Software Requirements Specification

For

Utilizing Hadoop Ecosystem to Implement RAG for Chain of Thought Reasoning in Language Models

Prepared by
[1]Aviral Pandey (21CS10089)
[1]Harsh Sharma (21CS30023)
[1]Ishan Raj (21CS10032)
[1]Prasanna Paithankar (21CS30065)
[1]Utsav Dhanuka (21CS10087)

[1]Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur

Submitted on 20th April 2024
for CS39202 – Database Management Systems Laboratory

Version no.: 2.0

This page has been intentionally left blank

Table of Contents

Introduction 5
- Purpose 5
- Document Conventions 5
- Intended Audience and Reading Suggestions 5
- Product Scope 5
- References 5

Overall Description 6
- Product Perspective 6
- Product Functions 6
- User Classes and Characteristics 6
- Operating Environment 6
- Design and Implementation Constraints 6
- User Documentation 6
- Assumptions and Dependencies 6

External Interface Requirements 7
- User Interfaces 7
- Hardware Interfaces 9
- Software Interfaces 9
- Communication Interfaces 9

System Features 10
- LangChain 10
- Retrieval Augmented Generation 10
- Chain of Thought Reasoning 10
- Large Language Model – Gemini Pro 11
- MapReduce 11
- Information & System Security 11

Solr/Zookeeper Database Modelling 13
- Dataset for RAG 13
- Solr Cloud Collections & Documents 13
- Docker Layout 13

Scalability, Monitoring & Dockerization 14
- Apache HTTP Server 14
- Scalability in terms of concurrent end-users 14
- Docker Containers 14
- Prometheus & Grafana 14
- Web Server Gateway Interface (WSGI) 14

Linking to https://cse.iitkgp.ac.in/ Domain 14

Other Nonfunctional Requirements 15

Performance Requirements 15

Safety and Security Requirements 15

Software Quality Attributes 15

Business Rules 15

Appendix A: Comparision with raw Gemini Pro
Appendix B: Code Maintenance
Appendix C: Glossary
Appendix D: Conflict of Interests

1. Introduction

1.1 Purpose

This SRS specifies the software requirements of the “Utilizing Hadoop Ecosystem to Implement RAG for Chain of Thought Reasoning in Language Models” pipeline, henceforth referred to as SolrChain. This is the first release of the software, and the SRS provides the full product description.

1.2 Document Convention

The document follows the IEEE Std 830-1998 standards which lays recommended practice for Software Requirements Specifications.

1.3 Intended Audience and Reading Suggestions

The SRS has been targeted towards the development and maintenance community for active suggestions and prototyping scope. The readers are advised to jump to their target of interest after navigating through the Table of Contents. For the readers wanting to have basic description of the software, we recommend them to go through Section 2 and 4.

1.4 Production Scope

The software and subordinates have not been envisioned to be deployed commercially by the authorships. No formal production and release will be carried out. Readers interested in usage are supposed to contact the authors and developers involved.

1.5 References

[1] Jason Wei and Xuezhi Wang and Dale Schuurmans and Maarten Bosma and Brian Ichter and Fei Xia and Ed Chi and Quoc Le and Denny Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". (2023). https://arxiv.org/abs/2201.11903

[2] What is RAG? https://aws.amazon.com/what-is/retrieval-augmented-generation/

[3] Augmenting LLM Apps with Vector Databases https://medium.com/@bijit211987/augmenting-llm-apps-with-vector-databases-fe44b774334a

[4] Apache Software Foundation, 2010. Hadoop, Available at: https://hadoop.apache.org

[5] Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." arXiv preprint arXiv:2307.09288 (2023).

2. Overall Description

2.1 Product Perspective

The SolrChain specifies the features of a new, self-contained product. The system hopes to create an LLM pipeline with improved performance on Q&A setups, which contain a specialised knowledge base and leads to outputs with Chain of Though Reasoning in real-time.

2.2 Product Functions

The web-based system for querying our RAG-integrated SolrChain accommodates diverse user roles and functionalities. External participants can seamlessly create accounts and get improved query replies using our framework. The Administrator holds the authority to add or remove users, and to manage and monitor the server and containers cohort.

2.3 User Classes and Characteristics

We define two broad user classes in our software implementation

a. End Users

The end users are the ones that input their prompts into the pipeline using the web interface. The experience is similar to using any other web based LLM service.

b. Administrator

The Administrator gains access to comprehensive database information, including the complete documents and distributed cluster access. Moreover, the admin can mange the load balancing and create more docker containers to distribute load on the system.

2.4 Operating Environment

The operating environment for any user is one in which they should have an internet-connected device and valid credentials to log in. However, users can explore the landing and Events page without logging in. The web application is compatible with all standard web browsers and operating systems.

