OSPP Project Proposal

Project Name： Implement Document LLM support in MatrixOrigin Database

Mentor： Arjun

Applicant： Charles, Han, Yazi

Date： 2024.06.04

---

1. Introduction

We are excited to submit our proposal for the OSPP project under the mentorship of Arjun. This project presents an exceptional opportunity for us to contribute to the MatrixOne open-source database, a platform we deeply admire for its innovative approach to data management.

We possess a robust skill set that aligns well with the project requirements. Our proficiency in Golang and database knowledge will be instrumental in implementing support for the DATALINK type. Our expertise in large language models (LLMs) will be crucial for developing document chunking and vector embedding models within the MatrixOrigin Database. Additionally, our experience with Python and comprehensive skills in front-end and back-end development equips us with the versatility needed to create a seamless application.

We have invested a lot of effort in preparing for this project. Given that the project aims to provide support for the DATALINK data type in MatirxOne, we have reviewed relevant parts of the project's source code and studied detailed documentation provided by IBM DB2 regarding DATALINK support. Since the project involves document chunking and generating vector embedding models, we examined portions of the LangChain source code and studied LLM functionality. We also researched various chunking strategies and compared Retrieval-Augmented Generation (RAG) tools, such as GPT and BERT. To prepare for developing the final application, we are familiarizing ourselves with the AWS S3 API.

Our motivation to participate in this project is driven by its potential to enhance our technical skills and allow us to contribute meaningfully to the MatrixOne community. In conclusion, our technical expertise and enthusiasm for this project make us ideal candidates to undertake and successfully complete this initiative. We are committed to delivering high-quality results that meet and exceed the project’s objectives.

2. Project Challenges and Key Points

2.1 Adding Support for DATALINK Type in MatrixOrigin

Challenge: Implementing and integrating DATALINK type support

Key Points:

- Compatibility with existing data types and structures should be ensured
- Mechanisms to check and enforce links should be implemented
- All DATALINK operations should be transactional, maintaining consistency across the MatrixOne and external storage systems
- Should pay attention to follow-up operations to manage the stored DATALINKs

2.2 Document Chunking and Vector Embedding Generation

Challenge: efficiently chunking documents and generating high-quality vector embeddings.

Key Points:

- research chunking srategies and embedding generation tools, choose an appropriate one for our application
- we should offer different chunking strategy options based on the documents for users to choose

2.3 Implementing Query Functionality for Similar Chunks Retrieval

Challenge: developing an efficient query system to retrieve document chunks similar to the input query text.

Key Points:

- optimize retrieval to return top N similar chunks quickly, generate N similar questions as the intermediary keys to get the accurate vector embeddings for input query text.
- for two overlapping chunks, which are the context for each other, we will return both as a whole for the user's input query

2.4 Integration with Chatbot Application for Demonstration

Challenge: Developing a chatbot application that performs high-quality retrieval and stores questions and answers efficiently

Key Points:

- The chatbot will support multiple rounds of interaction to continuously optimize the accuracy of search results
- when storing questions and answers, we will use the model to generate multiple similar questions and store these questions together to improve the matching degree during retrieval

3. Methodology

3.1 Add Support for DATALINK Type in MatrixOrigin

To add support for the DATALINK type in MatrixOne, packages related to data types in the database should all be modified, for example `matrixone/pkg/defines/type.go` and `matrixone/pkg/container/types/types.go`:

1. Define the datalink data type in all related packages
2. Modify the code that handle data types and schema management logic to support columns of the DATALINK type
3. Testing

Step1: add datalink type definition

Add a new type definition in `matrixone/pkg/container/types/types.go`:

```
type DATALink struct {

   URL string

  // other metadata fields...
}
```

Add a new type identifier in the type enumeration in `matrixone/pkg/defines/type.go`:

```
const {

  // existing types...

  T_datalink T = 72 // or other unused value


}
```

Add DATALINK to the type mapping in `matrixone/pkg/container/types/types.go`:

```
var Types = map[string]T{

  // existing types...

  "datalink": T_datalink,
}
```

