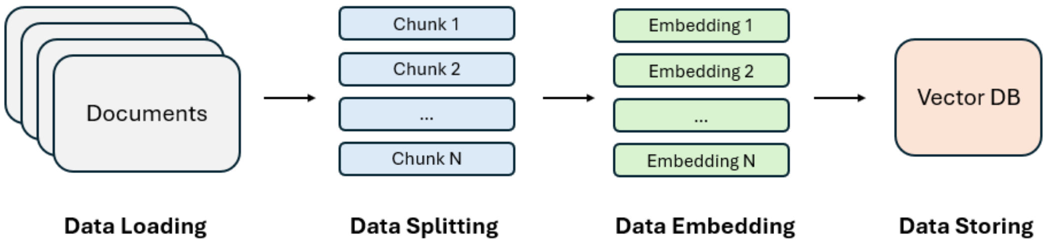
1. **Configuration**: The system can be easily customized through a yaml-based configuration file.
2. **Document Preprocessor**: Handles ingestion, cleaning, and splitting of internal documents into smaller portions of text (*chunks*) and transforming these chunks into embeddings.
3. **Vector Database**: Stores vector embeddings of the document chunks for efficient similarity searches.
4. **Retriever**: Locates relevant chunks from the vector database based on user queries.
5. **Post-retrieval strategies**: Includes re-ranker and full paragraph retriever.
6. **Generative Model**: Synthesizes human-like answers using a pre-trained large language model (e.g., GPT).

4.1 Tasks

The RAG pipeline provides two primary operational tasks:

- **Build vector database**: Responsible for preparing and indexing the documents to create a vector database.
- **Inference**: Handles user queries by retrieving and generating context-aware responses.

4.1.1 Build vector db



This task focuses on the ingestion and indexing of internal documents. It transforms a set of documents into a searchable structure, ensuring that subsequent retrieval operations are both fast and accurate. In technical language (and inside the codebase) this operation is also referred as "Build Index".

Workflow:

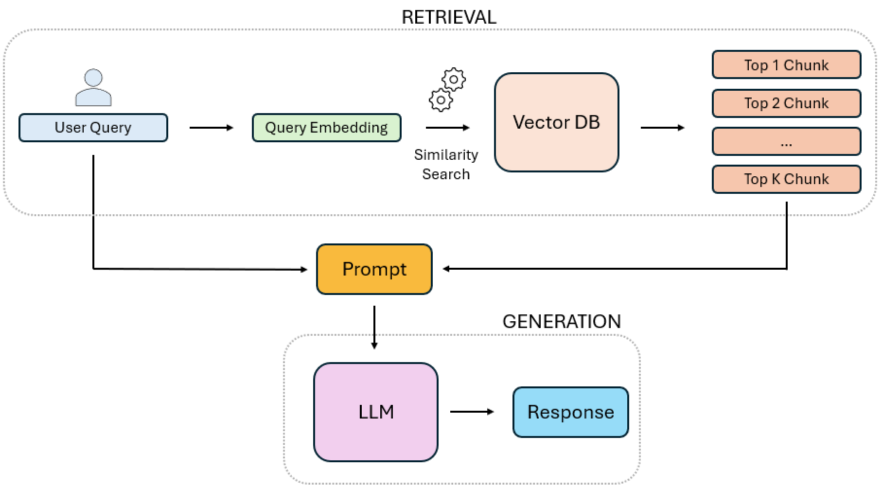
Figure 2: Build vector db workflow

Prerequisites: as version 1.0, supported formats include PDF and DOCX.

1. **Data Preprocessing (Loading and Splitting)**: Clean the text to remove irrelevant elements like boilerplate text, extra whitespace, or special characters. Split documents into chunks of manageable size (e.g., 512 characters) to optimize retrieval granularity. Parameters for cleaning and splitting can be configured using a single configuration file.
2. **Data Embedding**: Generate vector embeddings for each chunk using pre-trained models (e.g., bge-m3). Each embedding is a n-dimensional vector which captures the semantic meaning of the chunk.
3. **Data Storing**: Store the embeddings in a vector database (e.g., FAISS vector database) dumped on disk. Attach metadata (e.g., source file, paragraph number) for efficient filtering and explainability.

