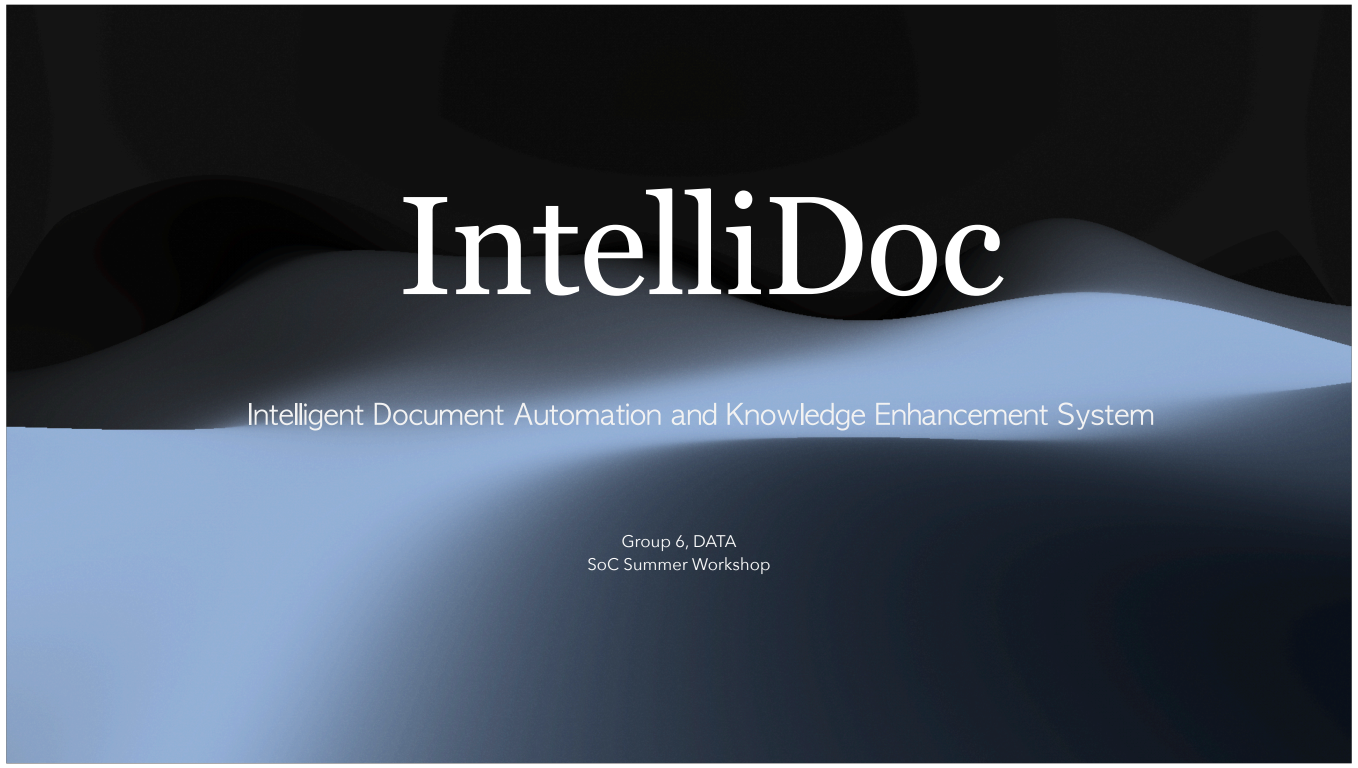


IntelliDoc

Intelligent Document Automation and Knowledge Enhancement System

Group 6, DATA
SoC Summer Workshop

(点击)
Good Afternoon every one,

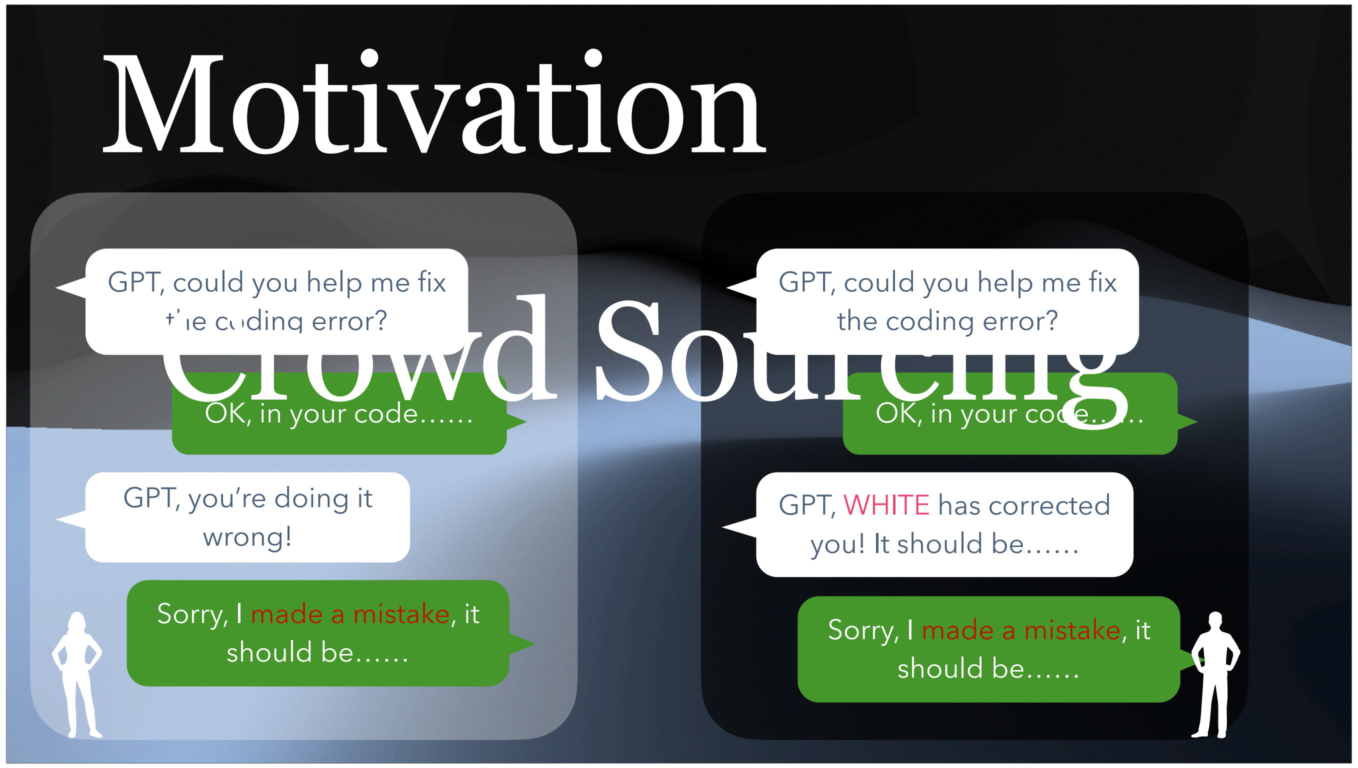
This is Group 6

Today, we are extremely honored to stand here to present our project: IntelliDoc

It is an Intelligent Document Automation and Knowledge Enhancement System.

We proposed an exceptionally brilliant concept and applied it to our current product.

I promise to you that this system is incredibly fancy and totally original.


Motivation
Crowd Sourcing
GPT, could you help me fix the coding error?
OK, in your code……
GPT, you're doing it wrong!
Sorry, I made a mistake, it should be……
GPT, could you help me fix the coding error?
OK, in your code……
GPT, WHITE has corrected you! It should be……
Sorry, I made a mistake, it should be……

Before introducing our project, I'd like to explain the motivation behind this.
（点击）

OK, here we showcase some common scenarios in our daily life.
In fact, AI assistants like GPT have become significant in our lives.
However, in scenarios like this, errors made by GPT can be incredibly frustrating
Indeed, this is a common issue with all LLMs, or rather, a trend for future development.
So, the key question is: how do we modify the model?