If the interoperability with MySQL is needed, a new MySQL type identifier can be added in `matrixone/pkg/container/types/types.go`:

```
const (

   // existing types...

   MYSQL_TYPE_DATALINK MysqlType = 0xXY // or other unused value
)
```

Step2: data type handling and related logic (validation->storage->retrieve->SQL parser->process query)

according to `matrixone-main/pkg/container/types`, a datelink.go should be created

validation of typical URL/URI:

```
// validation of the link
func (d *DATALink) validateDatalink() error {

 if _, err := url.ParseRequest(d.URL); err != nil {

  return errors.New("invalid URL format")

 }

 return nil
}
```

The code handling table definitions and schema management would need to be extended to support columns of the DATALINK type. This involves changes in the table creation logic, metadata storage, and schema validation:

```

// store process

func StoreValue(value interface{}) (interface{}, error) {

  switch v := value.(type) {

  case types.DATALINK:

    if err := v.validateDatalink(); err != nil {

      return nil, err

    }

    // Store as string or specific data structure

    return v.String(), nil

  // Handle other types...

  default:

    return nil, fmt.Errorf("unsupported type: %T", v)

  }

}

// retrieve process

func RetrieveValue(value interface{}) (interface{}, error) {

  // For DATALINK, convert back to DATALINK type
  if str, ok := value.(string); ok {

    return types.DATALINK(str), nil

  }

  return nil, fmt.Errorf("unsupported type: %T", value)

}

// query process

func ProcessQuery(query string) (interface{}, error) {

  // Parse and process query

  // Simplified example: SELECT statement handling

  if query == "SELECT * FROM my_table WHERE link_column = 'http://example.com'" {

    return types.DATALINK("http://example.com"), nil

  }

  // Handle other queries...

  return nil, fmt.Errorf("unsupported query: %s", query)

}
```

Step3: testing and debugging

a new datalink_test.go should be created

Example unit tests may be like:

```
func TestDATALINK(t *testing.T) {

  dl := types.DATALINK("http://example.com")

  if err := dl.validateDatalink(); err != nil {

    t.Errorf("expected valid DATALINK, got error: %v", err)

  }

  invalidDL := types.DATALINK("invalid_link")
```

```
    if err := invalidDL.validateDatalink(); err == nil {

      t.Errorf("expected invalid DATALINK, but got no error")

    }

  }
```

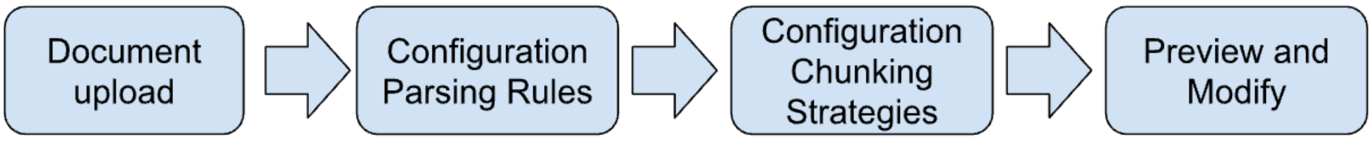
3.2 Document Chunking and Generating Vector Embeddings

When dealing with document data, chunking the document and generating vector embeddings can improve data storage and retrieval efficiency. Using vector embedding models enables more efficient text similarity calculations, thus enhancing query performance.

3.2.1 Document Upload

1. Three options for users while uploading documents.

- Quick upload
  - default parsing rules, ignore images and tables.
  - default output format: txt.
- Advanced upload


Document upload
Configuration Parsing Rules
Configuration Chunking Strategies
Preview and Modify

- Batch upload
  - Once parsing rules and chunking strategies are set up, they can be applicable to all the files.

2. Configuration parsing rules

Support for different document types with specific processing rules, like PDF, WORD, PPT, EXCEL, TXT, JSON and etc, which need final decisions in the upcoming discussion meeting. We take PDF as an example of pre-processing rules.

