Internship Report Entitled

WELD DEFECT DETECTION AND SEGEMENTATION IN RADIOGRAPHIC IMAGES USING DEEP LEARNING TECHNIQUES

Submitted by

ARAVIND VIJAYAN

Under the guidance of

Smt. SMITHA K.K

TECH.OFR-C, CISD/CISDG

JUNE, 2025

Certificate Of Completion

This is to certify that **Aravind Vijayan,** Bachelor of Technology in Computer Science and Engineering of Cochin University of Science and Technology has completed the work presented in the report titled "***Weld Defect Detection and Segmentation in Radiographic images using Deep Learning Techniques*** " at **Liquid Propulsion Systems Centre**, Valiamala, Trivandrum during the period from May 2025 – June 2025. This is a bonafide record of his original work carried out under my supervision and guidance.

Smt. Smitha K.K
Tech.Ofr-C, CISD/CISDG
LPSC, Thiruvananthapuram

ACKNOWLEDGEMENT

With a deep sense of gratitude, I wish to express my sincere thanks to my guide **Smt. Smitha K.K** (Tech.Ofr-C, CISD/CISDG), for her unwavering support, scholarly guidance, and relentless cooperation throughout this project. His mentorship has been invaluable in helping me to complete the internship smoothly.

I am also deeply grateful to **Mr. Dushyant M.P** (GD, CISDG) and **Mr. Ajith S** (Head, PPEG) for allowing me to carry out my project in their esteemed department at the Liquid Propulsion Systems Centre, Valiamala.

Additionally, I would like to thank all the Sci/Engineers of Computer Infrastructure and Software Development Group (CISDG) who assisted me during my internship. It is with immense pleasure and gratitude that I acknowledge their support.

I extend my heartfelt thanks to the department staff members and friends who contributed to the successful completion of this internship.

I perceive this opportunity as a significant milestone in my professional growth. I will strive to use the skills and knowledge gained in the best possible way and will continue to work on their improvement to attain the desired level of expertise.

Sincerely,
Aravind Vijayan

ABSTRACT

Weld quality inspection is a critical aspect of manufacturing across various industries, including aerospace, where structural integrity directly impacts safety and performance. Radiographic testing (RT) using X-ray imaging is a widely employed Non-Destructive Testing (NDT) method to detect internal defects in welded joints without damaging the components. At the Liquid Propulsion Systems Centre (LPSC), ISRO, precise inspection of welded aluminium tanks used in the PSLV rocket is essential to ensure mission success.

Traditionally, the analysis of these radiographic X-ray images has been performed manually by skilled inspectors, a process that is time-consuming and prone to subjectivity. To address these challenges, this project focuses on developing an automated deep learning-based system for detecting and segmenting weld defects from X-ray images. The process involves validating the quality of input images, isolating the weld area, and extracting embedded textual annotations for traceability. The core of the project lies in employing the YOLOv8 segmentation model, which enables accurate pixel-level identification of weld defects such as isolated, aligned, and clustered pores.

This deep learning-driven approach not only improves detection accuracy but also significantly reduces inspection time and human error. The solution is designed to be scalable and adaptable, with potential applications extending beyond aerospace to various industrial sectors requiring rigorous quality assurance.

By automating the segmentation of weld defects, this project contributes to modernizing traditional NDT workflows, enhancing efficiency, repeatability, and reliability in weld quality inspection processes.

LIST OF ABBREVIATIONS

ABBREVIATIONS	FULL FORMS
ISRO	Indian Space Research Organisation
LPSC	Liquid Propulsion Systems Centre
PSLV	Polar Satellite Launch Vehicle
NDT	Non-Destructive Testing
RT	Radiographic Testing
DL	Deep Learning
YOLO	You Only Look Once
CNN	Convolutional Neural Network
R-CNN	Region-based Convolutional Neural Network
GPU	Graphics Processing Unit
CPU	Central Processing Unit
RAM	Random Access Memory
ROI	Region of Interest
mAP	mean Average Precision

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Weld quality assurance is a fundamental requirement across various industries, especially in aerospace, where structural integrity is critical for mission success and safety. The inspection of welds using radiographic X-ray images is a widely adopted Non-Destructive Testing (NDT) technique that enables the detection of internal flaws without damaging the component. However, the traditional manual analysis of these images by trained inspectors is time-consuming, subjective, and often inconsistent, which can lead to missed defects or inaccurate evaluations.