2.5 Design Implementation and Constraints

The software’s efficiency depends on the manual work of maintaining the relevant documents and knowledge graphs in the Solr database. The software is also constrained by the number of docker containers and servers, which can be scaled indefinitely.

2.6 User Documentation

The software will come with a user manual to guide users in properly using the website.

2.7 Assumptions and Dependencies

As any LLM pipeline would state, they can make mistakes, and validating the output for sensitive queries is necessary.

3. External Interface Requirements

3.1 User Interfaces

Our User Interface will be implemented using Flask 3.0.2. Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. Our backend will be implemented in Python 3.10.x, and the front-end and back-end will communicate seamlessly over a local server that Flask helps us to host (deployment using Apache HTTP Server has been detailed later).

The user interface can be divided into multiple major sections:

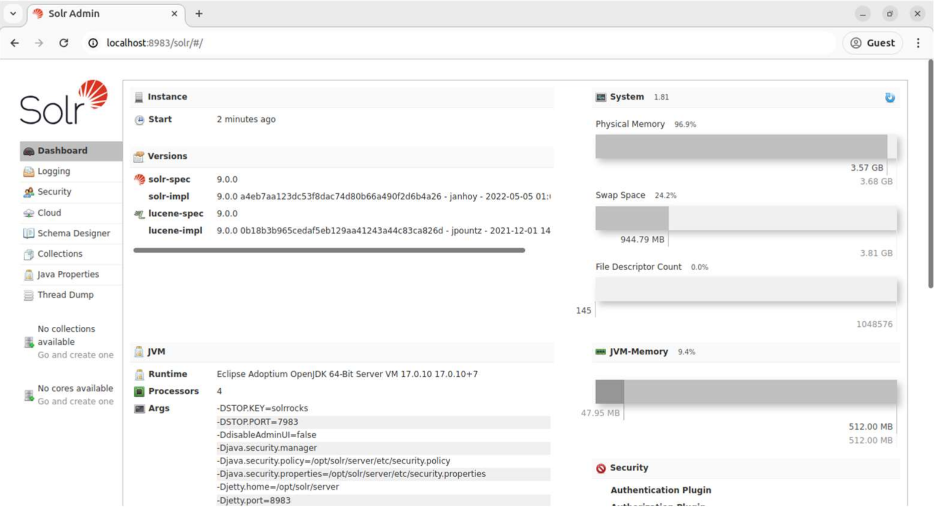
a. Landing Page

The landing page serves as the initial point of entry for all users visiting the website for the first time. Upon arrival, visitors are presented with various options, including logging in as different stakeholder classes or browsing the landing page anonymously. Here, users get a input bar for their queries which on entered gets the output from the pipeline deployed.

b. Admin Panels

The system administrator is provided with three interfaces to control the pipeline and database deployment. We list the three interfaces as follows

- **Solr Dashboard (localhost:8983):** The left-side of the screen is a menu under the Solr logo that provides the navigation through the screens of the UI. The first set of links are for system-level information and configuration and provide access to Logging Screen, Collections / Core Admin, and Java Properties Screen, among other things. At the end of this information is at least one pulldown listing Solr cores configured for this instance. On SolrCloud nodes, an additional pulldown list shows all collections in this cluster. Clicking on a collection or core name shows secondary menus of information for the specified collection or core, such as a Schema Browser Screen, Files Screen, Plugins & Stats Screen, and a Query Screen on indexed data. The left-side navigation appears on every screen, while the center changes to the detail of the option selected. The Dashboard shows several information items in the center of the screen, including system uptime, the version being run, system-level data, JVM arguments, and the security plugins enabled (if any). Under the covers, the Solr Admin UI uses the same HTTP APIs available to all clients to access Solr-related data to drive an external interface.
- **Prometheus (localhost:9090):** The Prometheus web UI allows the administrator to get the Zookeeper node metrics and configure Alerts accordingly. It is used for system monitoring and aids in the scalable design of our pipeline
- **Grafana (localhost:3000):** It is an open-source analytics and interactive visualisation web application that has its data source connected to the Prometheus server, giving the administrator greater and easier control over the pipeline.



localhost:8983/solr/#/

No collections available
Go and create one

No cores available
Go and create one

Instance

Start	2 minutes ago

Versions

solr-spec	9.0.0
solr-impl	9.0.0 a4eb7aa123dc53f8dac74d80b66a490f2d6b4a26 - janhoy - 2022-05-05 01:
lucene-spec	9.0.0
lucene-impl	9.0.0 0b18b3b965cedaf5eb129aa41243a44c83ca826d - jpountz - 2021-12-01 14