Document Type	**Image/Table Processing Rules**	**Parsing Rules**
PDF	Image OCR, Table OCR	OCR, Convert all text in images and tables to text, with coordinates preserved, and extract all texts. Note: Ensure text and coordinates are preserved.
PDF	Image Ignored, Table OCR	OCR, Convert all text in tables to text, ignore images. Note: Ensure text and coordinates are preserved.
PDF	Image OCR, Table Ignored	OCR, Convert all text in images to text, ignore tables. Note: Ensure text and coordinates are preserved.
PDF	Images are replaced with placeholders.	Images are replaced with placeholders, which can be edited online. The placeholder format needs to include an incrementing number
PDF	Images are replaced with short links.	The images need to be identified and saved on the server, and their locations should be replaced with short links. Note: Ensure short links, not long links, are required.
PDF	Image/Table Ignored	Ignore images and tables

3. Configuration chunking strategies

- Inputs are required.
- Once set up, these strategies can be applicable to all files.
- Default delimiters: |？|\?|！|!|，|\n| symbols, | indicates "or". Delimiters can be customized. Supports regular expressions.
- Chunking length input is optional：
    - If not specified, there is no limit on the chunk length. Once setup, applicable to all files. If the specified character position is not a punctuation mark, it will find the nearest period, question mark, or exclamation mark as the chunk delimiter. If none, find ,; marks.

3.2.2 Document Chunking

1. RAG tools: use third-party tooling (LangChain, NLTK, SpaCy etc), we choose LangChain for the reasons:

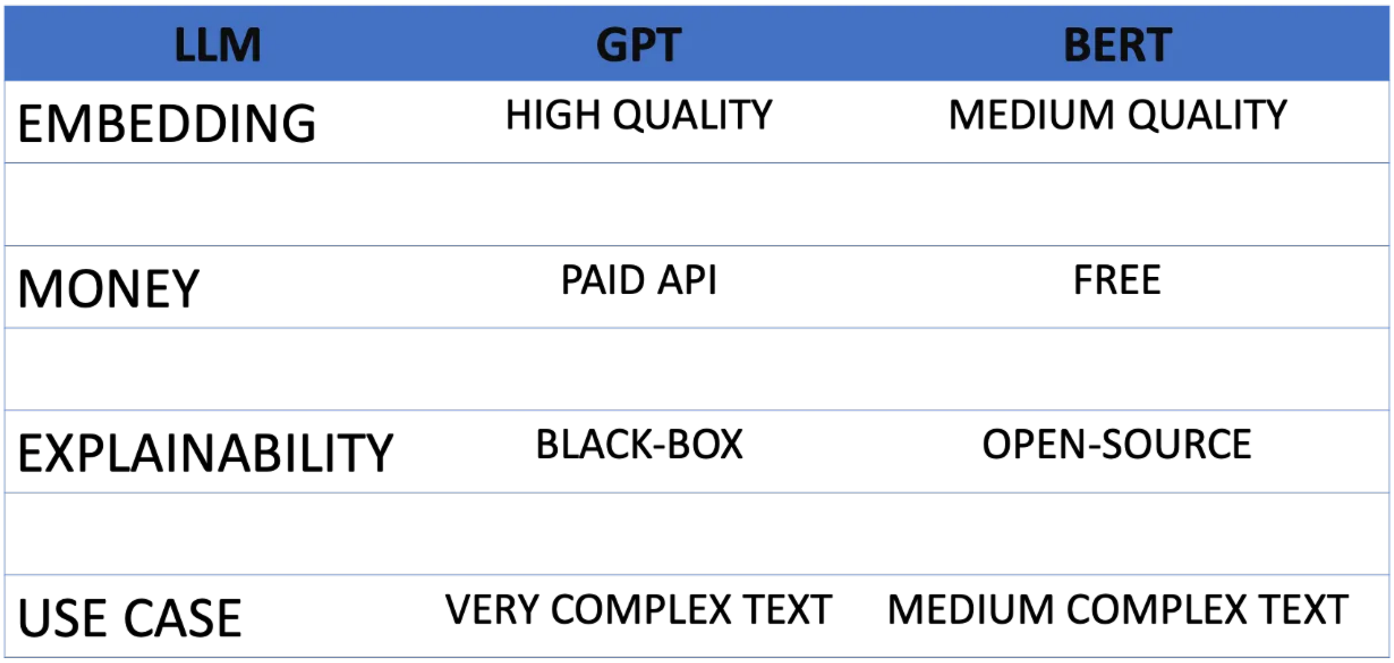
- Integration with LLMs: Designed specifically to work with large language models, making it easier to integrate advanced NLP capabilities.
- Flexibility: Supports a wide range of tasks including text generation, summarization, and question-answering.
- Ease of Use: Simplifies complex tasks with high-level abstractions.

