Technical Document: Development of Solar Panel Damage Detection Application

---

1. Introduction

This document outlines the step-by-step process for creating a solar panel damage detection application. The application leverages the YOLOv8 object detection model for damage detection and Streamlit for creating an interactive user interface.

---

2. System Requirements

Hardware:

- CPU: Multi-core processor
- GPU (Optional): NVIDIA GPU with CUDA support
- RAM: Minimum 8GB
- Storage: SSD with ample storage space

Software:

- Operating System: Windows, macOS, Linux
- Python: Version 3.7 or higher
- Libraries: PyTorch, OpenCV, Streamlit, Roboflow

---

3. Data Collection

- Gather diverse images of solar panels showcasing different types of damage.
- Annotate images using Roboflow to create bounding boxes around damaged areas.
- Use data version control tools like Git or DVC for managing datasets.

---

4. Data Preparation

- Select a subset of high-quality images for training, ensuring representation of all damage types.
- Clean the data by removing noise and handling missing values.
- Enhance dataset through feature engineering and data augmentation techniques.
- Standardize data format and normalize pixel values.

---

5. Model Training

- Choose YOLOv8 for its speed and accuracy in object detection tasks.
- Train the model on annotated dataset, adjusting hyperparameters as necessary.
- Validate model performance using metrics like mAP50 and mAP50-95.
- Save trained model for later use.

---

6. Streamlit Application Development

- Install Streamlit and necessary libraries.
- Create a user-friendly interface using Streamlit widgets.
- Load the trained YOLOv8 model for inference.
- Process uploaded images and display detection results in the application.

---

7. Deployment

- Test the application locally using Streamlit.
- Deploy the application to a cloud platform like Heroku, AWS, or Google Cloud.
- Ensure proper environment setup and dependencies in the deployment process.

---

8. Monitoring and Maintenance

- Monitor model performance and user feedback for continuous improvement.
- Update the model with new data and algorithms to maintain accuracy.
- Regularly maintain hardware and software infrastructure for optimal performance.