This project aims to develop an automated deep learning-based system for the detection and segmentation of weld defects in radiographic images, with a particular focus on welded aluminium tanks used in the PSLV rocket at the Liquid Propulsion Systems Centre (LPSC), ISRO. The system improves upon conventional methods by incorporating advanced image processing techniques to validate X-ray quality, isolate weld regions, and extract embedded textual annotations. Central to this approach is the use of a state-of-the-art segmentation model that precisely identifies various types of weld defects, including isolated, aligned, and clustered pores.

By integrating deep learning into the weld inspection workflow, this project seeks to enhance the accuracy, speed, and reliability of defect detection. The automation not only reduces human effort and error but also offers scalability, making the solution applicable to a wide range of industrial quality assurance processes beyond aerospace. Through this work, the project contributes to modernizing traditional NDT practices, enabling more efficient and objective weld quality evaluations.

2. ABOUT

2.1 Indian Space Research Organisation (ISRO)

India's journey into space began with the establishment of the Indian National Committee for Space Research (INCOSPAR) by the Government of India in 1962. The visionary Dr. Vikram Sarabhai, regarded as the father of the Indian space program, played a pivotal role in this initiative by founding the Thumba Equatorial Rocket Launching Station (TERLS) in Thiruvananthapuram, dedicated to upper atmospheric and microgravity research.

In 1969, INCOSPAR was superseded by the Indian Space Research Organisation (ISRO), which has since remained committed to its mission of harnessing space technology for the benefit of the nation and its people. Over the years, ISRO has evolved into one of the world's leading space agencies.

ISRO operates one of the largest fleets of communication satellites under the INSAT series and remote sensing satellites under the IRS series, addressing the growing demands for efficient communication and earth observation. It develops and delivers a wide range of application-specific satellite products and tools supporting national broadcasting, telecommunications, weather forecasting, disaster management, geographic information systems (GIS), cartography, navigation, telemedicine, and distance education.

In terms of launch vehicle development, ISRO has successfully designed the Polar Satellite Launch Vehicle (PSLV) for deploying IRS satellites and the Geosynchronous Satellite Launch Vehicle (GSLV) for GSAT class satellites. The organisation also actively pursues ambitious space science missions such as Chandrayaan-2 and the Mars Orbiter Mission (MOM).

ISRO supports academic and research institutions across India by fostering projects aligned with the national space program. To commercialize its space products and services, the government-owned Antrix Corporation was established in 1992.

Looking ahead, ISRO is focused on future technologies including the development of heavy lift launch vehicles, human spaceflight programs, reusable launch vehicles, semi-cryogenic engines, and single and two-stage-to-orbit (SSTO and TSTO) vehicles. Additionally, it is advancing the use of composite materials and other innovative technologies to meet the evolving needs and ambitions of the country's space endeavours.

2.2 Liquid Propulsion Systems Centre (LPSC)

The Liquid Propulsion Systems Centre (LPSC) is the premier centre responsible for the development and realization of advanced earth-to-orbit propulsion stages for launch vehicles, as well as in-space propulsion systems for spacecraft. LPSC's activities and facilities are distributed across two campuses: the headquarters and design offices located at Valiamala, Thiruvananthapuram, and the Spacecraft Propulsion Systems division situated in Bengaluru, Karnataka.

LPSC is entrusted with the design, development, and system engineering of high-performance space propulsion systems utilizing both earth-storable and cryogenic propellants for ISRO's launch vehicles and satellites. The centre also undertakes the development of critical components such as fluid control valves, transducers, propellant management devices, and other essential elements of liquid propulsion systems.

The Valiamala campus serves as the Centre Headquarters and is responsible for research and development, system design and engineering, and project management. It houses core entities including the Fluid Control Components Entity, the Materials & Mechanical Engineering Entity, and the Earth Storable & Cryogenic Propulsion Entities, all of which play vital roles in the centre's mission.