2. If the chunk of text makes sense without the surrounding context to a human, it will make sense to the language model as well. Therefore, finding the optimal chunk size for the documents in the corpus is crucial to ensuring that the search results are accurate and relevant. There are different kinds of chunking strategies, we take PDF and EXCEL as examples. And we should find the appropriate chunking strategies for our application.

Document Type	Option	Chunking Strategy
PDF	Title	Chunking by title (customizable features, e.g 1, 1.1)
PDF	Page	Chunking by pages (customizable features, e.g P43, page43)
PDF	Special Symbols	Chunking by special symbols (customizable features)
PDF	Word count	Chunking by word count (customizable word count)
EXCEL	Behavior (the first row is the header)	Chunking by rows (the first row is the header)
EXCEL	Behavior (no header)	Chunking by rows (no header)

3.2.3 Generating Vector Embeddings

1. Use tools to generate vector embeddings, like GPT or BERT. We have researched them and the results are as below. We choose GPT who does a better job by the quality of clustering.



LLM	GPT	BERT
EMBEDDING	HIGH QUALITY	MEDIUM QUALITY
MONEY	PAID API	FREE
EXPLAINABILITY	BLACK-BOX	OPEN-SOURCE
USE CASE	VERY COMPLEX TEXT	MEDIUM COMPLEX TEXT

2. Pseudo code in python:

```
##Read data

file_name = 'path_to_file'

df = pd.read_csv(file_name)

##Set parameters

embedding_model = "text-embedding-ada-002"

embedding_encoding = "cl100k_base" # this the encoding for text-embedding-ada-002

max_tokens = 8000 # the maximum for text-embedding-ada-002 is 8191

top_n = 1000

encoding = tiktoken.get_encoding(embedding_encoding)

col_embedding = 'embedding'

n_tsne=2

n_iter = 1000

##Gets the embedding from OpenAI

def get_embedding(text, model):

 openai.api_key = "YOUR_OPENAPI_KEY"

 text = text.replace("\n", " ")
 return openai.Embedding.create(input = [text], model=model)['data'][0]['embedding']
```

3.2.4 Store Chunks and Embeddings

1. Once chunks and their embeddings are generated, create a table to store them in the PgVector database. Pseudo code:

- create extension vector;
- create a table to store our documents and their embeddings

```
create table documents (

 id bigserial primary key,

 content text,

 embedding vector(1536)

);
```

2. INSERT Operation, when inserting a new document, chunk the document, generate the corresponding vector embeddings, and insert the data into the document_chunks table.

3.2.5 ALTER, UPDATE, DELETE, and DROP Processes

1. CRUD operations

- ALTER: when modifying the table structure, ensure compatibility with existing chunks and embedding data.
- UPDATE: when updating a chunk, regenerate the vector embedding for that chunk and update the table record.
- DELETE: when deleting a document, delete all corresponding chunks and embedding records.
- DROP: when dropping the entire table, ensure data backup to prevent data loss.

2. To ensure simple and resilient embeddings, we have to take into account that:

- No modifications to the original table, which allows systems and applications that already use this table not to be impacted by changes to the embedding system.
- Automatically update embeddings when rows in the source table change. This lessens the maintenance burden. And this update need not be instantaneous or within the same commit. We can just ensure eventual consistency.

- Ensure resilience against network and service failures: as we generate embeddings via a call to the external OpenAI API, in scenarios where the external system is down or a network malfunction occurs, it's crucial that the remainder of our database system continues working.

3.3 Retrieve Chunks Similar to Input Query Text

1. To improve retrieval performance, we first generate similar questions based on the chunking using GPT and store these questions' embeddings.Then, we search these questions based on the input query first, and get the chunking to return to the user.
2. Use MMR algorithm (Maximal marginal relevance) to optimize for similarity to query AND diversity among selected documents, and get docs selected using the maximal marginal relevance.