4.1.2 Inference

The inference task enables the chatbot to provide accurate answers by combining retrieval and generation. It works exploiting the indexed data to ensure high-quality responses.



Workflow:

Figure 3: Inference workflow

1. **Query Reception**: User submits a question via a CLI or file
2. **Query Embedding**: Convert the user query into a vector embedding using the same pre-trained model used during indexing.
3. **Vector Search**: Search the vector database for the most semantically similar chunks to the query. Retrieve the top-k chunks (e.g., top 5 most relevant) using a similarity metric (e.g., cosine similarity).
4. **Post-Retrieval strategies**: Perform re-ranking or full paragraph retrieving, if necessary.
5. **Prompt Construction**: Combine the *retrieved chunks* with the *question* and the *instructions* to form a context window for the generative model.
6. **Answer Generation**: Pass the query and context to a generative AI model (e.g., llama3.1) to generate a natural language response tailored to answer the user's query.

5. Self-Deployment Guide

The following provides a detailed description of how to use the script provided to deploy the service.

5.1 Prerequisites

The following prerequisites must be met before continuing:

- System requirements:
  - Minimum: **Embedder** (bge-m3) **+ LLM** (Llama-3.1-8B-Instruct)
    - **VRAM:** ~36GB
  - Recommended: **Embedder** (bge-m3) **+ LLM** (Llama-3.1-70B-Instruct)
    - **VRAM:** ~252GB
- The user setting up the service must have a Linux terminal with Python 3.8 or later.
- If installing on a remote host, SSH access is required.
- For some LLMs, you will also need to register your SSH key to access the download. Consult the documentation of your LLM of choice for instructions.
- `git-lfs` is required to download some larger LLMs. Make sure it is installed and available on the install destination.

5.2 Step-by-step Installation & Deploy

To install Vectoria follow the steps described below.

1. Clone the repository containing the Ansible codebase:

```
git clone https://github.com/Eurocc-Italy/Vectoria.git
```

2. Create and activate a Python virtual environment:

```
python3 -m venv virtualenv
source virtualenv/bin/activate
```

3. Upgrade `pip` to the latest version (24.0 or later is required):

```
pip install --upgrade pip
```

4. Move to the Vectoria ansible folder and install the necessary requirements:

```
cd Vectoria/ansible
pip install -r requirements.txt
```

5. In the `inventory/hosts` file, edit the `ansible_host` and `ansible_user` variables with the hostname (or IP address) and the username of the install destination to be used, respectively.

6. If installing locally, edit the `deploy.yml` file, changing the following `hosts` variable from `hpc` to `localhost`.

7. Edit the `roles/vectoria/vars/config.yml` file as necessary. Comments explaining what each variable pertains to are provided in the file itself. Some reasonable defaults are provided. For the inference engine of choice, we recommend the following models based on the scenario:
   - Consumer PC (~8GB vRAM): `Llama-3.2-1B-Instruct`
   - High-end workstation (~32 GB vRAM): `Llama-3.1-8B-Instruct`

   - HPC cluster (~256 GB vRAM): Llama-3.1-70B-Instruct

8. Run the Ansible playbook:

```
ansible-playbook -i inventory deploy.yml
```

9. Alternatively, it is also possible to install only the Vectoria library locally via pip (editable install mode is required), and configure each setting manually:

```
pip install -e vectoria
```

5.3 USAGE

In this section we will show how to use Vectoria to:

- Build the vector database necessary for RAG operations;
- Launch jobs on HPC with queries in inference;
- Use the GUI locally.

5.3.1 Building the vector database

During the deployment phase, a `docs` folder is created at the `install_path` indicated in the configuration file, where users can store the documents to be used for RAG. With this documents folder populated, move to the `install_path/vectoria/bin` directory, where the relevant scripts for all necessary tasks should already be configured according to the specifications provided in the deployment configuration file. The `start_build_index.sbatch` file is the one necessary for building the vector database.

If running on HPC, launch the script via Slurm:

```
sbatch start_build_index.sbatch
```

If running locally, the script can also be ran as a regular bash script without modifications. Before running the script, make sure the environment is set up correctly by sourcing the `setup_vectoria.sh` file:

```
source setup_vectoria.sh
bash start_build_index.sbatch
```

After the job is completed, you should see a new folder at the install_path/vectoria/test/index path containing the index.faiss and index.pkl files.

