Solar Panel Deployment Documentation

1. Overview of the Deployment Process

The deployment process involved taking a trained machine learning model and making it available for use in a real-world or production environment. The steps taken during deployment included:

i. **Model Serialization:** After the model was trained, it was saved for later use.
ii. **API Creation:** The model was served via a REST API using the Flask web framework.
iii. **Integration with Google Maps API:** The front-end allowed users to input a location, and a satellite image of that location was fetched using the Google Maps Static API.
iv. **Model Prediction:** The image was passed to the model for classification (detecting solar panels), and the result was returned to the user.
v. **User Feedback Mechanism:** Users could provide feedback on whether the prediction was correct or not, which was logged for potential model retraining.

2. Serialization of the Trained Model

The trained model was serialized using TensorFlow's ".keras" format. This format was chosen because it includes the entire model, including the architecture, weights, and training configuration, allowing the model to be easily loaded and served in different environments. The serialized model was stored as `solar_panel_detection_model.keras`.

Considerations for efficient storage and reuse included:

- Compatibility: The ".keras" format is compatible with TensorFlow Serving and can be loaded across various platforms.
- Storage Efficiency: The format provides a compact way to store the model's components while preserving its performance during inference.

3. Serving the Serialized Model for Predictions

The serialized model was loaded into a Flask application, which served as the back-end for handling user requests and providing model predictions. Flask was selected due to its ease of use, control over web interface styling, lightweight nature, and compatibility with Python and TensorFlow.

When a user submitted a location name, the back-end fetched the corresponding satellite image via the Google Maps Static API. The image was preprocessed and passed to the model for classification, which determined whether solar panels were present. The result was returned to the user in real-time via the Flask API.

4. APIs Used and Integration

The deployment relied on two main APIs:

- Google Maps Static API: This API fetched satellite images of a given location based on the user's input. The API was integrated via a GET request, where the location, zoom level, and image size parameters were specified.
- Flask API: A RESTful API was built using Flask to handle user requests, process satellite images, and return the model's prediction.

Input and Response Formats:

- **Input Format:** The input was a location string (e.g., city or address) submitted by the user.
- **Response Format:** The response included:
    - A prediction indicating whether solar panels were detected.
    - The satellite image fetched from the Google Maps Static API.
    - User feedback submission to indicate whether the prediction was correct or not.

5. Security Considerations

Several security measures were taken into account during the deployment:

- API Key Protection: The Google Maps Static API key was securely stored on the server side and not exposed in the front-end.
- Input Validation: User input (location) was validated to prevent injection attacks or invalid data.
- Cross-Origin Resource Sharing (CORS): CORS policies were enforced to prevent unauthorized access from different origins.
- HTTPS: In production, HTTPS should be enforced to ensure secure communication between the client and server.

6. Monitoring and Logging Mechanisms

Basic logging mechanisms were set up using Flask's built-in logging capabilities to monitor requests, errors, and user feedback:

- Model Predictions: Each model prediction and associated feedback from the user (correct/incorrect) was logged, allowing the team to monitor real-world performance.
- Error Logging: Errors during API requests (e.g., invalid input or API key issues) were logged to help with debugging and error handling.

- User Feedback: User feedback on the accuracy of the model's predictions was logged for potential future improvements.

Future monitoring enhancements could include setting up more sophisticated monitoring tools such as Prometheus for real-time performance tracking, and using a service like ELK Stack for advanced log analysis.