> **Args:**
>
> query: Text to look up documents similar to.
>
> k: Number of Documents to return. Defaults to 4.
>
> fetch_k: Number of Documents to fetch to pass to MMR algorithm.
>
> Defaults to 20.
>
> lambda_mult: Number between 0 and 1 that determines the degree
>
> of diversity among the results with 0 corresponding
>
> to maximum diversity and 1 to minimum diversity.
>
> Defaults to 0.5.
>
> **Returns:**
>
> List of Documents selected by maximal marginal relevance and distance for
>
> each.

Pseudo code in python:

```python
def max_marginal_relevance_search_by_vector(

    self,

    embedding: List[float],

    k: int = 4,

    fetch_k: int = 20,

    lambda_mult: float = 0.5,
    **kwargs: Any,

  ) -> List[Tuple[Document, float]]:

    self.pingdb()

    session = self._get_session()

    sql = text("SELECT *,cosine_similarity(doc_embedding_vector, :embedding_str) as score
FROM %s ORDER BY score DESC LIMIT :limit_count ;" % (self.table_name))

    results = session.execute(

      sql, {'embedding_str': self._to_vector_str(embedding), 'limit_count': fetch_k})

    session.commit()

    session.close()

    results_maps = results.mappings().all()

    embeddings = [

      self._str_to_vector(result.doc_embedding_vector) for result in results_maps

    ]

    mmr_selected = maximal_marginal_relevance(

      np.array(embedding), embeddings, k=k, lambda_mult=lambda_mult

    )

    return [

      (

        self._document_from_payload(

          results_maps[i].payload, results_maps[i].id,
results_maps[i].doc_embedding_vector),

        results_maps[i].score,

      )

      for i in mmr_selected

    ]
```

3.4 Chatbot

This project will ultimately develop an intelligent database chatbot that helps users solve database-related issues through a conversational interface. The assistant can be based on various LLMs, such as Azure, Llama-2 13B Chat, and GPT-4. The database is deployed on a Kubernetes cluster, with AWS S3 for underlying storage. First, documents uploaded by users are persisted in both the database and file storage. Next, users can query the chatbot to retrieve details from the stored documents. The assistant is configured as a senior expert in databases, focusing on answering database-related questions. Additionally, other questions will be uniformly responded with, "I only answer questions related to databases."

To elaborate on the feature overview, the chatbot provides general database query assistance. The intelligent database chatbot interacts with GPT through the following steps:

1. The user inputs a question. This can be any database-related query.
2. The system generates multiple prompts. Based on the user's input, the system creates several possible prompts to ensure all aspects of the question are covered. These prompts may include different phrasings and angles to increase the chances of obtaining an accurate answer.
3. The prompts are sent to the GPT interface. The system sequentially sends the generated prompts to the GPT interface, leveraging its powerful natural language processing capabilities to parse and answer the questions.
4. The GPT responses are collected and displayed to the user. The system gathers all returned answers, filters, and integrates them to ensure accuracy and completeness. The best response is then displayed to the user.

Regarding the interaction process, the intelligent database assistant interacts with GPT through the following steps:

1. The user inputs a query. The user can input a question related to a specific table or describe their requirements, and the system will identify the intent of the query.
2. The system calculates the vector of the user's input. Using advanced natural language processing techniques, the system converts the user's text input into a vector representation to capture its semantic information.
3. The vector is used to search the vector database for tables with similarity above a threshold. The system searches the vector database for tables whose similarity to the user's input vector exceeds a preset threshold. This process ensures that the retrieved tables are highly relevant to the user's query.
4. The relevant table information is returned. The system returns the retrieved table information to the user, including table names, field descriptions, data types, and other details, helping the user quickly find the required tables.

Furthermore, in the process of retrieving relevant tables, first, the user inputs a query. Then, the system calculates the vector of the user's input. Next, the vector is used to search the vector database for tables with similarity above a threshold. Finally, the relevant tables are returned.

In terms of vector similarity and the vector database, vector similarity is achieved by first converting the text into vectors using a vectorization interface. Then, the relevance is determined by calculating the similarity of the vectors. Additionally, the vector database supports not only regular CRUD operations but also vector storage and search. Subsequently, results are returned sorted by similarity.