5.3.2 Launching queries on HPC

After building the vector database it is possible to launch a query on HPC using the provided `start_inference.sbatch` script, found in the same folder as the previous and already configured according to the specifications provided during the deploy. If using a custom configuration file or a different vector database, it is possible to provide those options here at the `CONFIG_FILE_PATH` and `FAISS_INDEX_PATH` variables, respectively.

At the end of the file are the two options available for interacting with Vectoria: i) writing the questions in a JSON file specified in the TEST_SET_PATH variable; ii) writing the questions directly on the command line interface. To use either, simply comment/uncomment the corresponding lines as necessary.

After this, you can launch the script via Slurm if running on HPC:

```
sbatch start_inference.sbatch
```

Or you can run the script locally:

```
bash start_inference.sbatch
```

The answer(s) to the provided question(s) will be present in the `.err` file if ran via Slurm, or directly on terminal if ran locally.

5.3.3 Launching queries locally via the GUI

When running Vectoria locally, it is also possible to use the GUI to interact with the service. First, make sure Streamlit is installed in the virtual environment:

```
source <install_path>/vectoria/bin/setup_vectoria.sh
pip install streamlit
```

Then, move to the `install_path/vectoria/vectoria_lib/gui` folder and run the following command:

```
streamlit run gui_v1.py
```

The terminal should provide two links, one for the local host and one for the local network. If running locally, open the web page linked for the local host. The local network address can be used by other users in the same local network, sending queries to the workstation hosting the service.

On this web page it is also possible to build the vector database, if not done previously. If a vector database was previously built as explained in the previous section, provide its path to the GUI and load it. At this point it is possible to query the local LLM and ask questions to the chatbot.

NOTE: it is also possible to use vector databases built with different LLMs. For example, one use case could see the vector database being built on HPC with a large LLM such as Llama-70B and queried locally with a much smaller LLM such as Llama-1B.

NOTE: by default, the service is configured to provide quite short, concise answers. If a lengthier, more detailed answer is required it is possible to adjust the `max_new_tokens` parameter in the configuration file. If a short answer is satisfactory, however, changing this parameter will **not** make the answer unnecessarily verbose.

5.3.4 Advanced Configuration

This section explains the RAG configuration file in detail. It contains every parameter that can be customized to adapt the behavior of the system to the desired one, covering several components in detail. The default configuration file can be found at the `install_path/vectoria/etc/default/default_config.yaml` path; custom configurations can be provided by setting the correct path in the relevant Slurm scripts.

PARAMETERS	TYPE	VALUES (default in bold)	DESCRIPTION
vectoria_logs_dir:	str	vectoria_logs	PATH where log files will be dumped
log_level:	str	[**DEBUG** | INFO | CRITICAL]	Sets general log level
langchain_tracking:	str	[**true** | false]	Enables LangChain tracking backend
system_prompts_lang:	str	[**eng** | it]	Language for system prompts, e.g., English or Italian
data_ingestion:			

multiprocessing:	str	[true | false]	Enables/disables parallel processing for ingestion tasks
extraction:			
format:	str	[docx | pdf]	Specifies input document format
dump_doc_structure_on_file:	str	[true | false]	Saves document structure to file if true
regexes_for_metadata_extraction:			List of regex patterns for metadata extraction
- name:	str	DOC_ID	Name of the metadata field
pattern:	str	'^Document Title'	Regex pattern to match the field
regexes_for_replacement:			List of regex rules for text preprocessing
- name:	str	remove_multiple_spaces	Rule name
pattern:	str	'[\t]{2,}'	Regex to find multiple spaces/tabs
replace_with:		' '	Replacement value
- name:	str	remove_bullets	Rule name
pattern:	str	'^\s*[\u2022\u25AA\u27A2]\s*'	Regex to match bullets
replace_with:		''	Replace bullets with empty string
- name:	str	remove_ligature_st	Rule name
pattern:	str	'ﬆ'	Regex to match 'ﬆ'
replace_with:	str	'st'	Replace 'ﬆ' with 'st'
chunking:			
chunk_size:	int	[256 | 512 | 1024]	Size of each chunk for processing
chunk_overlap:	int	[128 | 256 | 512]	Overlap between consecutive chunks
separators:	List[str]	["\n\n", "\n", " ", ""]	List of separators for chunking
is_separator_regex:	List[str]	[false, false, false, false]	Whether separators are regex patterns
dump_chunks_on_file:	str	[true | false]	Save chunked output to file if true
vector_store:			
name:	str	faiss	Vector store backend (e.g., FAISS)
model_name:	str	BAAI/bge-m3	Path to the model used for generating embeddings
device:	str	[cuda | cpu]	Device for vector store operations
normalize_embeddings:	str	false	Normalizes embeddings if true
retriever:			
enabled:	str	[true | false]	Enables/disables the retriever