JVM

Runtime	Eclipse Adoptium OpenJDK 64-Bit Server VM 17.0.10 17.0.10+7
Processors	4
Args	-DSTOP.KEY=solrrocks -DSTOP.PORT=7983 -DdisableAdminUI=false -Djava.security.manager -Djava.security.policy=/opt/solr/server/etc/security.policy -Djava.security.properties=/opt/solr/server/etc/security.properties -Djetty.home=/opt/solr/server -Djetty.port=8983

System 1.81

Physical Memory 96.9% — 3.57 GB / 3.68 GB

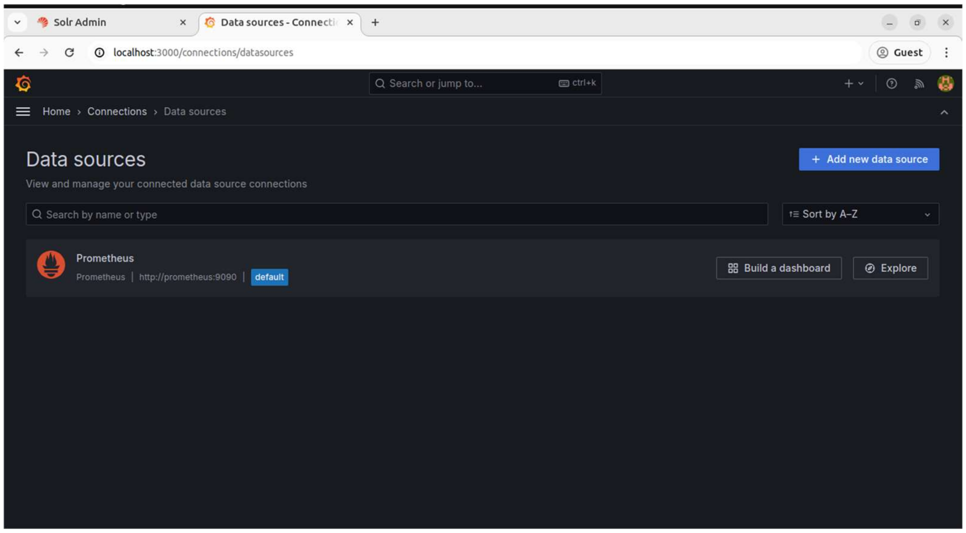
Swap Space 24.2% — 944.79 MB / 3.81 GB

File Descriptor Count 0.0% — 145 / 1048576

JVM-Memory 9.4% — 47.95 MB / 512.00 MB / 512.00 MB

Security

Authentication Plugin

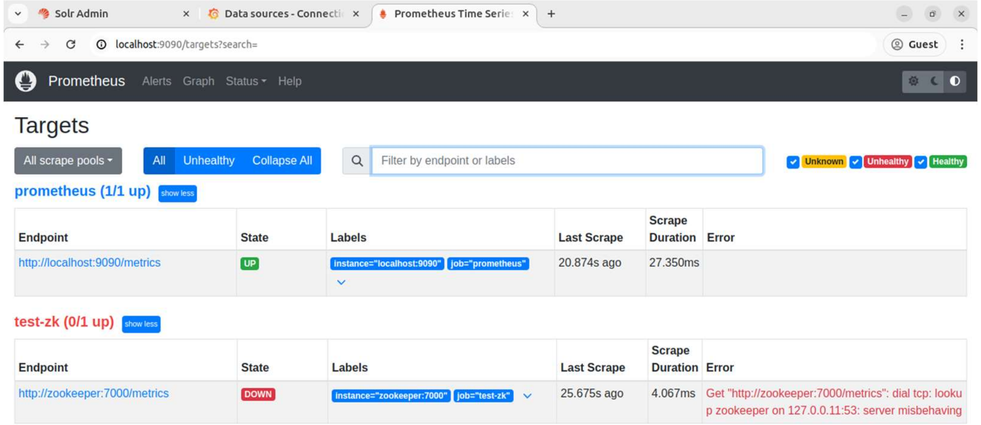


localhost:3000/connections/datasources

Data sources

View and manage your connected data source connections

Prometheus
Prometheus | http://prometheus:9090 | default



localhost:9090/targets?search=

Targets

prometheus (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	20.874s ago	27.350ms	

test-zk (0/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://zookeeper:7000/metrics	DOWN	instance="zookeeper:7000" job="test-zk"	25.675s ago	4.067ms	Get "http://zookeeper:7000/metrics": dial tcp: lookup zookeeper on 127.0.0.11:53: server misbehaving

3.2 Hardware Interfaces

The application does not raise any non-standard hardware interrupts and faults. Hence, the design philosophy has not incorporated the hardware module design and interface.

3.3 Software Interfaces

Our backend software will run on Python 3.10.x. We have adhered to the object-oriented programming paradigm as much as possible. Further, our backend will communicate with both the Flask server for frontend and Solr/Zookeeper nodes for databases. Whenever a user logs in, an object of the class is instantiated with required data fields fetched from the SQLite databases.

