Models Comparison

Model	Parameters	Size	Download
Llama 3	8B	4.7GB	`ollama run llama3`
Llama 3	70B	40GB	`ollama run llama3:70b`
Phi 3 Mini	3.8B	2.3GB	`ollama run phi3`
Phi 3 Medium	14B	7.9GB	`ollama run phi3:medium`
Gemma 2	9B	5.5GB	`ollama run gemma2`
Gemma 2	27B	16GB	`ollama run gemma2:27b`
Mistral	7B	4.1GB	`ollama run mistral`
Moondream 2	1.4B	829MB	`ollama run moondream`
Neural Chat	7B	4.1GB	`ollama run neural-chat`
Starling	7B	4.1GB	`ollama run starling-lm`
Code Llama	7B	3.8GB	`ollama run codellama`
Llama 2 Uncensored	7B	3.8GB	`ollama run llama2-uncensored`
LLaVA	7B	4.5GB	`ollama run llava`
Solar	10.7B	6.1GB	`ollama run solar`

These are the models provided by Ollama and are of varied sizes. I have performed the entire project on the **moondream** model.

For the entire assignment I created a single EKS cluster in a single Availability Zone and deployed 4 nodes with varied instance types for testing purposes. (AWS allows deploying instances with cumulative vCPU = 4 for personal account). I implemented HPA and tested the model on different load testing environments.

Because of time constraints, I will not be performing comparison of all models, but I will convey the entire approach on how I would go about comparing different models.

Approach:

Step 1

I would create a Dockerfile and a start script specifying the model name to be used inside the script. Then I would also provide the model's name inside the API wrapper at the ollama.chat() method.

Step 2

I would build an image for this Dockerfile and push it to my Docker Hub account for public visibility and access.

Step 3

Next, I would create YAML manifests for K8s, deployment.yml, service.yml and hpa.yml.

Step 4

Since some of the models are of very huge sizes, I would deploy 4 different EKS clusters in 4 different regions for high availability and scalability. For example: 1 cluster in us-west-1, 1 in us-east-1 and 1 in us-west-2 and 1 in us-east-2.

Depending upon the users' location, I would deploy the clusters in those regions as well.

Step 5:

Next, I would ensure that I have **kubectl** configured for each cluster and then deploy the application to each one.

Step 6

Post deployment of the application, there would be 4 different load balancers ready to intercept requests. I would set up a **Global Load Balancer** and configure **Amazon Route 53** to have a single endpoint to receive requests.

I would set up a latency-based routing policy on Route 53. This will route traffic to the closest ALB based on latency, providing a single endpoint for users.

Step 7

Now, I would make curl requests to the model using different prompts and check for its efficiency and accuracy.

Step 8

Lastly, I would use K6.io to perform load testing that would simulate different levels of concurrent requests and implement different test scenarios (e.g., gradual ramp-up, spike tests, endurance tests)

Repeat each step for different model and find out the one with the best performance :)