LPSC has made significant contributions to the development of liquid propulsion stages for ISRO's Polar Satellite Launch Vehicle (PSLV) and Geosynchronous Satellite Launch Vehicle (GSLV). Notably, the centre developed the cryogenic upper stage for the GSLV MkIII, greatly enhancing ISRO's payload capacity to geostationary orbits. This breakthrough cryogenic technology is critical for launching heavier payloads and has positioned India competitively in the global satellite launch market. Additionally, LPSC is responsible for the propulsion systems of interplanetary missions, including the Mars Orbiter Mission and the forthcoming Chandrayaan-3.

Beyond launch vehicle propulsion, LPSC designs and develops propulsion systems for spacecraft, encompassing attitude control, orbital correction, and primary propulsion. The centre has engineered propulsion solutions for communication satellites, remote sensing satellites, and scientific missions. A notable achievement includes the liquid apogee motor, which facilitates orbit-raising maneuvers for geostationary satellites—an essential capability for precise satellite positioning and operation in space.

2.3 Deep Learning Image-Based Models and Applications

Deep learning has emerged as a transformative approach in the field of image analysis, enabling automated and highly accurate interpretation of complex visual data. In the context of non-destructive testing (NDT), deep learning image-based models combine the power of advanced neural network architectures with large datasets to detect, classify, and segment defects in radiographic images with unprecedented precision.

Key Components of Deep Learning Image-Based Models:

1. **Feature Extraction:** These models automatically learn and extract hierarchical features from raw image data, capturing spatial patterns, textures, and subtle variations that are critical for distinguishing weld defects.
2. **Defect Detection:** By leveraging convolutional neural networks (CNNs) and region-based algorithms, deep learning models localize defects within radiographic images, enabling rapid and reliable identification of anomalies such as isolated pores, aligned pores, and clustered pores.
3. **Segmentation:** Advanced segmentation models provide pixel-level delineation of defect boundaries, offering detailed insights into defect morphology, size, and distribution. This fine-grained analysis is essential for accurate assessment and quality control.
4. **Integration with Traditional Techniques:** When combined with classical image processing and optical character recognition (OCR), these models create comprehensive inspection work-flows that include image validation, weld localization, defect segmentation, and annotation extraction.

Applications in Industrial Quality Assurance:

- **Automated Weld Inspection:** Deep learning image-based models streamline the evaluation of welded structures by providing consistent, objective, and rapid defect detection and segmentation, reducing reliance on manual inspections.
- **Scalability Across Sectors**: Although initially developed for aerospace applications such as inspection of aluminium tanks in launch vehicles, these models are broadly applicable across industries requiring stringent weld quality assurance, including automotive, construction, and energy sectors.

- **Enhanced Decision Support:** By providing precise spatial and morphological defect information, these models enable engineers and quality control personnel to make better-informed decisions regarding material integrity and safety.

2.4 Project Overview and Objectives

This project focuses on developing a deep learning-based system for the automatic detection and segmentation of weld defects in radiographic images of welded aluminium tanks used in launch vehicles. The primary goal is to create an accurate and efficient solution that supports quality assurance by identifying critical weld defects such as isolated, aligned, and clustered pores, enhancing the structural integrity and safety of these tanks.

During the project, multiple models and techniques were explored and evaluated to address different sub-tasks in the inspection pipeline. Traditional Convolutional Neural Networks (CNN) and region-based architectures like Faster R-CNN were tested for validating the authenticity of X-ray images. Image processing techniques using OpenCV were employed to accurately detect and isolate the weld area within the radiographs. For weld defect segmentation, advanced models like YOLOv8 were utilized to achieve precise pixel-level defect delineation.