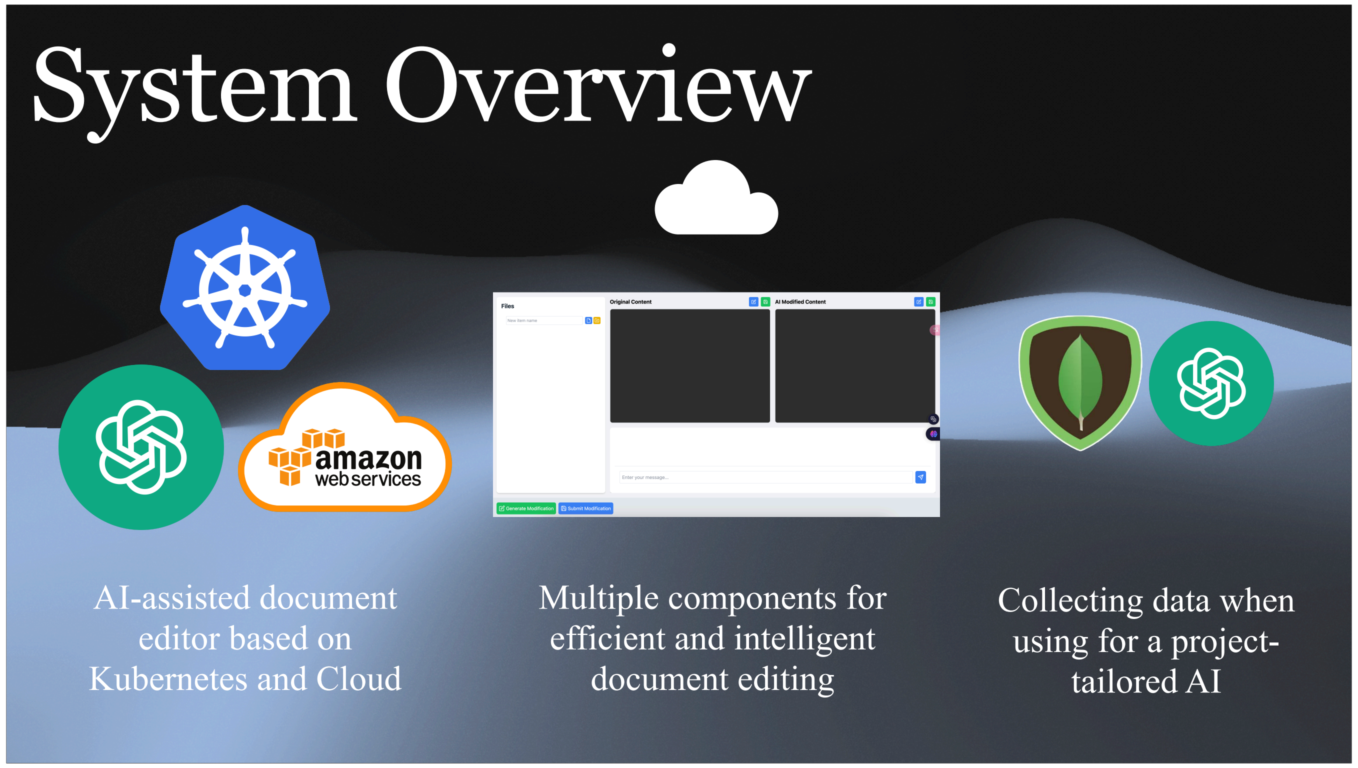
We believe the answer lies in data, which perfectly fits our team name. Data has become almost the most crucial factor determining the effectiveness of Large Models.
(点击)

So, where does the data come from?
（点击）

They are from human activities!

And this is what we call “crowdsourcing”. It can gather huge amount of data from people's local writing and development processes.

So why don’t we focus on the collection and reuse of data?
This is the reason and inner motivation of us.
（点击）



System Overview

AI-assisted document editor based on Kubernetes and Cloud

Multiple components for efficient and intelligent document editing

Collecting data when using for a project-tailored AI

(点击)
Then, I'd like to introduce our project starting with the system overview.

（点击）
Our project is divided into three parts:

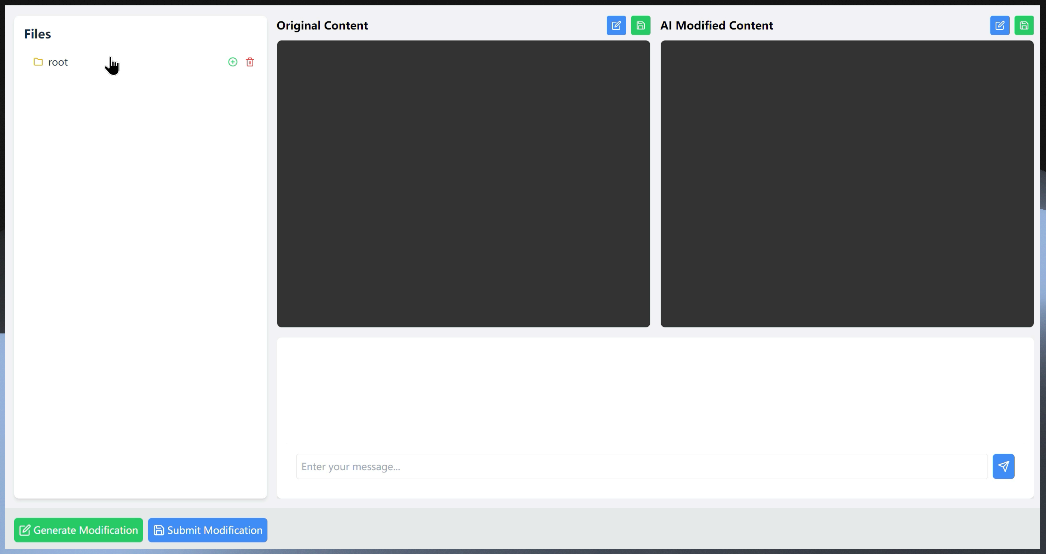
（点击）AI-assisted document editor based on Kubernetes and Cloud

（点击）Multiple components for efficient and intelligent document editing

（点击）Collecting data when using AI.

（点击）

Demonstration



https://github.com/Cloud-Computing-Group-NUS/Project-Code

100.24.7.54:32478

Here is our WebUI demonstration of the whole system

We prepared a video demo and we offer you a chance to explore the platform by your self at the end of slides

(点击视频播放)

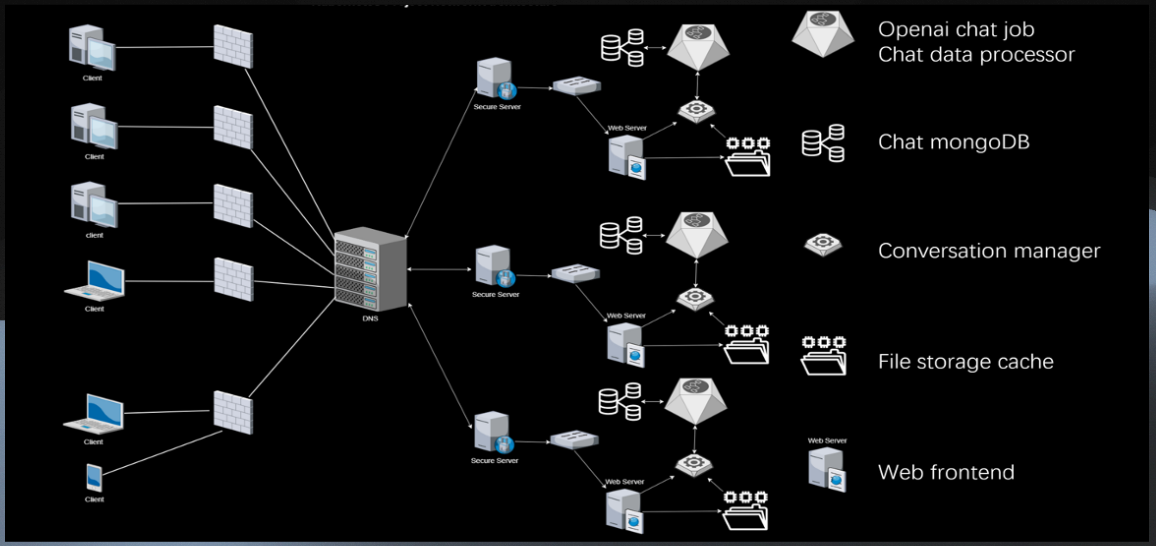
Here we go!

OK, the video is over and here we give oue the link, you can experience now if you can not wait

As you can see in the video,
- the left side corresponds to file interactions with cloud storage.
    - Users can create, delete, modify, and save FileObject here.
- The two browser on the right
    - Display the content of the user-selected file
    - and AI-modified file content awaiting user’s requirements.