Considering storage triggers and synchronization, for storage triggers, documents uploaded by users are synchronized nightly, updating all tables across all clusters. Furthermore, for user-selected synchronization, users can choose to synchronize all table information under a specific cluster. First, the user selects the cluster to sync, then the system performs the synchronization.

To improve search quality, the implementation involves generating similar questions and optimizing storage and retrieval. First, after the user asks a question, GPT will generate three similar questions for retrieval, improving search quality. Additionally, when storing an answer, GPT first generates ten related questions, which are stored. Then, these questions are matched during retrieval to optimize results.

3.5 Bonus Goal: Access Control management

- Define roles and their corresponding permissions
- Modify the MatrixOne schema to include tables for users, roles, and permissions
- Define sets of rules that govern how data is accessed in the database and include standards for authentication, authorization, and auditing
- Use database encryption options such as Transparent Data Encryption (TDE) to secure data stored in the database and ensure that data exchanges are encrypted using protocols like TLS
- Test and debug

4. Related Tech Experiences

4.1 Golang Proficiency

- Charles is quite familiar with Golang. During the MIT6.824 class, he utilized Golang techniques like Go's RPC framework to develop a distributed MapReduce system. He implemented a worker process that managed file I/O and application functions, and a coordinator process that assigned tasks and handled failures. In addition, he designed and implemented a robust key/value server in Golang, ensuring linearizable operations and resilience against network failures through sophisticated duplicate detection and retry mechanisms.
- Han has some experience with Golang and designed and implemented a lightweight routing based on Golang during his undergraduate period. In this project, he encapsulates Request and Response, provides support for return types such as JSON and HTML, and designs and implements corresponding functions that automatically parse corresponding template functions. To implement dynamic route resolution, he designs prefix tree routing, and performance is guaranteed. In addition, there are other functions implemented such as group control, middleware, template rendering, html template rendering, and error-handling mechanisms.
- Yazi has been preparing Golang for this project and is familiar with its syntax and usage now.

4.2 LLM Knowledge

- Yazi is driving a copilot project that integrates an advanced Copilot tailored for share-point development environments. The objective is to enhance SharePoint's capabilities by enabling it to automatically generate customized pages based on user-provided queries and images, and realize multimodal authoring skills. Yazi has to utilize LLMs to interpret the textual queries and visual content provided by the users, and effectively parse the input and generate relevant SharePoint pages. Through this project, Yazi works on creating a new generative-AI LLM workflow.

4.3 Database Query Engine Flow

- Charles has a deep understanding of database. For the open-source database BusTub from CMU, he designed and implemented an efficient buffer pool manager, incorporating an LRU-K replacement policy to optimize memory usage and data retrieval speed. He also developed a B+ tree index structure that supported concurrent insertions, deletions, and queries, ensuring data integrity and efficient access even in multi-user scenarios.
- Through many project experiences, Han has an understanding of relational databases and NoSQL. Not only does he have hands-on experience with MySQL and Redis, but he also knows transactions, indexes, database paging, and database storage engines such as InnoDB, etc. Han is currently a back-end engineer intern at OPPO and the group's business is about cloud database. He is currently involved in improving and maintaining the group's OPPO database developer center(ODC). ODC hosts the management of multiple database products and is a platform that integrates data management, structure management, user management, permission management, security audit, and data operation. After getting familiar with ODC's business, Han will participate in the maintenance of the vector database Milvus within the group.
- Yazi has a deep understanding of MySQL, such as transactions, indexes, locking mechanisms; understanding of NoSQL, such as REDIS data types, persistence mechanisms, memory eviction mechanisms.

4.4 Familiar with Python

- Charles has a deep understanding of Python, TensorFlow, and PyQt. During COVID-19, he developed a Pneumonia Detector by making use of U-Net model and watershed algorithm for the segmentation of X-ray images, and CNN model as well as transfer learning model for the classification of pneumonia.
- Yazi used Python to implement a DATABASE that relates to Atlantic Coast Conference (ACC) basketball teams, and will allow queries to discover information about things like player statistics, team attributes, etc.