Weightage has been given to clean and readable code with industry-standard linting and code formatting. We make use of Git version control, Flake8, autopep8 and isort to regulate our workflow and standardization.

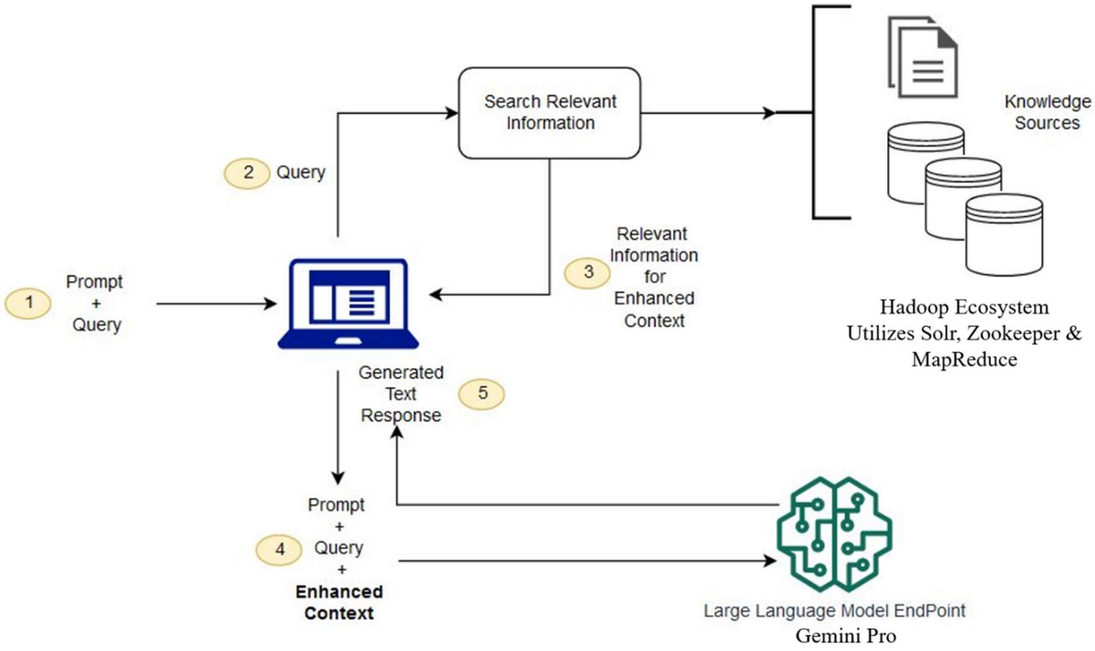
3.4 Communication Interfaces

The software interacts with users through both text and graphical user interfaces (GUI). Based on the functionality expected from the user, it redirects them to the appropriate web address. The prompt input sends the query to backend which ensures speedy reply to the user which is more accurate and informative than using raw LLM models.

4. System Features

4.1 LangChain

LangChain is a Python framework used to make LLM pipelines deployable. They provide a robust solution and are widely used in commercial applications. We employ the Langchain framework to integrate our LLM and Solr nodes in order to create a Retrieval Augmented Generation pipeline. It has native support for Gemini Pro, and thus, it was possible to integrate the LLM API with our model seamlessly. The most important hurdle was to query the Solr nodes, as there was no LangChain support for it. An incomplete GitHub Python library was found and amended to integrate Solr, which took up most of the project time. Appropriate prompt template as formatted to output appreciable answers. Text formatting was employed between each sub-step. Finally, the raw user query is received at the backend and fed into the pipeline. The RAG-based output is then displayed to the end user.


1 Prompt + Query
2 Query
Search Relevant Information
Knowledge Sources
3 Relevant Information for Enhanced Context
Hadoop Ecosystem Utilizes Solr, Zookeeper & MapReduce
Generated Text Response 5
4 Prompt + Query + Enhanced Context
Large Language Model EndPoint
Gemini Pro

4.2 Retrieval Augmented Generation

When they register or edit their profiles, emails are sent to students and participants. Moreover, the organizer of an event can send emails to the participants and volunteers of his event. The mailing interface is done by using a SMTP server, the configuration is provided by Flask-Mail extension. The receiver SMTP server is smtp.google.com

4.3 Chain of Though Reasoning

The idea behind chain of thought prompting is that showing the LLM examples of logical reasoning to approach a solution will encourage it to follow a similar method of reasoning in its response. We implement this by uploading relevant mathematical documents in our Solr nodes, providing step by step solution in our RAG model.

