**Automated Image Rotation using ML Model**

Design Choice:

Implemented an ML model to automatically detect and correct the orientation of uploaded documents to ensure they are centered and properly aligned before processing. This is done because I observed that VLM was able to extract information accurately from the images which were oriented properly when compared to images which were not oriented properly

Trade-offs:

1. Increased accuracy but addition of latency and resource utilization.

2. Increased Processing Time: Introducing an additional ML model step adds to the overall processing time, potentially affecting user experience due to increased latency.

3. Increased Resource Utilization: Leveraging an ML model ensures high-quality image preprocessing. Thus, Running an ML model consumes more computational resources (CPU/GPU), which can impact scalability and increase operational costs.

**Note: Supervised Fine-Tuning of Vision-Language Model used will remove this overhead. But this will be an additional cost incurred for the company. – This Feature is not yet available at Firework ai. This feature is crucial because Vision-Language Prompt Learning (methodology to instruct VLM through prompt to analyze image irrespective of orientation) did not work as expected.**

**Manual rotation of images to correct its orientation:**

This additional step will make sure that the images are properly oriented to center before it is sent to VLM in case the automated rotation doesn't align it as expected. But this would be an overhead on user and might decrease the user experience but its an add on to the accuracy.

**Model Choice:**

1. Phi 3.5 Vision Instruct: Was chosen because its serverless and it need not be deployed, and it provides great results with the text extraction from the image. Even it has went through SFT and DPO. It would be great if Fireworks came up with SFT for users to fine-tune it based on their use case. And the cost of Phi 3.5 Vision Instruct for our use case was reasonable as it is billed only 576 prompt tokens per image input whereas were able to do the task of extract text from image properly.
2. Japanese Stabel VLM: Was not chosen as it won't align with our use case. It's used for generating Japanese descriptions for input images and optionally input texts such as questions.

3. Llama 3.2 11B Vision Instruct: Was not chosen because even though it's optimized for visual recognition, image reasoning, captioning, and answering general questions about an image. It's failing to extract the information from the image. And the cost is also high as it is billed 6400 prompt tokens per image input.
4. FireLLaVA-13B: Was not chosen as it's not serverless and it must be deployed. And I didn't feel the necessity to try it out as I was getting my job done through another model.

**Note: Since we are depending on Fireworks.ai API for our use case, we need to monitor API usage to manage costs effectively and develop fallback strategies in case of API downtime to maintain application reliability.**

**Containerization with Docker and Docker Compose**

Containerized the application using Docker to ensure consistent environments across development and production, and orchestrated services using Docker Compose.

**Scalability and High Traffic Handling**

This application is just designed to run locally with capability handle decent amount of high traffic volumes by leveraging containerization and scalable services.

I have implemented asynchronous capabilities which helps the application to process multiple requests concurrently, like making API calls or handling image uploads without blocking other operations. This significantly improves performance, especially under heavy load.

Trade-off :However, while async helps with I/O-bound tasks, it adds complexity, making the code trickier to manage and debug. Additionally, async doesn't benefit CPU-heavy operations, which might still create bottlenecks.

In summary, while asynchronous capabilities greatly enhance the scalability of our application for I/O-bound operations, they come with a trade-off in terms of added complexity and diminishing returns when dealing with CPU-heavy tasks.

When deployed in Production we must make sure to consider the below:

- Load Balancing: Implement load balancing to distribute traffic evenly across containers, enhancing performance and reliability.

- Auto-Scaling: Utilize auto-scaling features to dynamically adjust the number of running containers based on traffic patterns.
- Monitoring and Alerts: Set up monitoring tools to track performance metrics and trigger alerts for unusual traffic spikes or performance degradation.

**User Experience and Responsiveness**

Focused on delivering a seamless and intuitive user interface to enhance user experience by making using of user-friendly features and messages and used React JS, Tailwind CSS, Material UI to implement the same.