4.5 Frontend & Backend Development Experiences

- Charles is currently working as a Java Backend Developer in the Advertising Engine Team within the Core Local Business Department at Meituan. During his time there, he completed a system to build an inverted index by traversing an offline word list, retrieving standard product unit IDs from the index based on takeout keyword and weight score conditions, and updating the index periodically. He also improved the performance of retrieving by adjusting certain JVM parameters, implementing incremental updates instead of full index updates, and using caching.

- Han is currently working as a back-end engineer intern, understanding the basic concepts of back-end development, such as RESTful API design, database operations, etc. He masters the HTTP protocol, Web framework, and related technology stacks, such as routing, middleware, HTTP request/response processing, front-end and back-end interaction, etc. In addition, he understands software design principles and common architectural patterns (such as MVC and microservices), and has good code organization and architecture design capabilities. To achieve code management and team collaboration, he frequently uses Git and other version control tools. Han has good problem-solving and debugging skills and is able to quickly locate and solve common development problems.
- Yazi has developed an RPC framework enabling client-server communications, allowing clients to request services from remote server-side programs over the network. And Yazi utilized REDIS to implement a registry center with registration and subscription capabilities; implemented client-side load balancing through a consistent hashing algorithm.

5. Project Planning (06/10 - 09/30)

5.1 Plan

1. DATALINK type support: 3 weeks
2. Document chunking and generating vector embeddings: 4 weeks
3. Retrieve Chunks Similar to Input Query Text: 2 weeks
4. Integration with Chatbot Application： 3 weeks
5. Access Control management: 1 week
6. System task: 1 week

5.2 Timeline

TARGET	TIME	GOAL
Target 0: Preparation	Week 1	Everyone : - Learn golang. - Get a basic understanding of what happens in a Database: Vector, Batch, Vectorized Execution: https://www.youtube.com/watch?v=bZOvAKGkzpQ&t=554s - Understand how LLM works internally
	Week 2	Familiarize with MatrixOne(MO) codebase. (building source code, running BVT, running UT, and creating a PR, writing a Doc etc) - Work on this issue: https://github.com/matrixorigin/matrixone/issues/15027 . - Write a golang code that converts PDF to Chunks, Chunks to

		embeding - Store chunks and embedding in a Database (for now PgVector). - Write the code to read data from the PgVector database, convert it to meaningful sentense using langchain.
Target1: Add support for DATALINK	Week 3	Preparation: - review the packages that define old data types - review the storage, retrieve, and query pipelines Datalink type definition: - type definitiontype - identifier in type enumeration - type mapping - MySQL type identifierother - typed related packages
	Week 4	Datatype handling and related logic 1: - link validation - storage processretrieve process
	Week 5	Datatype handling and related logic 3: - SQL parser - query process - Testing and debugging for Datalink support
Task 2: Document chunking and generate vector embeddings	Week 6	Document Chunking: - Upload documents - Compare different chunking strategies and choose an appropriate one for our applications - Convert PDF (and other document types, TBD) to chunks
	Week 7	Generate embeddings: - For each chunk, generate similar questions using GPT - Generate the embeddings for the chunk and corresponding questions
	Week 8	Store chunks and embeddings - Implement ALTER, UPDATE, DELETE, DROP processes to support this new datalink type
	Week 9	Do tests and debug for chunking and embeddings
Task 3: Retrieve Chunks Similar to Input Query Text	Week 10	Retrieve Chunks Similar to Input Query Text - Use MMR algorithm to retrieve chunks - Tune the parameters to optimize the performance
	Week 11	Do tests and debug together with chunking and embeddings
Task 4: Integration		- Design and implement a user-friendly front-end with React.

with Chatbot Application	Week 12	- Set up the backend server, configure the Kubernetes cluster, and integrate AWS S3 for storage. - Configure LLMs and implement prompt generation logic.
	Week 13	- Develop query processing and display GPT responses. - Implement text-to-vector conversion and similarity-based retrieval.
	Week 14	- Generate similar questions to improve search quality and optimize retrieval processes. - Test, debug, and perform load testing. - Document the system, deploy it on the Kubernetes cluster, and provide user training materials.
Task 5: Access Control Management	Week 15	- Define roles and their corresponding permissions. - Define sets of rules that govern how data is accessed in the database and include standards for authentication, authorization, and auditing. - Use database encryption options such as Transparent Data Encryption (TDE) to secure data stored in the database and ensure that data exchanges are encrypted using protocols like TLS. - Test and debug.
Task 6:System test	Week 16	Test together.