4.4 Large Language Model – Gemini Pro

The original abstract proposed using Llama 2, but we found that Gemini Pro was free to access, and we did not need to download weights like Llama 2. This gave us the advantage of just using API to prompt the LLM servers, saving space and computation. This does lead to a small drawback of latency. Further inspiration to use Gemini is the inbuilt support of Langchain for it.

4.5 MapReduce

Hadoop's MapReduce framework excels in parallelising large-scale data processing. By distributing RAG's initial context retrieval phase across multiple nodes, MapReduce significantly speeds up this step. This parallelisation enables the system to scan massive datasets simultaneously. Complementing MapReduce, Apache Solr provides high-performance indexing and distributed search capabilities. Solr's advanced indexing techniques built upon Lucene create a highly efficient search engine optimised for the knowledge base. This allows for near-instantaneous retrieval of the most relevant documents during the RAG process, minimising delays. By strategically integrating Hadoop's MapReduce and Apache Solr, we can overcome the latency drawbacks of RAG. MapReduce distributes the workload for large-scale context retrieval, while Solr ensures swift and targeted searches within the knowledge base. This combined approach makes RAG a more viable solution for applications demanding real-time performance, particularly those involving chain-of-thought reasoning.

4.6 Information & System Security

Special attention has been given to the security and privacy integrity of the web application architecture. Our application handles a vast amount of data like personal data, making it a prime target for malicious actors. We need to implement security features to safeguard from data breaches, unauthorized access, injection attacks and cross-site scripting. This not only protects but also preserves the integrity and reliability of the application itself. In the following subsections, we highlight how we are implementing data security policies in our web framework.

4.6.1 User Authentication

Users are authenticated before every request they make to the server using secure cookies. They are explicitly authenticated at login, register and edit profile pages using HTML forms. These forms are protected using CSRF tokens to prevent corresponding attacks. The passwords are hashed and stored in our database, which provides data security in case of breach.

4.6.2 Session Management and Cookies

We implement user sessions that are set as secure cookies (the ones that can only be sent back over an HTTPS connection to the server. We set a total of 3 cookies in a session instance as follows:

1. user_id: It stores the primary key of that user. This serves personalization of content and also authentication.
2. role: This token effectively stores the user type, which can be user or admin. This aids the backend to provide respective services to each user.
3. CSRF_token: This cookie that is a nounce and will be used to protect against CSRF attacks. They are set when the user interacts with a form. The details will be discussed shortly.

We integrate security by setting finite idle session time (1800 seconds) and Flask inherently provides us with SecureCookie implementation. Moreover, we have abided by the standard web browser rules to keep a number of cookies to a minimum. All the cookies are just essential cookies that aid in the normal functioning of the website and no other cookies have been set respecting the privacy of the user.

4.6.3 SQL Injection Attacks

The dreaded SQL Injection attacks have been managed by sanitizing the query strings.

4.6.4 Cross Site Request Forgery (CSRF) Attacks

CSRF attacks are commonly carried out in applications that involve form. A malicious site can send a POST request to our website to make unintended changes when a user session is active. To prevent this, we introduce a CSRF token, which is a nounce that is integrated as a hidden field in every form that can't be replicated by the attacker. This eliminated almost all the CSRF attacks.

4.6.5 Cross Site Scripting (XSS) Attacks

XSS or CSS attacks happen when an attacker attempts to introduce malicious code into our website. We have attempted to sanitize the data that comes from the user side at all steps and also minimize the use of such inputs in server-side functions.

4.6.6 Privilege Setup

When the application is deployed as highlighted in a later section, appropriate permissions are set in the server file system in order to prevent higher privileges in case of breach, mitigating any further exploit.

