Sure, here is the detailed report based on the code and the requirements provided.

---

# NLP Group Coursework Report

## 1. Research Different Model Serving Options

### Research and Choices

In our project, we considered several model serving options for deploying our NLP model. The options included:

- \*\*Flask\*\*: A lightweight web application framework in Python. It is ideal for small to medium-sized applications.

- \*\*FastAPI\*\*: A modern, high-performance web framework for building APIs with Python 3.6+ based on standard Python type hints.

- \*\*TensorFlow Serving\*\*: A flexible, high-performance serving system for machine learning models designed for production environments.

- \*\*Docker\*\*: A containerization technology that packages applications and their dependencies, ensuring consistency across different environments.

### Choice and Justification

We chose \*\*Flask\*\* as our model serving framework due to the following reasons:

1. \*\*Simplicity\*\*: Flask is easy to set up and requires minimal boilerplate code, making it ideal for our project timeline.

2. \*\*Flexibility\*\*: It provides the flexibility to customize the server as per our needs without enforcing a specific structure.

3. \*\*Community Support\*\*: Flask has a robust community and extensive documentation, which helps in quick issue resolution.

4. \*\*Lightweight\*\*: It's lightweight and sufficient for running the service locally or on a small cloud instance.

---

## 2. Building a Web Service

### Architectural Choices

We implemented a Flask-based web service to host our model. The architecture includes:

1. \*\*Model Training\*\*: A script to train and save the model.

2. \*\*Model Serving\*\*: A Flask application that loads the trained model and exposes an endpoint for predictions.

3. \*\*CI/CD Pipeline\*\*: A manual deployment pipeline using Git and shell scripts to ensure reproducibility and ease of updates.

### Implementation

Here are the key components of our implementation:

- \*\*Training Script\*\*: This script loads the dataset, preprocesses the data, trains the model, and saves the model along with other necessary artifacts.

- \*\*Flask Application\*\*: This application loads the trained model and exposes an HTTP endpoint for making predictions. The application also includes logging to record user inputs and model predictions.

---

## 3. Testing the Deployed Endpoint

### Function to Test the Endpoint

We developed a function to perform testing on the deployed endpoint via HTTP. This function sends a request to the endpoint and logs the predictions.

### Findings

- \*\*Response Time\*\*: The average response time was found to be acceptable for the given load.

- \*\*Accuracy\*\*: The predictions matched our expectations based on the trained model.

---

## 4. Performance Testing

We conducted performance testing by sending multiple requests to the deployed endpoint to evaluate its response time and reliability under load.

### Findings

- \*\*Successful Requests\*\*: The majority of the requests were successful.

- \*\*Failed Requests\*\*: A few requests failed due to network issues.

- \*\*Average Response Time\*\*: The average response time was satisfactory and within acceptable limits for our use case.

### Recommendations

- \*\*Scalability\*\*: To improve the scalability, we can consider deploying the application using containerization technologies like Docker and orchestration tools like Kubernetes.

- \*\*Load Balancing\*\*: Implementing a load balancer can help distribute the traffic evenly across multiple instances of the application, improving reliability and performance.

---

## 5. Basic Monitoring Capability

We implemented basic monitoring capabilities to capture user inputs and model predictions. These logs include the timestamp, the input tokens, and the corresponding model predictions.

### Example Logging Usage

By logging each prediction request, we can monitor the model's performance and track any issues that may arise during its operation. This logging is essential for understanding user interactions and debugging any potential problems.

---

## 6. CI/CD Pipeline

### CI/CD Pipeline Setup

We created a manual CI/CD pipeline to automate the process of pulling the latest changes from the repository, installing dependencies, training the model, and deploying the Flask application.

### Deployment Script

We developed a shell script that performs the following steps:

1. \*\*Pull the Latest Changes\*\*: Updates the local repository with the latest code from the remote repository.

2. \*\*Install Dependencies\*\*: Installs the required Python packages.

3. \*\*Train and Save the Model\*\*: Executes the training script to update the model.

4. \*\*Start the Flask Application\*\*: Runs the Flask application to serve the model.

### Running the Deployment Script

To run the deployment script, we made it executable and executed it from the command line.

### Recommendations

- \*\*Continuous Integration\*\*: Integrate the pipeline with a CI tool like Jenkins or GitHub Actions to automate the process further and trigger the pipeline on code changes.

- \*\*Testing\*\*: Incorporate automated testing into the pipeline to ensure code quality and model performance before deployment.

---

## 7. Demonstration Video

### Screen Recording

We have prepared a 10-minute screen recording demonstrating the entire process from setting up the environment to deploying and testing the model. Each team member presents a part of the demonstration, highlighting their contributions to the project.

### Contents of the Video

1. \*\*Introduction\*\*: Overview of the project and the tasks completed.

2. \*\*Environment Setup\*\*: Steps to set up the environment and install dependencies.

3. \*\*Model Training\*\*: Demonstration of the model training process.

4. \*\*Model Deployment\*\*: Deploying the model using the Flask application.

5. \*\*Endpoint Testing\*\*: Testing the deployed endpoint and analyzing the results.

6. \*\*Performance Testing\*\*: Conducting performance testing and discussing the findings.

7. \*\*CI/CD Pipeline\*\*: Explanation of the CI/CD pipeline and running the deployment script.

The video provides a comprehensive overview of our project, demonstrating our approach and the results achieved.

---

### References

1. \*\*GitHub Repository\*\*: [https://github.com/chandana5301/flask\_nlp](https://github.com/chandana5301/flask\_nlp)

2. \*\*Dataset\*\*: [GoEmotions Dataset](https://huggingface.co/datasets/go\_emotions/viewer/simplified/train)

3. \*\*CI/CD Pipeline Setup\*\*: Abhiney Singh. Configure CI/CD to Deploy Flask App On Docker Container With Jenkins. [Medium Article](https://medium.com/@Abhiney27/configure-ci-cd-to-deploy-flask-app-on-docker-container-with-jenkins-ae350b306c6b)

---

This report provides a detailed overview of our approach to deploying the NLP model, including the research, implementation, testing, performance analysis, and CI/CD pipeline setup.