- And here we go to the bottom side
    - This is the ChatBot just like GPT4
    - There are 2 buttons
        - One is aim to generate AI-modified file content

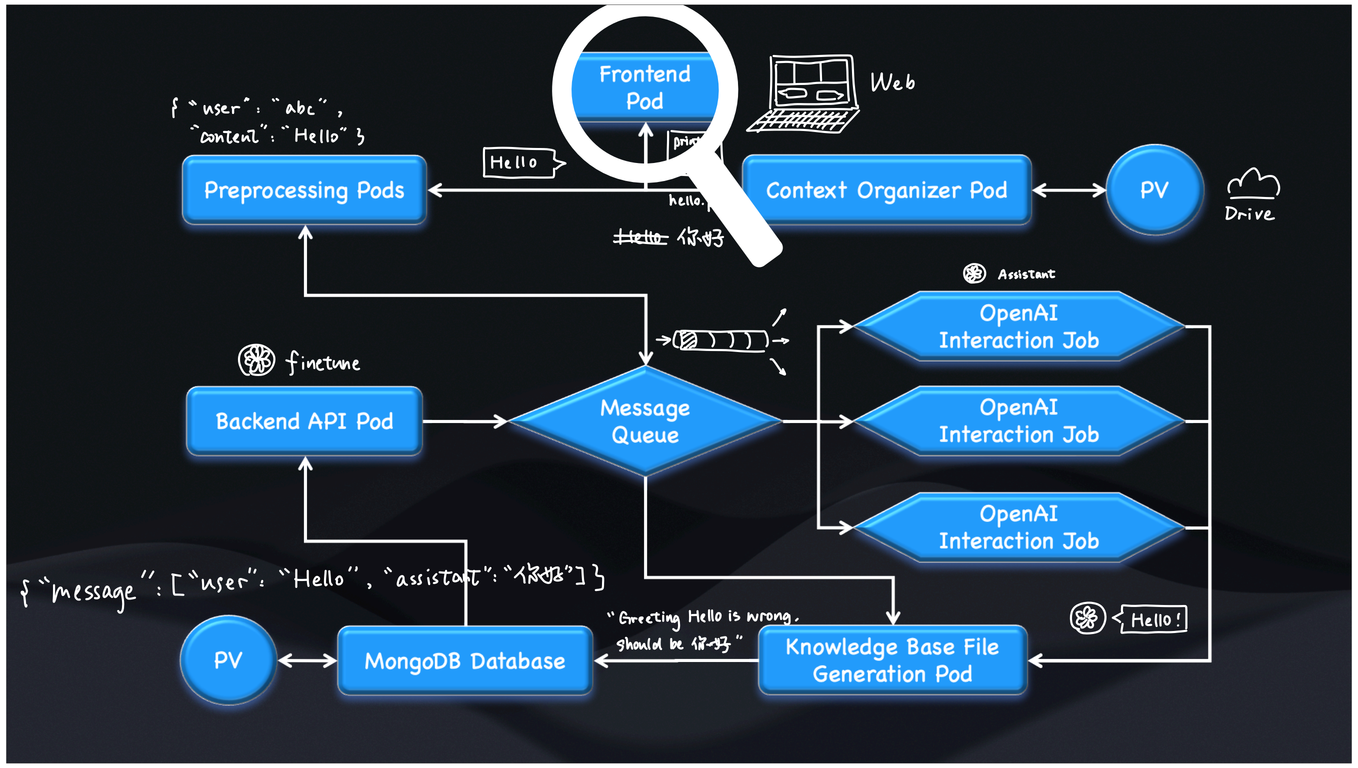
- Another is to submit corresponding modifications to the database.



Overal pipeline

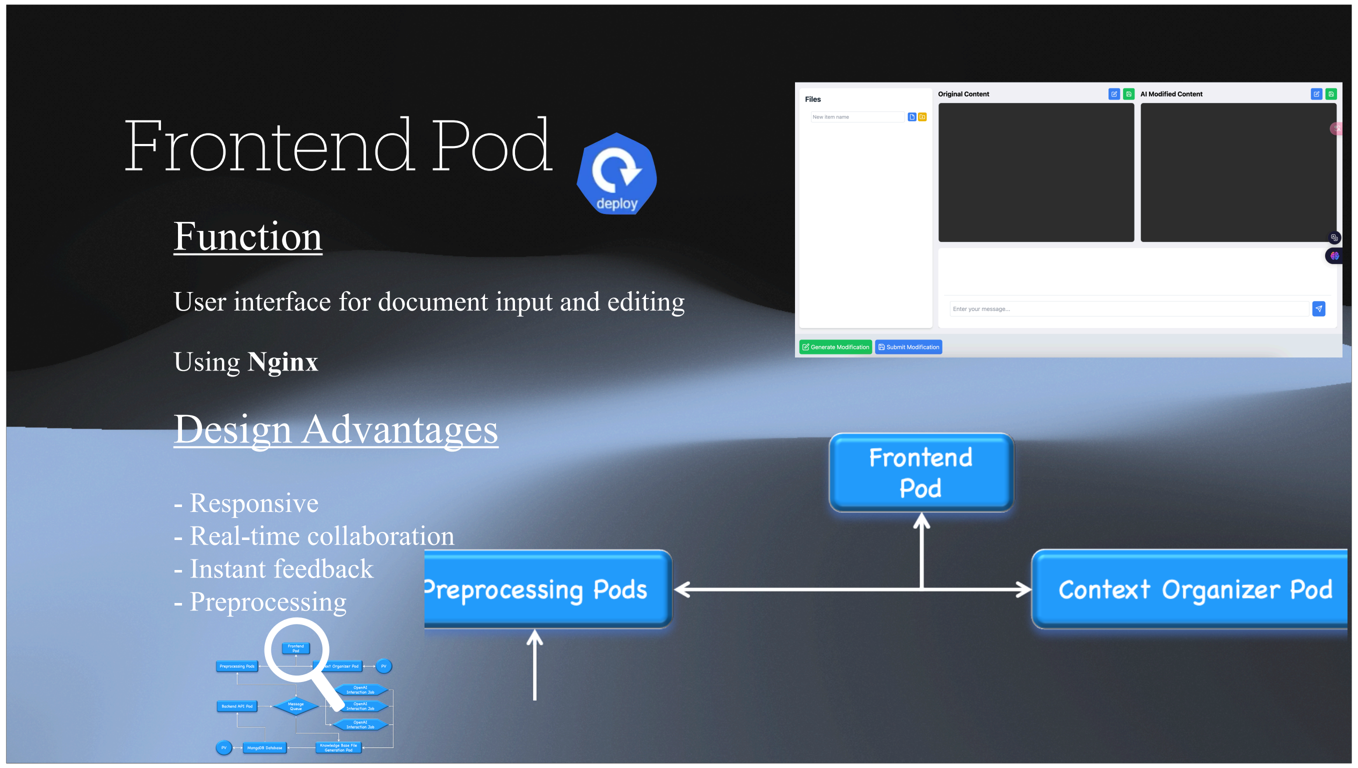
The whole system is like this , consisting of numerous fancy parts and they will be explained and analyzed in the following parts

(add)



And here we offer a much more spectacular glimpse of our system.

Let's follow the data flow throughout the whole project to go deeper into it step by step



Frontend Pod

Function

User interface for document input and editing

Using **Nginx**

Design Advantages

- Responsive
- Real-time collaboration
- Instant feedback
- Preprocessing

deploy
Files
Original Content
AI Modified Content
Generate Modification
Submit Modification
Frontend Pod
Preprocessing Pods
Context Organizer Pod

The first pod, which is also the most labor-intensive one in the entire project, is our frontend pod.

(点击)

It provides User interface for document input and editing.

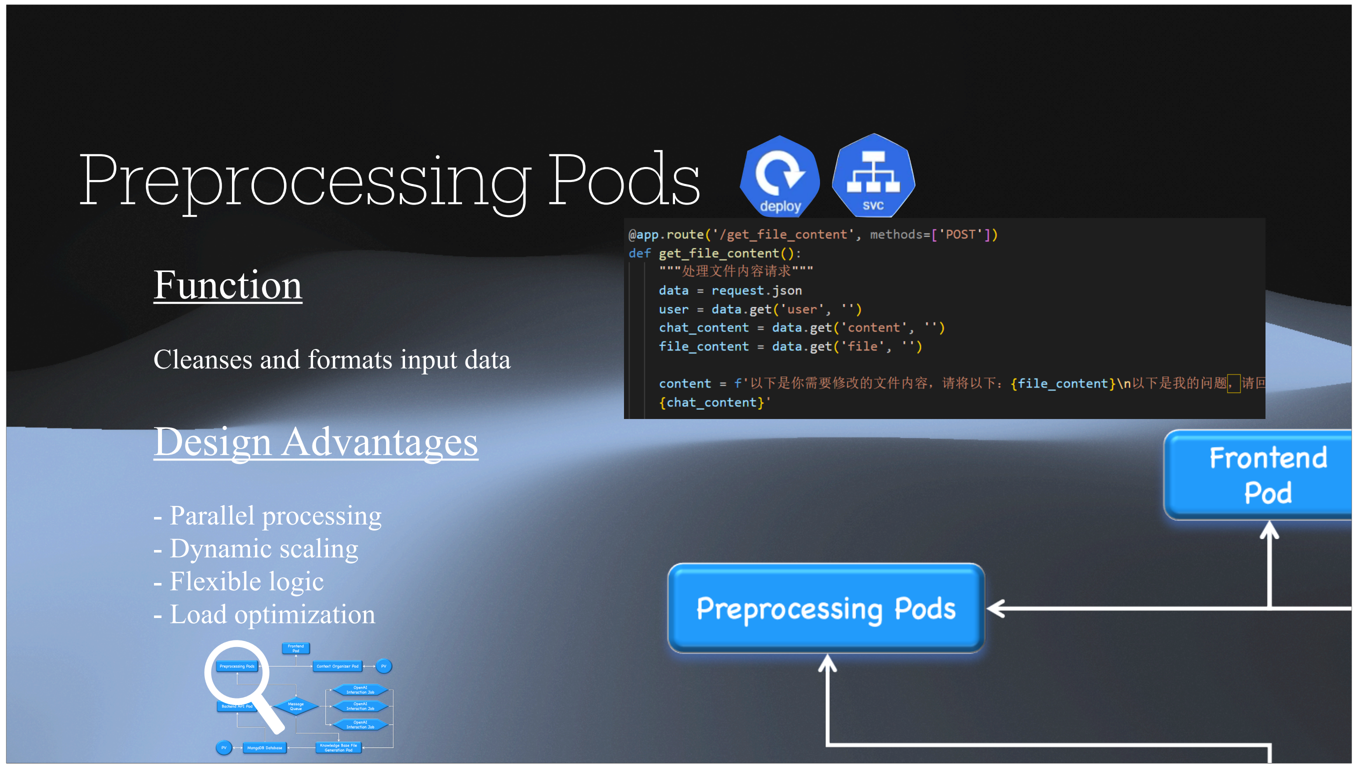
The detailed message about how to use it is briefly introduced in previous video and it’s much more fancy if you explore it by yourself.

