Take-Home Assignment: LLM Benchmark Simulation and Infrastructure Deployment

Objective:

You are tasked with creating a tool that benchmarks the performance of various Language Learning Models (LLMs) against the quality metrics listed below. Your solution should simulate and record results for each metric, rank the LLMs, and expose the ranking via an API endpoint. Additionally, you will deploy your solution as microservices in a distributed environment using infrastructure as code.

Part 1: LLM Benchmark Simulation

Overview:

Your solution should simulate and generate data for at least 3 LLM models against the following metrics:

- **Time to First Token (TTFT)**
- **Tokens Per Second (TPS)**
- **End-to-End Request Latency (e2e_latency)**
- **Requests Per Second (RPS)**

LLMs to Consider:

- GPT-4o
- Llama 3.1 405
- Mistral Large2
- Claude 3.5 Sonnet
- Gemini 1.5 Pro
- GPT-4o mini
- Llama 3.1 70B
- amba 1.5Large
- Mixtral 8x22B
- Gemini 1.5Flash

- Claude 3 Haiku
- Llama 3.1 8B

Requirements:

1. **Metric Simulator:**
    - Design a database model to store the simulation results for each LLM against each metric.
    - Write a randomizer that generates 1,000 data points for each LLM against each given metric.
    - Store the simulation results for each LLM against each metric in your database.
2. **Metric Benchmarking:**
    - Fetch the simulation results for each metric and rank the LLMs based on their mean values.
    - Expose an API endpoint that returns the ranking of LLMs for a given metric.
    - Ensure the latency of the endpoint is less than 2 seconds.
3. **Frontend**
    - Implement a simple web dashboard or use an existing tool like Grafana to visualize the benchmark results and rankings.
    - Include various charts (e.g., bar charts, line graphs) to show how each API, LLM, Provider, etc performed across different metrics. This can help users quickly grasp the comparative performance of the models.
    -
4. **Non-Functional Features:**
    - The system should be robust and fault-tolerant.
    - The solution should be scalable, capable of handling additional metrics, LLMs, and more APIs with ease.
5. **Documentation:**
    - Provide a README file with instructions on how to run and use the app.

Bonus Points:

- Implement retry logic or use a message queue to handle spikes in requests or failed operations.
- Include a mechanism to seed the randomizer, allowing for reproducibility in simulations.
- Consider securing the API with API keys or OAuth for real-world scenarios.

Part 2: Infrastructure Deployment

Overview:

You will implement Infrastructure as Code (IaC) to deploy your solution as microservices in a Kubernetes cluster using Helm charts.

Requirements:

1. **Helm Charts:**
    - Write Helm charts for deploying the microservices onto a Kubernetes cluster.
    - The Helm charts should include configurations for services, deployments, and necessary networking components (e.g., Ingress).
2. **Cluster Setup:**
    - Provide instructions for deploying the microservices onto a Kubernetes cluster.
    - The deployment should scale, handle configuration via Helm, and be easily reproducible.
3. **Bonus Points:**
    - Implement CI/CD pipelines to automate the deployment process using tools like GitHub Actions, Jenkins, or GitLab CI.
    - Ensure monitoring and logging are in place for the deployed services.

Submission:

- Submit your code via a Git repository (GitHub, GitLab, etc.) with a clear README file.
- Include Helm charts, deployment instructions, and any additional scripts required to set up the infrastructure.
- Provide sample output from the LLM simulation tool for review.

Evaluation Criteria:

- **Correctness:** Does the simulation tool generate accurate and consistent results?
- **Scalability:** Can the infrastructure handle scaling the microservices efficiently?
- **Code Quality:** Is the code clean, well-documented, and easy to maintain?
- **Infrastructure as Code:** Is the Helm chart setup reproducible and effective for deploying the microservices?
- **Bonus:** Automation of the deployment process through CI/CD and additional monitoring/logging setups.

References:

- For inspiration, you can refer to Artificial Analysis AI.
- For better understanding of the metrics visit NVIDIA LLM Benchmarking Metrics

---