5.3 Distribution of Work

Task	Charles	Yazi	Tanghan
Learn golang	●	●	●
Get a basic understanding of MatrixOne	●	●	●
Understand how LLM works	●	○	○
Type definition	●	○	○
Type identifier in type enumeration	●	○	○
Type mapping	●	○	○
MySQL type identifier	●	○	○
Other type related packages	●	○	○
Link validation	●	○	○

Storage process	●	○	○
Retrieve process	●	○	○
SQL parser	●	○	○
Query process	●	○	○
Testing and debugging for DATALINK support	●	○	○
Upload documents	●	○	●
Compare different chunking strategies and choose an appropriate one for our applications	○	●	○
Convert PDF (and other document types, TBD) to chunks	○	●	○
For each chunk, generate similar questions using GPT	○	●	○
Generate the embeddings for the chunk and corresponding questions	○	●	●
Implement ALTER processes to support this new datalink type	●	○	●
Implement UPDATE processes to support this new datalink type	●	○	●
Implement DELETE processes to support this new datalink type	●	●	○
Implement DROP processes to support this new datalink type	●	●	○
Test and debug for chunking and embeddings	○	●	○
Use MMR algorithm to retrieve chunks	○	●	○
Tune the parameters to optimize the performance	○	●	●
Tests and debug on retrieval together with chunking and embeddings	○	○	●
Front-end interface	○	●	●
Database integration	●	○	●
LLM integration	○	●	○
Response processing	●	○	○
Vector similarity and search implementation	●	●	●
Search quality optimization	●	●	●
Testing and debugging for the chatbot	●	○	●
Documentation	●	●	●
Deployment	●	○	●

Define roles and their corresponding permissions.	●	○	●
Define sets of rules that govern how data is accessed in the database and include standards for authentication, authorization, and auditing	○	●	●
Use TDE to secure data stored in the database and ensure that data exchanges are encrypted using TLS.	●	●	○
System test	●	●	●

● Leader ○ Assistant

5.4 GANTT Chart

Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Learn golang	⇒	⇒														
Get a basic understanding of MatrixOne	⇒	⇒														
Understand how LLM works	⇒	⇒														
Type definition			⇒													
Type identifier in type enumeration			⇒													
Type mapping			⇒													
MySQL type identifier			⇒													
Other type related packages			⇒	⇒												
Link validation				⇒												
Storage process				⇒												
Retrieve process				⇒												
SQL parser					⇒											
Query process					⇒											
Testing and debugging for DATALINK support					⇒											
Upload documents						⇒										
Compare different chunking strategies and choose an appropriate one for our applications						⇒										
Convert PDF (and other document types, TBD) to chunks						⇒										
For each chunk, generate similar questions using GPT							⇒									
Generate the embeddings for the chunk and corresponding questions							⇒									
Implement ALTER processes to support this new datalink type								⇒								
Implement UPDATE processes to support this new datalink type								⇒								
Implement DELETE processes to support this new datalink type								⇒								
Implement DROP processes to support this new datalink type								⇒								

Test and debug for chunking and embeddings									⇒							
Use MMR algorithm to retrieve chunks										⇒						
Tune the parameters to optimize the performance										⇒						
Tests and debug on retrieval together with chunking and embeddings											⇒					
Front-end interface												⇒				
Database integration												⇒				
LLM integration												⇒				
Response processing													⇒			
Vector similarity and search implementation													⇒			
Search quality optimization													⇒			
Testing and debugging for the chatbot														⇒		
Documentation														⇒		
Deployment														⇒		
Define roles and their corresponding permissions in the system															⇒	
Define sets of rules that govern how data is accessed in the database and include standards for authentication, authorization, and auditing.															⇒	
Use TDE to secure data stored in the database and ensure that data exchanges are encrypted using TLS.															⇒	
System test.																⇒