I bet it’s incredibly magic!

Actually, we met some problems at first.

Users access the website through their own browsers, but our entire frontend is connected to a backend deployed with k8s.

Without forwarding improvements, direct access to the backend from the browser will be denied. So, we used NGINX for forwarding.



Preprocessing Pods

Function

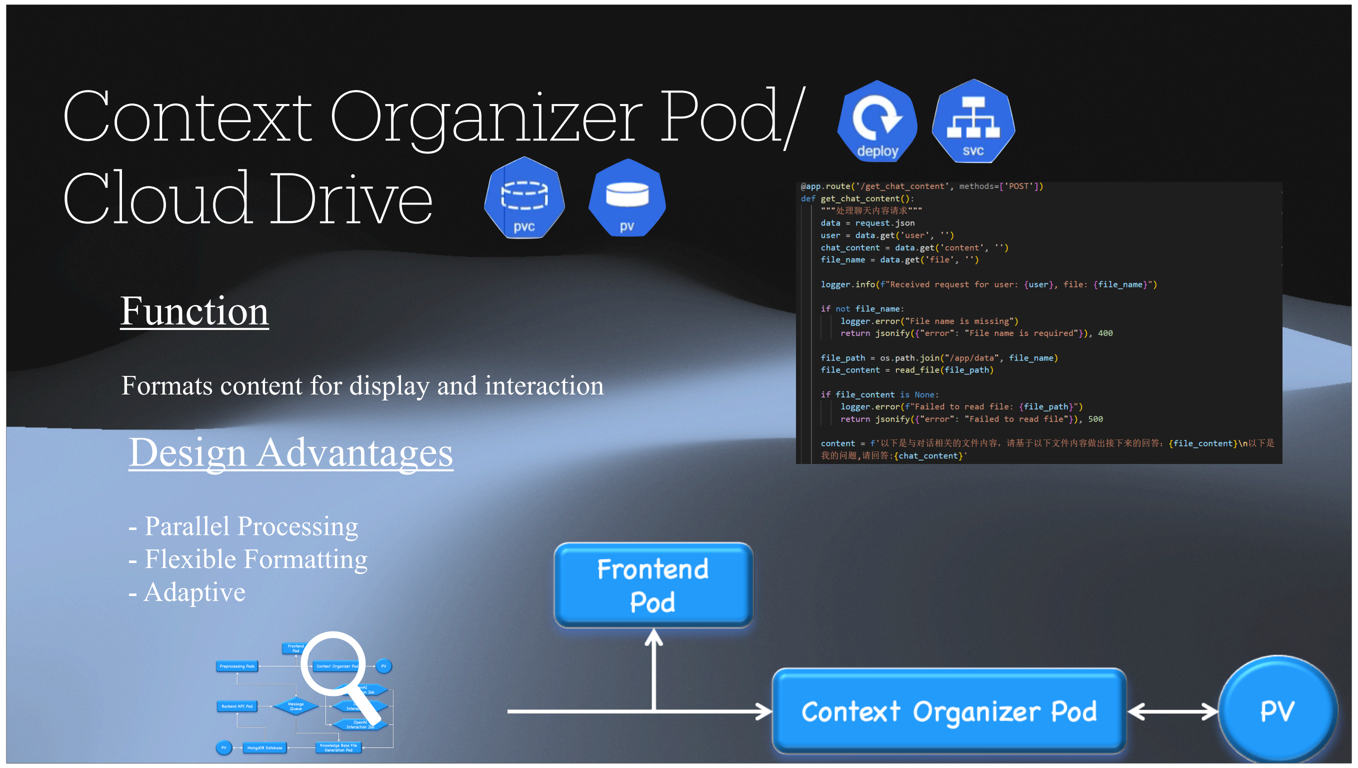
Cleanses and formats input data

Design Advantages

- Parallel processing
- Dynamic scaling
- Flexible logic
- Load optimization

```python
@app.route('/get_file_content', methods=['POST'])
def get_file_content():
    """处理文件内容请求"""
    data = request.json
    user = data.get('user', '')
    chat_content = data.get('content', '')
    file_content = data.get('file', '')

    content = f'以下是你需要修改的文件内容，请将以下：{file_content}\n以下是我的问题，请回
    {chat_content}'
```

（换zsy）（点击两下） The next pod is the preprocessing pod, whose function is to cleanse and format input data. For scenarios where numerous users input simultaneously, we need to maintain a sufficiently robust and quick-processing structure to ensure the stability of the entire system. This guarantees that each user's input can be instantly responded to and loaded into the storage queue, ultimately scheduling resources to process this request as quickly as possible. This pod also serves the function of load balancing and intelligent scheduling.



Context Organizer Pod/ Cloud Drive

deploy
svc
pvc
pv

Function

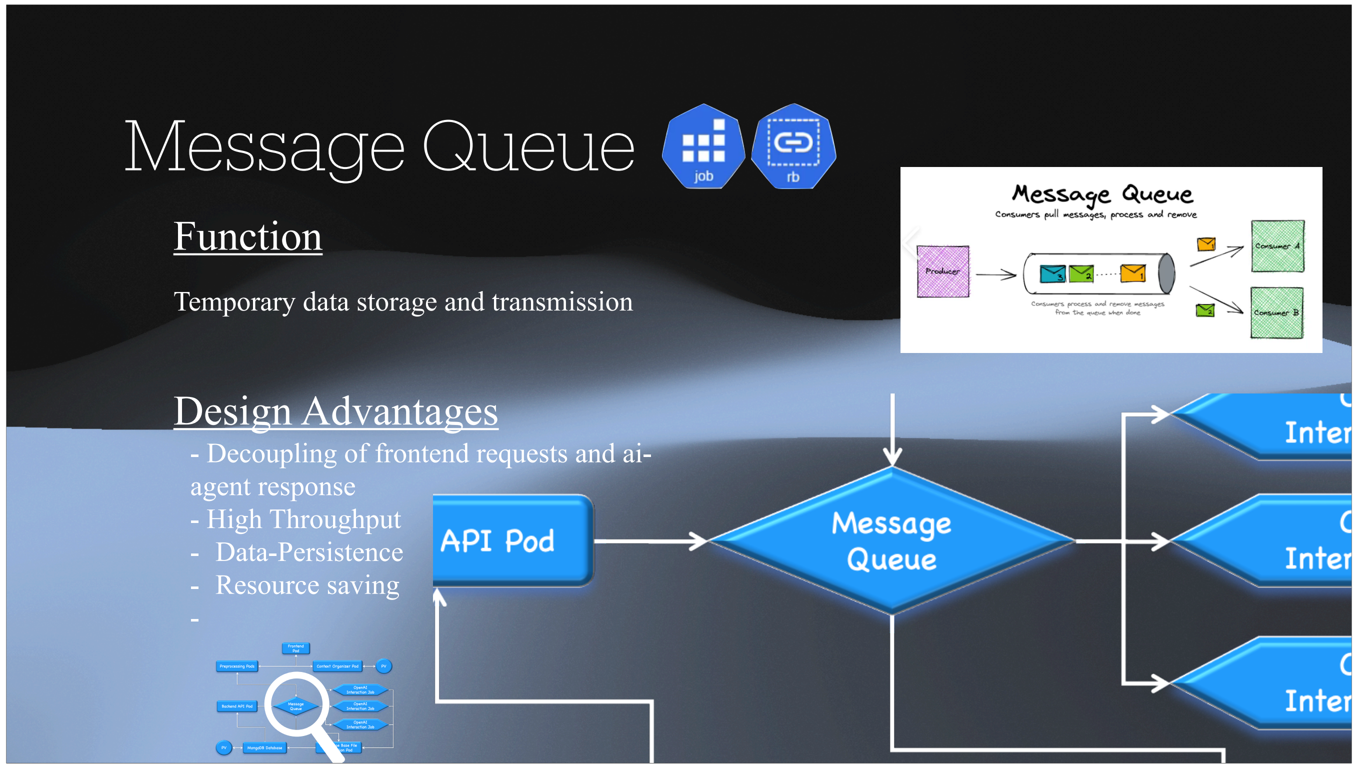
Formats content for display and interaction