5. Solr/Zookeeper Database Modelling

5.1 Dataset for RAG

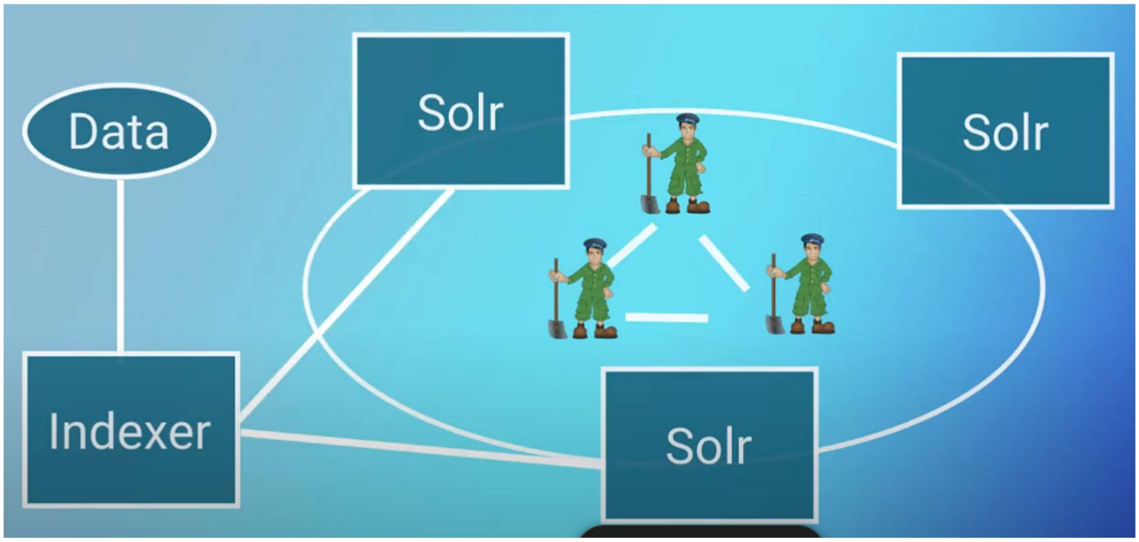
To update the documents for RAG in our Solr nodes, we scrape the top 50 and top 25 cited research papers of Professor Pabitra Mitra and Professor K. S. Rao, respectively. The PDF text has been transformed into chunks of size 500 with overlap of 20 tokens. Vector embedding is further performed upon then (the embedding is normalised) using the HuggingFaceEmbeddings module. The text document, along with its vector embedding, is then stored in the Solr collection. Cosine similarity is used to find (Dense Vector Search functionality of Solr) the top 4 matching documents with the query to retrieve to the RAG pipeline.

5.2 Solr Cloud Collections & Documents

We create a collection named Langchain in our Solr cloud. The total number of uploaded documents for testing is over 760, each containing text and its corresponding vector embedding.

5.3 Docker Layout

We create 3 Solr nodes and 3 Zookeeper nodes, each one of them interacting with each other to ensure smooth operations. All the nodes are defined in the Docker compose file, which also initialises the Prometheus and Grafan nodes. The Zookeeper nodes are to provide redundancy and load balancing between the Solr nodes. The following diagram depicts the layout.



6. Scalability, Monitoring & Dockerization

6.1 Apache HTTP Server

We have self-hosted the web application using Apache HTTP as a web server. We selected Apache as it is a popular, robust and scalable web server sufficing our needs. The server address is http://10.145.95.65/, and it is hosted on the campus' internal network. The *.conf* has been appropriately configured. We attempted to use Certbot to provide SSL certification to transpile the application to HTTPS, but it failed as the port has not been forwarded to the eternal internet.

6.2 Scalability in terms of concurrent end-users

The pipeline has been created such that it can be indefinitely expanded to industrial scals by just increasing the number of servers. Even the basic module contains load balancing and redundancies.

6.3 Docker Containers

Dockerization is heavily used in the setup to simulate multiple server clusters.

6.4 Prometheus & Grafana

Details of Prometheus and Grafana are provided in the administrator panel sectione. They are used to monitor the docker cluster.

6.5 Web Server Gateway Interface (WSGI)

To establish communication of the Flask applications with the Apache server, a *.wsgi* file was configured using the mod_wsgi module.

6.6 Linking to https://cse.iitkgp.ac.in/ domain

To link the internal IP to our cse domain, we create a *.htaccess* file in the public_html folder of the sftp server.

7. Other Non-functional Requirements

7.1 Performance Requirements

Our SolrChain is built upon 3 major modules, namely the backend, the frontend and the databases. The front end is chosen to be hosted as a web application because it allows seamless cross-platform usage. Further the entire premise of a Solrchain depends on it being online, scalable and accessible anywhere and anytime. Given their proven performance benefits and mutually seamless compatibility, Python, Solr/Zookeeper, Prometheus and Grafana are chosen to work with.

7.2 Safety and Security Requirements

Databases are crucial for our application. We implement all the security standards both for our databases by encrypting them and our frontend forms using Flask's built-in form encryption.

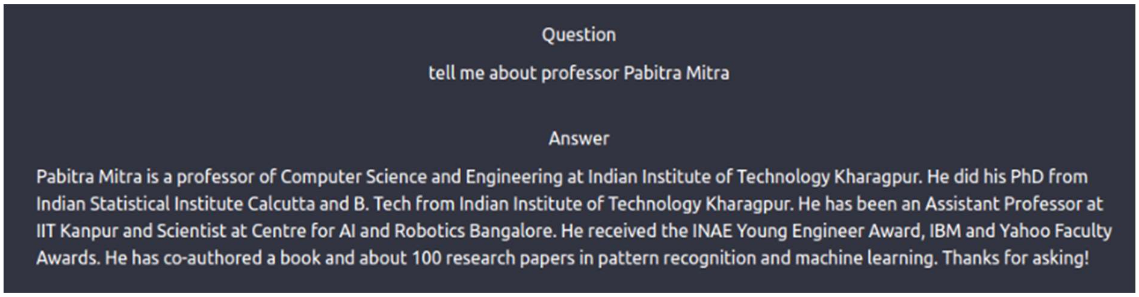
7.3 Software Quality Attributes

Our SolrChain promises the utmost quality and robustness. Both ease of usage and functional completeness have been ensured at each step. There is a clear distinction between rights of the different stakeholder classes which ensures no misuse/loss of business and data. The separate hosting of the movies and auditing allows for seamless changes in the library. Further, the admin is given rights to delete entries and change rights of other classes as well. All these features combined make our SolrChain complete on all fronts of quality.

