Justification for Cost Evaluation and Resource Management

As a team, we implemented multiple cost-conscious strategies while deploying and maintaining our cloud-based ML application on Azure. The following points provide evidence of our ability to evaluate financial costs and optimize resource usage without compromising performance or system integrity:

- ## Selection of Cost-Efficient Resources

  We deployed our model to a Kubernetes Online Endpoint using a single instance (**instance_count=1**) and the **defaultinstancetype** on the **adsai-lambda-2** compute target. This configuration was selected to reduce operational cost while being sufficient for expected inference load. Instead of scaling up compute resources unnecessarily, we focused on optimizing deployment logic and inference efficiency.

- ## Use of Blue-Green Deployment for Risk Mitigation and Cost Efficiency

  Among all deployment strategies, we opted for **Blue-Green Deployment** because it offered the best balance between risk control and cost. This approach allowed us to deploy a new model version (green) alongside the existing one (blue), test its functionality in a live environment, and only switch traffic when fully validated. Unlike Shadow or A/B deployments—which require running multiple models simultaneously under production load—Blue-Green kept cost minimal by allowing only one deployment to actively serve traffic at any time. This ensured that compute costs were not doubled, yet operational risk was effectively mitigated.

- ## Avoidance of Big Bang and Canary Risks

  We intentionally avoided Direct (Big Bang) Deployment, which, although low-cost, carries a high risk of system failure if the new model is faulty. Similarly, Canary Deployment would require complex traffic routing and monitoring logic, adding unnecessary overhead for our project's scope. Blue-Green gave us confidence in rollback safety while avoiding extended periods of duplicate resource usage.

- ## Scripted and Automated Deployment for Cost Control

  We created custom deployment scripts (**create_endpoint.py, create_green_deployment.py, switch_traffic_to_green.py**) to ensure reproducibility and avoid costly misconfigurations. These scripts handled everything from registering models to updating traffic allocations, minimizing manual errors and deployment overhead.

- ## Real-Time Logging Instead of External Monitoring Tools

  We leveraged Azure's built-in log retrieval tools (**ml_client.online_deployments.get_logs**) instead of third-party monitoring services. Inference behavior was tracked using printed logs in **score.py**, providing transparency into tokenization, model output, and prediction formatting—without additional costs.

- ## Model and Environment Reuse
  Instead of creating new versions of models or environments for each iteration, we reused our previously registered **nlp8_model** and **nlp8-env2**. This avoided excess storage usage and eliminated unnecessary retraining or reconfiguration.

- ## Problem Identification Without Wasting Compute
  During one debugging phase, we encountered a persistent 504 Gateway Timeout error. Instead of redeploying blindly, we methodically reviewed the endpoint logs, verified model loading, updated the **score.py** script for efficiency, and tested locally. This analytical approach prevented redundant deployments and compute waste.

- ## VPN Handling to Prevent Redundant Debugging
  We also discovered that VPN configuration affected endpoint access. By identifying this early, we avoided repeating diagnostics or mislabeling functional deployments as failures—preventing unnecessary reruns or reconfigurations.

- ## Time Invested in Budget-Conscious Decisions
  We dedicated around 10 hours to reviewing and comparing deployment strategies, adjusting compute settings, testing traffic switching logic, and writing automated scripts—all with the goal of delivering a low-cost yet reliable deployment setup.

In summary, we demonstrated a strong ability to balance performance and budget constraints in the cloud. By choosing the **Blue-Green Deployment** strategy, automating deployments, and using minimal compute resources, we successfully deployed a functional, scalable, and cost-effective ML inference system.