Design Advantages

- Parallel Processing
- Flexible Formatting
- Adaptive

```python
@app.route('/get_chat_content', methods=['POST'])
def get_chat_content():
    """处理聊天内容请求"""
    data = request.json
    user = data.get('user', '')
    chat_content = data.get('content', '')
    file_name = data.get('file', '')

    logger.info(f"Received request for user: {user}, file: {file_name}")

    if not file_name:
        logger.error("File name is missing")
        return jsonify({"error": "File name is required"}), 400

    file_path = os.path.join("/app/data", file_name)
    file_content = read_file(file_path)

    if file_content is None:
        logger.error(f"Failed to read file: {file_path}")
        return jsonify({"error": "Failed to read file"}), 500

    content = f'以下是与对话相关的文件内容，请基于以下文件内容做出接下来的回答：{file_content}\n以下是我的问题,请回答:{chat_content}'
```

Another data flow is directed to the file-related POD, which we call the context organizer pod. Its main function is similar to cloud storage, but its design ensures that modifications to the same file by different users are correctly processed in sequence. Additionally, all resources are shared in real-time. Although we encountered some minor issues during implementation, causing occasional display problems in our interface, at least synchronization is achieved. (笑) The PV established here is to store the content of the cloud storage in real-time to disk resources, preventing loss of cloud storage content.



Message Queue

Function

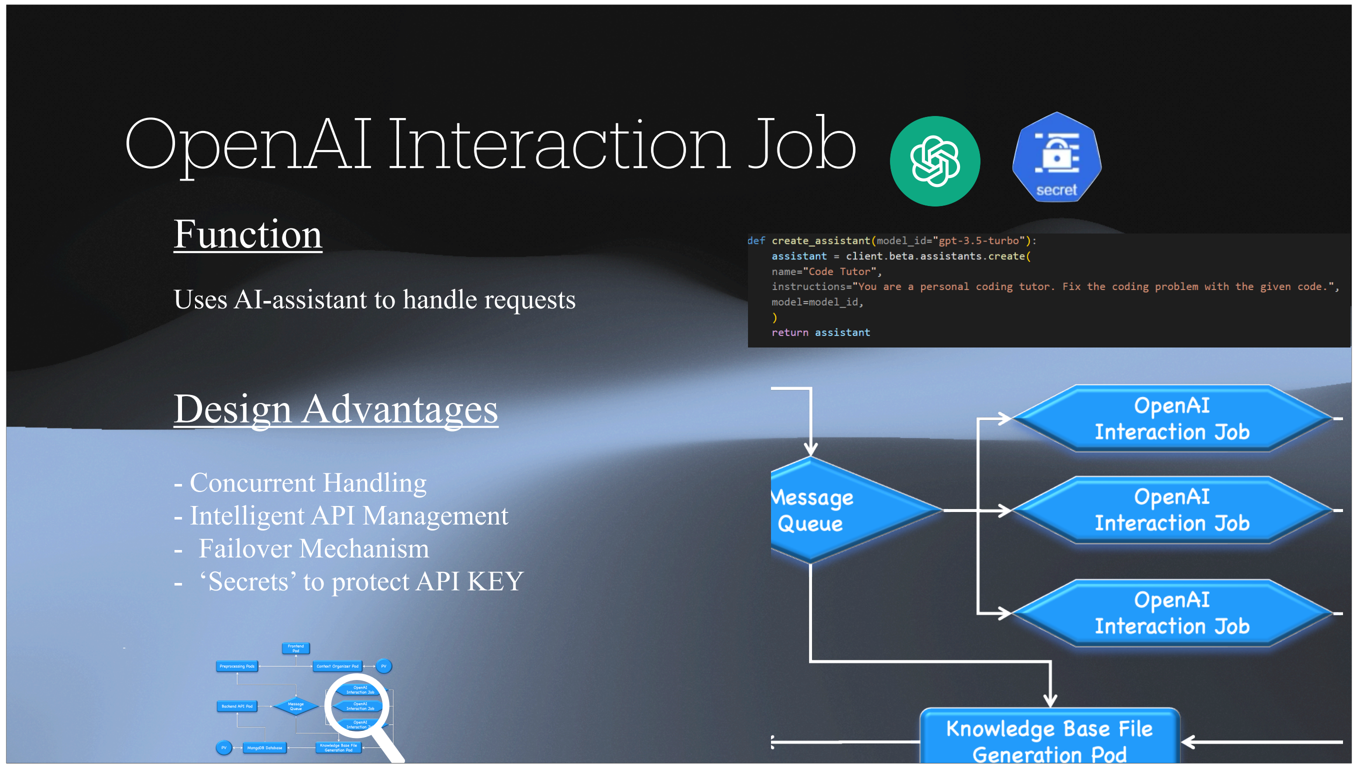
Temporary data storage and transmission

Design Advantages

- Decoupling of frontend requests and ai-agent response
- High Throughput
- Data-Persistence
- Resource saving
-

job
rb
Message Queue
Consumers pull messages, process and remove
Producer
Consumer A
Consumer B
Consumers process and remove messages from the queue when done
API Pod
Message Queue
Inter
Inter
Inter

（点击） The next part is the message queue, which is one of the most interesting design aspects of our entire project. Its function is simply temporary data storage and transmission, but in the Kubernetes environment, it also takes on the role of creating new jobs. We implemented this queue functionality ourselves and maintained the contextual order. When resources allow, it processes corresponding requests for each user as quickly as possible. We will introduce this part in more detail later.



OpenAI Interaction Job

<u>Function</u>

Uses AI-assistant to handle requests

<u>Design Advantages</u>

- Concurrent Handling
- Intelligent API Management
- Failover Mechanism
- ‘Secrets’ to protect API KEY

```
def create_assistant(model_id="gpt-3.5-turbo"):
    assistant = client.beta.assistants.create(
    name="Code Tutor",
    instructions="You are a personal coding tutor. Fix the coding problem with the given code.",
    model=model_id,
    )
    return assistant
```

（换xxy）

So i will travel you through the rest of our pipeline. Next stop is OpenAI interaction job.

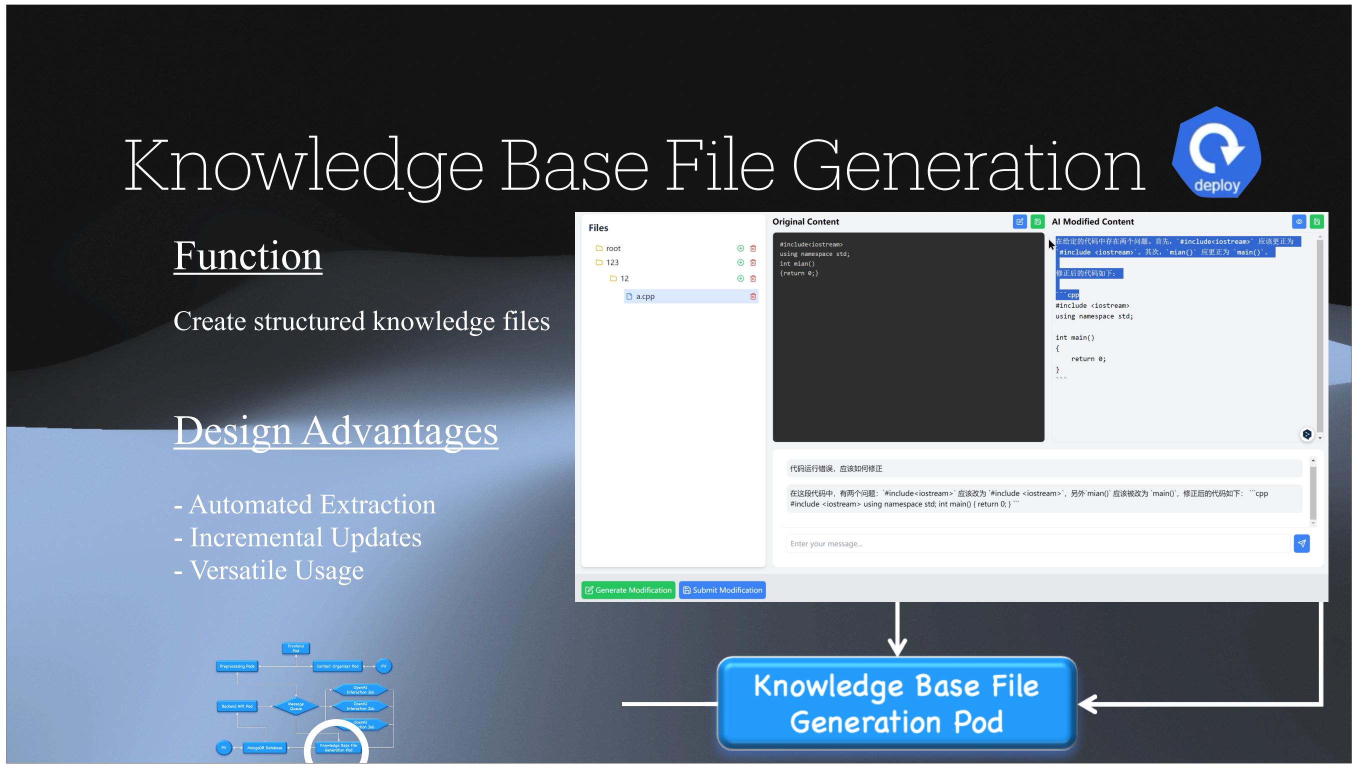
OpenAI interaction Job is responsible for forwarding message distributed by the message queue to a given OpenAI assistant and reply with gpt’s response.