7.4 Business Rules

The Software during development and after deployment is not meant to be monetized. The authors agree to publish the software under the guidelines of open-source Creative Commons after final evaluation.

Appendix A: Comparision with raw Gemini Pro

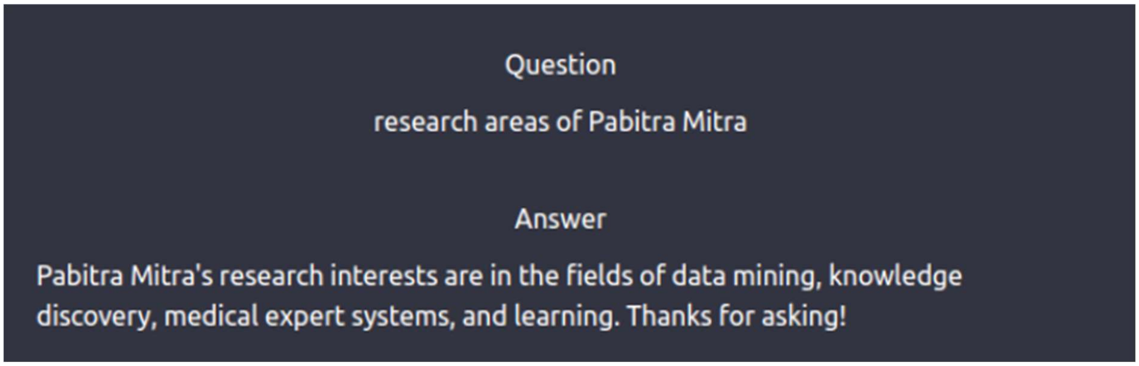


Question

tell me about professor Pabitra Mitra

Answer

Pabitra Mitra is a professor of Computer Science and Engineering at Indian Institute of Technology Kharagpur. He did his PhD from Indian Statistical Institute Calcutta and B. Tech from Indian Institute of Technology Kharagpur. He has been an Assistant Professor at IIT Kanpur and Scientist at Centre for AI and Robotics Bangalore. He received the INAE Young Engineer Award, IBM and Yahoo Faculty Awards. He has co-authored a book and about 100 research papers in pattern recognition and machine learning. Thanks for asking!

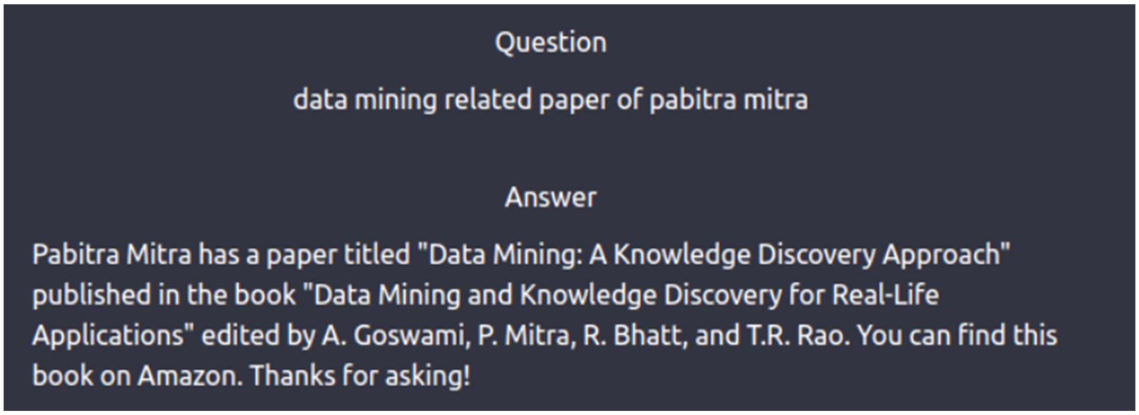


Question

research areas of Pabitra Mitra

Answer

Pabitra Mitra's research interests are in the fields of data mining, knowledge discovery, medical expert systems, and learning. Thanks for asking!