top_k:	int	5	Number of top results to retrieve
search_type:	str	'mmr'	Search type (e.g., Maximal Marginal Relevance)
fetch_k:	int	5	Number of documents to fetch before filtering
lambda_mult:	float	0.5	Lambda parameter for MMR search
reranker:			
enabled:	str	[true | false]	Enables/disables reranking
reranked_top_k:	int	3	Number of reranked results to return
inference_engine:			Inference engine configuration for reranking
name:	str	[huggingface | ollama | openai | vllm]	Name of inference engine backend
url:	str	null	URL for the inference API endpoint (e.g. http://localhost:8898/v1)
api_key:	str	null	API key for the inference engine
model_name:	str	BAAI/bge-reranker-v2-gemma	Path to the model for reranking
device:	str	[cuda | cpu]	Device for inference engine
load_in_4bit:	str	[true | false]	Enables 4-bit model loading for optimization
load_in_8bit:	str	[true | false]	Enables 8-bit model loading for optimization
max_new_tokens:	int	150	Maximum tokens generated in output
trust_remote_code:	str	[true | false]	Trust code from remote repositories if true
device_map:	str	[auto | null]	Device mapping for distributed inference
temperature:	float	0.1	Sampling temperature for generation
full_paragraphs_retriever:			
enabled:	str	[true | false]	Enables/disables retrieval of full paragraphs
inference_engine:			General inference engine configuration
name:	str	[huggingface | ollama | openai | vllm]	Name of the inference engine
url:	str	null	URL for the inference engine API
api_key:	str	null	API key for the inference engine
model_name:	str	meta-llama/Meta-Llama-3.1-8B-Instruct	Path to the model for inference
device:	str	[cuda | cpu]	Device for inference
load_in_8bit:	str	[true | false]	Enables 8-bit model loading for optimization
max_new_tokens:	int	150	Maximum tokens generated in output

trust_remote_code:	str	[true | false]	Trust code from remote repositories if true
device_map:	str	[auto | null]	Automatic device mapping for distributed inference
temperature:	float	0.1	Sampling temperature for generation
evaluation:			Evaluation configuration
tool:	str	ragas	Evaluation tool name
inference_engine:			Inference engine for evaluation
name:	str	vllm	Name of the inference engine backend
model_name:	str	hugging-quants/Meta-Llama-3.1-8B-Instruct-AWQ-INT4	Path to the model for evaluation
url:	str	null	API endpoint for inference engine
api_key:		null	API key for the inference engine
embeddings_engine:			Embedding engine for evaluation
name:	str	vllm	Name of the embedding engine
model_name:	str	BAAI/bge-multilingual-gemma2	Model for embedding generation
url:	str	null	API endpoint for embeddings engine
api_key:	str	null	API key for the embeddings engine

Application information and contributions

This manual, *`version 1.1`*, refers to tag *`v1.0`* of repos:

https://github.com/Eurocc-Italy/Vectoria

The **Vectoria** application was developed by **EuroCC Italy** national competence center,

with the specific contribution of **CINECA** and **Leonardo S.p.A**.

The project was realized thanks to:

Luca Babetto, Leonardo Baroncelli, Carolina Berucci, Chiara Malizia, Eric Pascolo, Andrea Proia

Thanks to SAVIA Team for the technical input (https://assemblealegislativa.github.io/savia/)