✅ PHASE 1: Project Setup & Planning

1. **Define Objectives**
    - Detect road lanes using deep learning.
    - Identify unintentional lane departures.
    - Alert the driver (visually/audio) in real-time.
2. **Collect Requirements**
    - Real-time processing (embedded or laptop).
    - Target hardware: Raspberry Pi, Jetson Nano, or PC.
    - Dashcam or front-facing camera input.
3. **Choose Tools & Frameworks**
    - **Language**: Python
    - **Frameworks**: OpenCV, TensorFlow / PyTorch
    - **IDE**: VS Code / Jupyter Notebook
    - **Version Control**: GitHub

🗀 PHASE 2: Dataset Preparation

4. **Choose or Collect Dataset**
    - Use public datasets like:
        - TuSimple Lane Detection Dataset
        - CULane Dataset
        - BDD100K
    - Optionally record your own using dashcam or smartphone.
5. **Annotate Data (if custom)**
    - Label lane lines using tools like CVAT or LabelMe.
6. **Preprocess Dataset**
    - Resize images.
    - Normalize pixels.
    - Apply augmentations (brightness, flipping, shadows).
    - Convert labels to segmentation masks or coordinates.

□ PHASE 3: Model Design & Training

7. **Select Deep Learning Model**

- **Segmentation-Based**: U-Net, ENet, SCNN
- **Detection-Based**: YOLO for lane line anchors
- Pre-trained models: LaneNet, UltraFast Lane Detection

8. **Implement Model**
    - Use PyTorch or TensorFlow to define the architecture.
    - Use pre-trained weights if needed.
9. **Train Model**
    - Set loss function (e.g., CrossEntropy for segmentation).
    - Set optimizer (Adam/SGD).
    - Train and validate using split dataset.
    - Monitor performance with accuracy, IoU, F1-score.
10. **Save and Export Model**

- Save weights.
- Export to ONNX or TensorFlow Lite for real-time deployment.

☎ PHASE 4: Lane Detection Pipeline

11. **Real-time Lane Detection**

- Capture video frames from camera.
- Preprocess frames and feed to model.
- Post-process predictions (polylines, overlays).

12. **Apply Perspective Transformation (Bird's Eye View)**

- Warp perspective to better understand lane curves.

13. **Smooth Lane Lines Across Frames**

- Use exponential moving average or Kalman Filter.

⚠ PHASE 5: Lane Departure Warning System (LDWS)

14. **Define Lane Boundaries**

- Detect vehicle position w.r.t lane center.

15. **Trigger Departure Warning**

- If the vehicle drifts too close or crosses the boundary without indicator:
    - Show visual cue.
    - Play audio alert (using pygame or playsound).

□ PHASE 6: Testing & Evaluation

16. **Test on Different Scenarios**

- Straight & curved roads
- Day & night
- Rain & fog (try synthetic rain tools)

17. **Evaluate Accuracy**

- False positives (wrong departure warnings)
- False negatives (missed warnings)

18. **Optimize for Speed**

- Resize input frames.
- Use lightweight models (like MobileNet).
- Consider hardware acceleration (e.g., TensorRT, OpenVINO)

🚀 **PHASE 7: Deployment**

19. **Deploy to Edge Device or PC**

- Raspberry Pi / Jetson Nano / Laptop
- Ensure real-time performance (20–30 FPS)

20. **User Interface (Optional)**

- Display video with detected lanes and warnings
- Add controls for testing/debugging

📄 **PHASE 8: Documentation & Report**

21. **Document Code & Architecture**

- README file
- Installation and usage instructions

22. **Write Final Project Report**

- Problem Statement
- Methodology
- Experiments & Results
- Conclusion & Future Work

23. **Prepare Presentation / Demo**

- PPT with key visuals
- Live demo or recorded video of the system