For each job, it's lifecycle begins from a new webpage is opened , where exactly a unique id is assigned to a user, and then mapped to a particular assistant and a exclusive thread . when the webpage is closed, the job, assistant, and thread are deleted.

During its lifecycle, it creates an assistant and a thread to maintain multi-turn conversation.

Due to the message queue design, multiple agents can answer users' requests on different devices at the same time.

Also, secrets is implemented to protect your OpenAI key.



Knowledge Base File Generation

Function

Create structured knowledge files

Design Advantages

- Automated Extraction
- Incremental Updates
- Versatile Usage

Original Content

```
#include<iostream>
using namespace std;
int mian()
{return 0;}
```

AI Modified Content

在给定的代码中存在两个问题。首先，`#include<iostream>` 应该更正为 `#include <iostream>`。其次，`mian()` 应更正为 `main()`。

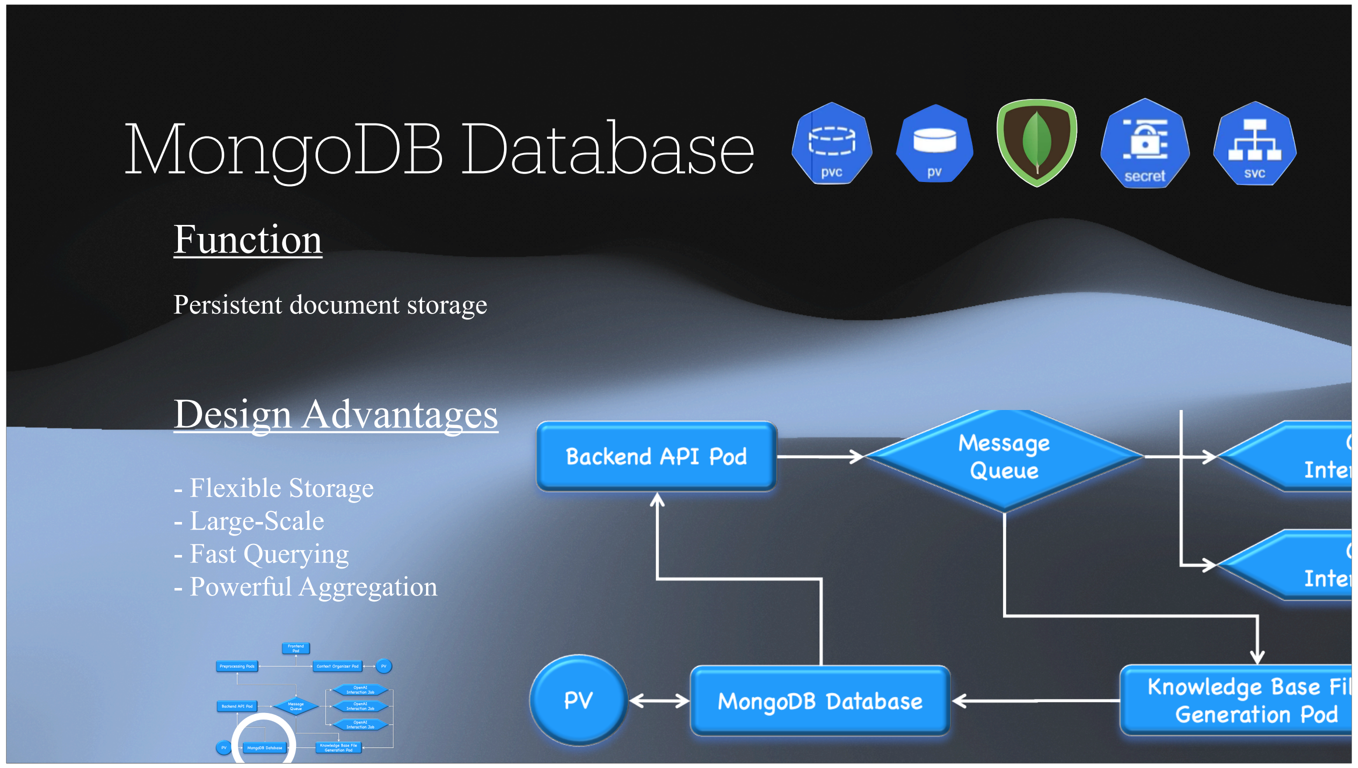
修正后的代码如下：

```cpp
#include <iostream>
using namespace std;

int main()
{
    return 0;
}
```

代码运行错误，应该如何修正

在这段代码中，有两个问题：`#include<iostream>` 应该改为 `#include <iostream>`，另外`mian()` 应该被改为 `main()`，修正后的代码如下： ```cpp #include <iostream> using namespace std; int main() { return 0; } ```

Knowledge Base File Generation synthesizes the code generated by OpenAI and your modified code, format them into certain json format and save the message into MongoDB database.



MongoDB Database

pvc
pv
secret
svc
Backend API Pod
Message Queue
Inte
Inte
PV
MongoDB Database
Knowledge Base Fi
Generation Pod