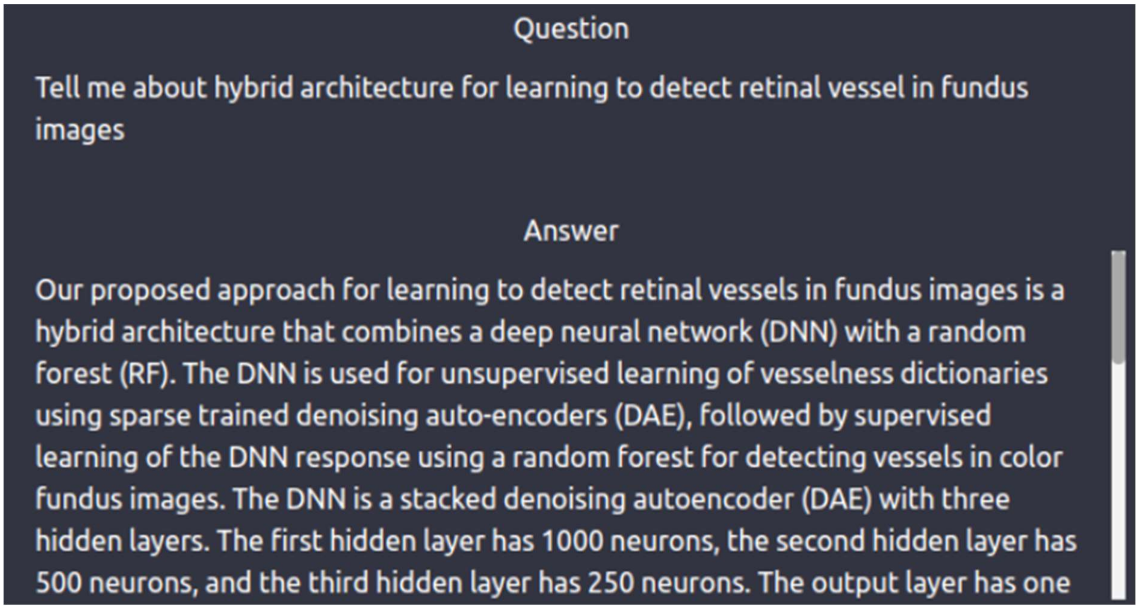


Question

data mining related paper of pabitra mitra

Answer

Pabitra Mitra has a paper titled "Data Mining: A Knowledge Discovery Approach" published in the book "Data Mining and Knowledge Discovery for Real-Life Applications" edited by A. Goswami, P. Mitra, R. Bhatt, and T.R. Rao. You can find this book on Amazon. Thanks for asking!



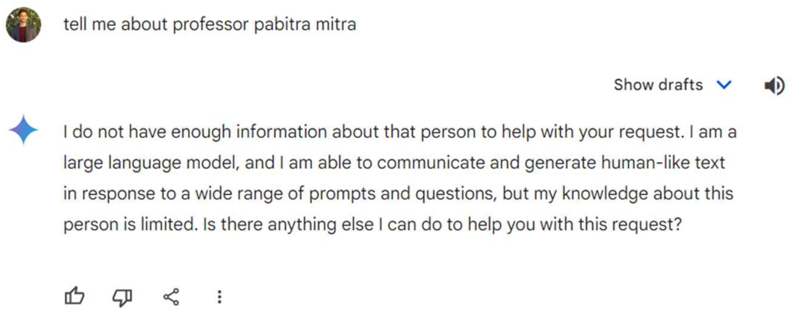
Question

Tell me about hybrid architecture for learning to detect retinal vessel in fundus images

Answer

Our proposed approach for learning to detect retinal vessels in fundus images is a hybrid architecture that combines a deep neural network (DNN) with a random forest (RF). The DNN is used for unsupervised learning of vesselness dictionaries using sparse trained denoising auto-encoders (DAE), followed by supervised learning of the DNN response using a random forest for detecting vessels in color fundus images. The DNN is a stacked denoising autoencoder (DAE) with three hidden layers. The first hidden layer has 1000 neurons, the second hidden layer has 500 neurons, and the third hidden layer has 250 neurons. The output layer has one

While Gemini Pro on all such prompts outputs the following:



tell me about professor pabitra mitra

I do not have enough information about that person to help with your request. I am a large language model, and I am able to communicate and generate human-like text in response to a wide range of prompts and questions, but my knowledge about this person is limited. Is there anything else I can do to help you with this request?

Appendix B: Code Maintenance

The code files are hosted in a GitHub repository:
https://github.com/PrasannaPaithankar/DBMS-Laboratory-Spr-24

Appendix C: Glossary

Python	Programming language
Flask	python API framework to seamlessly integrate frontend and backend
GUI	Graphical User Interface
Server	A computing node that provides service to other client nodes

Appendix D: Conflict of Interest

The authors share no conflicts of interest and the authorship list follows the Hardy-Littlewood Rule.