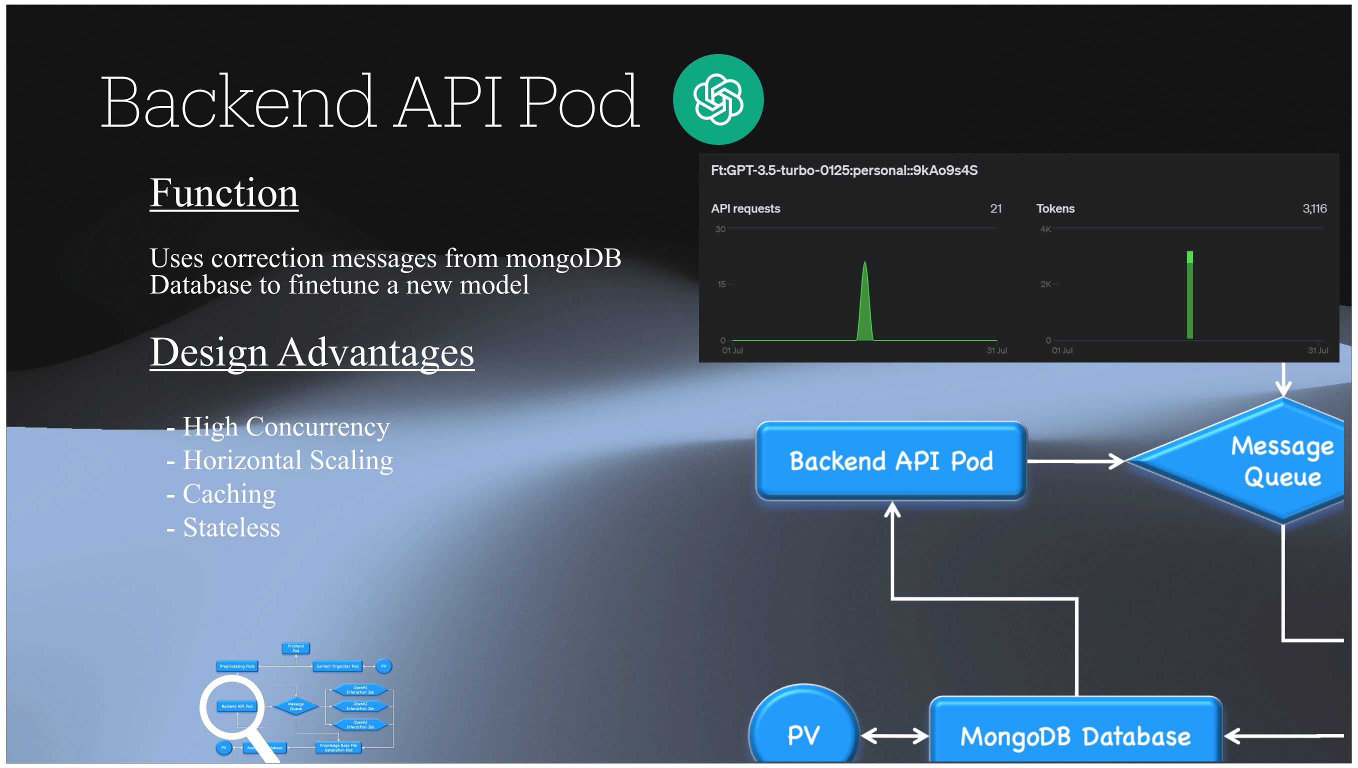
Function

Persistent document storage

Design Advantages

- Flexible Storage
- Large-Scale
- Fast Querying
- Powerful Aggregation

（点击两下） To address the storage needs of user-generated training data, our project uses MongoDB operator as a method for storing documents. Its main role is to provide persistent document storage. Its characteristics of flexibility, large-scale capacity, and fast querying perfectly meet our needs for parallel collection and organization of training data.



Backend API Pod

Function

Uses correction messages from mongoDB Database to finetune a new model

Design Advantages

- High Concurrency
- Horizontal Scaling
- Caching
- Stateless

Ft:GPT-3.5-turbo-0125:personal::9kAo9s4S

API requests 21

Tokens 3,116

Backend API Pod
Message Queue
PV
MongoDB Database

Backend API Pod uses correction messages from mongoDB Database to finetune a new model. The whole procedure is like customing our own finetune dataset. With more modification of more people,
the ai understands better our project target, which is indeed our idea of crowd sourcing.

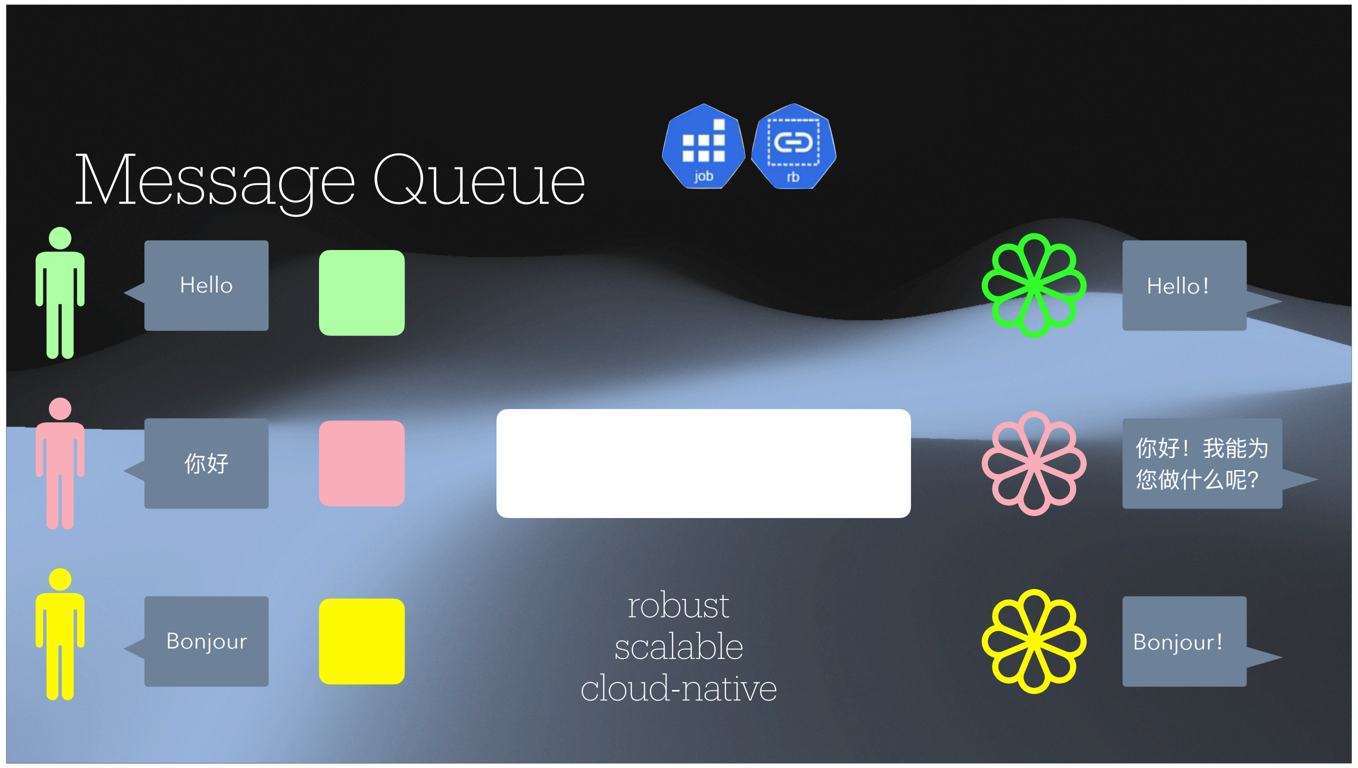
In essence, Backend API Pod and OpenAI Interaction Job are connected on a cloud provided by OpenAI. Therefore, OpenAI Interaction job can automatically pull the latest version of finetuned model, providing latest tailored service behind the scene.

Also, notice that finetune is indeed costly. Look at the picture on the top right corner. One basic finetune procedure(10 messages) with 100 epochs costs 0.02 USD, which amounts to the total test messages we've sent during our whole development project. You can see that a simple finetune costs 3k tokens.

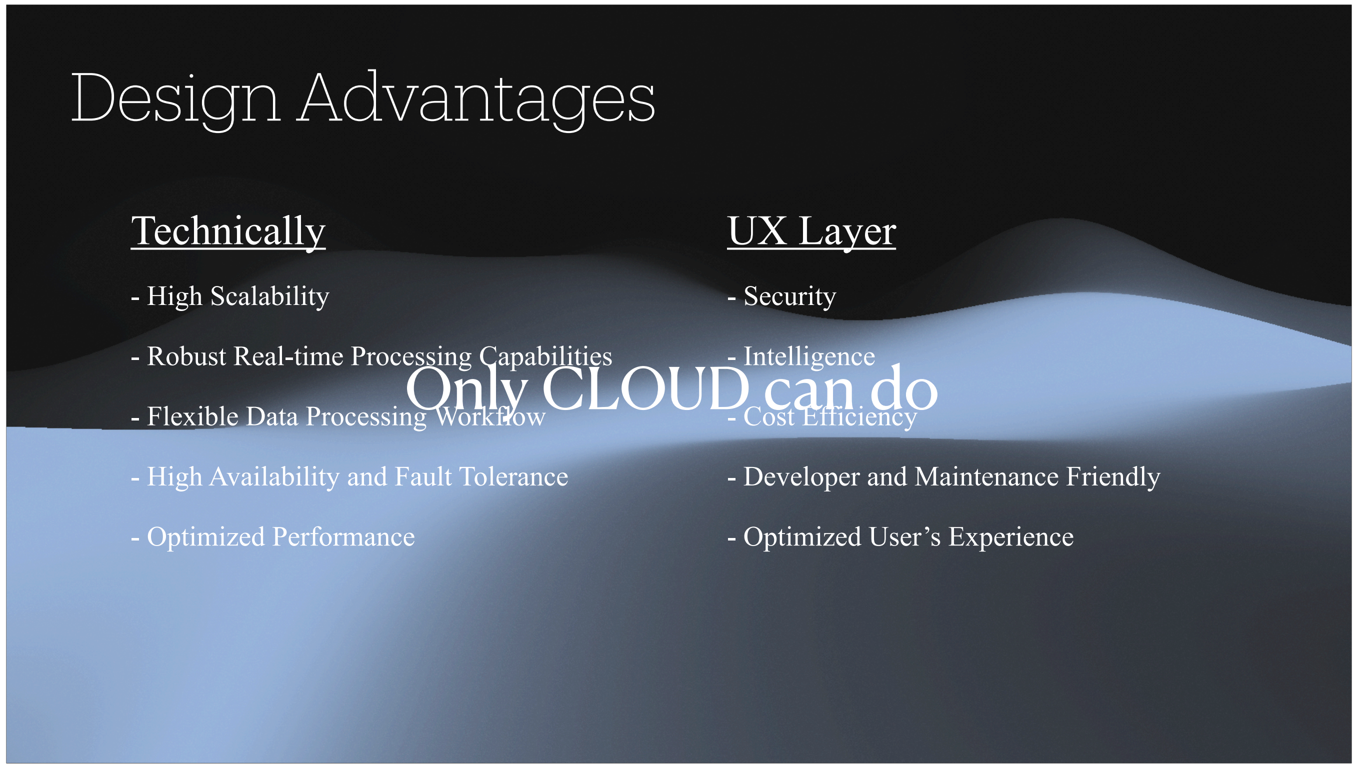
Fancy Design

Message Queue

Next, given that our course revolves around Kubernetes, we want to focus on the message queue part, which utilizes the most Kubernetes features. （点击）



First, when multiple users simultaneously make conversation requests, （点击）the message queue sorts all requests in chronological order and maintains a job for each user. （点击）It continuously forwards each user's session to the corresponding job for processing. When a user closes the interface, it also closes the job corresponding to that user. In other words, this pod completes the creation of jobs, content processing and forwarding, and final closure operations. This is achieved by calling the Kubernetes API. To give this pod all the relevant permissions, we used RBAC (Role-Based Access Control), which is a mechanism for managing access permissions to the Kubernetes API. It allows administrators to dynamically configure access policies. Through these methods, we have implemented fine-grained control arrangements, maximizing the parallel advantages of Kubernetes, thereby ensuring a good user experience.



Design Advantages

Technically

- High Scalability
- Robust Real-time Processing Capabilities
- Flexible Data Processing Workflow
- High Availability and Fault Tolerance
- Optimized Performance

UX Layer

- Security
- Intelligence
- Cost Efficiency
- Developer and Maintenance Friendly
- Optimized User's Experience

Only CLOUD can do

In conclusion, the deliberate design of our project lies in two aspects.

Technically, it has high scalability: with independent scaling, automated horizontal scaling and data-level scaling.

It has robust real-time processing capabilities with stable web socket connections, high-throughput streams and parallel preprocessing.

It has high availability and fault tolerance, with all key pods deployed on multi-instances, and a Persistent Volume maintaining the finetune database as well as the Cloud Drive. Due to pod's property of self-recovery, it has error recovery ability.

It has optimized performance given that pods' lifecycle exists only when a front end tab exists

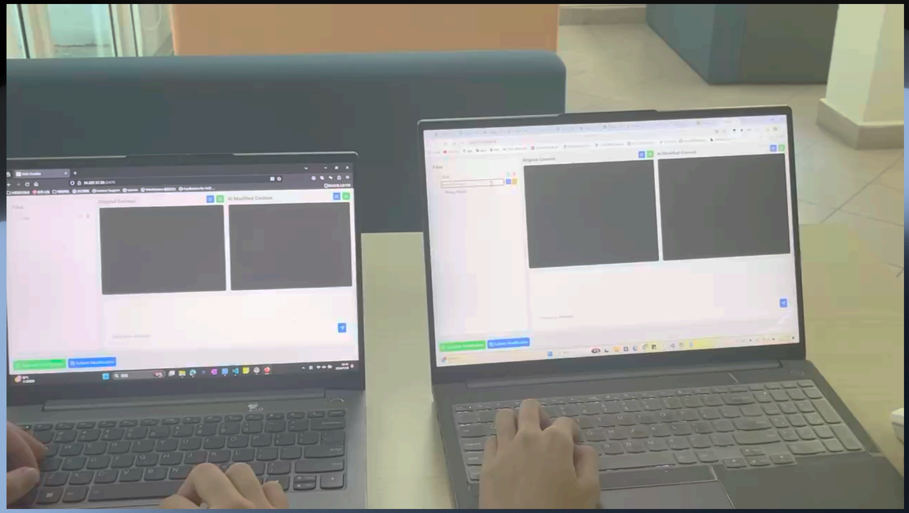
From the aspect of UX Layer design, it maintains high security with secrets implemented, Intelligence with a finetuned ai assistant, it is cost effective as few pods will stay idle but running.

It is incredibly developer and maintainance friendly, as we've containerized all the code and modularize our design.

Last but not least, it provides excellent user experience, with all fancy functions as real-time code rendering, AI assistance, clear and beautiful frontend webpage.

DEMO

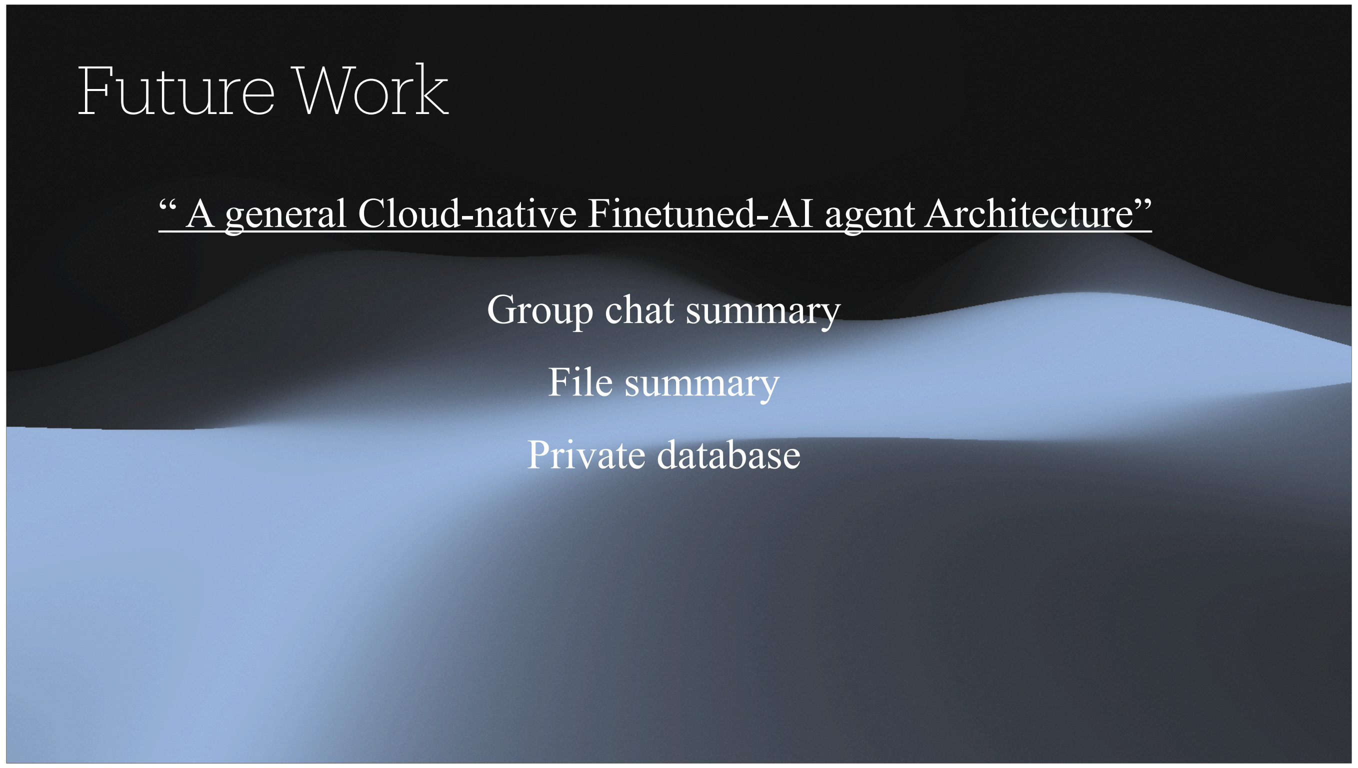
Supports up to 50(theoretically) collaborators online



HTTP: 100.24.7.54:32478

Warning:

- First create a folder of your name, then feel free to edit and create under your workspace
- For naming the folders, please follow the standard lowerCamelCase
- Don’t send too many requests at one time, as every token counts
- Open exactly one webpage at the same time
- First select a file, then ask questions related to the file
- Legal file types: .cpp, .py, .md(with additional rendering)
- First time set up may be slow, wait patiently
- Have fun!



Future Work

“ A general Cloud-native Finetuned-AI agent Architecture”

Group chat summary

File summary

Private database

Eventually, we'd like to extend a little and envision the broad application of our project.

To sum up, our project builds a persistent Memory (which is the cloud drive), a knowledge base for storing messages for finetuning, a message queue for multi-person collaboration, the architecture can extend to way more scenarios.

For instance, if you want a chat-bot to assist you in your work group, like summarizing hundreds of messages you've sent during a heated discussion, or telling you the latest deadline, saving your effort of scrolling through all the group files,
then the memory saves all the messages, and send to the message queue a summary request, which sends request to OpenAI and get the summary answer.

Thank you

https://github.com/Cloud-Computing-Group-NUS/Project-Code/tree/main

Group 6, DATA
SoC Summer Workshop