Key Objectives of the Project:

1. **X-ray Image Validation:**
   Evaluate and compare multiple deep learning models, including CNN and Faster R-CNN, to develop a reliable method for confirming the validity of radiographic images before further analysis.
2. **Weld Area Detection:**
   Apply OpenCV-based image processing techniques to localize the weld region in the X-ray images, enabling focused and accurate defect detection.
3. **Annotation Extraction:**
   Use Optical Character Recognition (OCR) tools to extract relevant textual information such as weld IDs and inspection notes from the images, facilitating detailed documentation.
4. **Weld Defect Segmentation:**
   Implement state-of-the-art segmentation models, primarily YOLOv8, to detect and precisely segment weld defects including isolated, aligned, and clustered pores, providing detailed defect mapping.

5. **Performance Evaluation and Optimization:**

   Continuously assess the performance of all models and optimize the integrated system to ensure accuracy, speed, and robustness suitable for industrial deployment.

6. **Scalability and Generalization:**

   Develop a scalable system adaptable to different welded aluminium tanks and potentially other welded components across industries, ensuring broad applicability.

3. LITERATURE SURVEY

3.1 Overview of Weld Defect Detection

Weld inspection is critical for ensuring structural integrity in industries such as aerospace, shipbuilding, and pressure vessel manufacturing. Traditional methods for inspecting welds, such as manual radiographic interpretation, are time-consuming and subject to human error. In recent years, deep learning (DL) techniques have gained popularity due to their high accuracy and automation potential in detecting weld defects from radiographic images.

3.2 Challenges in Radiographic Weld Defect Detection

The detection of weld defects through X-ray images presents several challenges:

- **Low Contrast and Noise**: Weld defects often appear faint and can be obscured by image noise, making them difficult to detect with traditional algorithms.
- **Class Imbalance**: Certain defects, like cracks or inclusions, are rarer than others, complicating the training of robust classification models.
- **Tiny Defect Sizes**: Small-scale defects such as clustered pores may be lost in down sampled feature maps used by some detection models.
- **Data Scarcity**: Annotated weld radiograph datasets are limited due to proprietary concerns and annotation complexity.

3.3 Key Research Summaries

Recent advances in deep learning have significantly improved the accuracy and speed of weld defect detection from radiographic images. The following summaries highlight five relevant techniques and model architectures that demonstrate notable performance in this domain.

1. LF-YOLO (Light and Fast YOLO):

Liu et al. proposed LF-YOLO, a modified YOLOv5 model enhanced with Residual Multi-scale Feature Fusion (RMF) and Enhanced Feature Extraction (EFE) modules. This lightweight model achieved a high mAP_{50} of 92.9% and 61.5 FPS, making it suitable for real-time weld defect detection in industrial pipelines where both accuracy and speed are critical.

2. EL-YOLOv8 (Efficient Lightweight YOLOv8):

Cheng et al. developed EL-YOLOv8 by optimizing YOLOv8 using Efficient Multi-branch Attention (EMA) and FasterNetBlock modules. The model achieved an impressive $mAP_{0.5}$ of 91.5% with a real-time processing speed of 205 FPS. It demonstrates excellent balance between detection accuracy and computational efficiency for X-ray inspection of pipeline welds.

3. YOLOv5 vs. Faster R-CNN on Steel Welds:

Yang et al. compared YOLOv5 and Faster R-CNN for detecting weld defects in steel pipes. While both models performed well in terms of accuracy, YOLOv5 significantly outperformed Faster R-CNN in inference speed, reinforcing its suitability for real-time detection tasks in industrial environments.

4. WRT-SAM (Weld Radiography Tuned SAM):

Zhou et al. adapted Meta's Segment Anything Model (SAM) for weld X-ray segmentation, incorporating multi-scale tuning and frequency-based prompts. The model achieved high segmentation performance with a precision of 84.0% and AUC of 0.9746, showcasing the potential of foundation models in weld defect segmentation.

5. Improved YOLOv8-Seg for Weld Seam Tracking:

An enhanced YOLOv8s-seg model was proposed by integrating MobileNetV3, C2fGhost, and EMA modules to reduce model size while retaining accuracy. With a compact size of 4.85 MB, this version is ideal for real-time weld seam tracking on edge devices in automated welding systems.

3.4 Technological Advancements

Deep learning methods have significantly advanced the capabilities of weld defect detection:

- **Convolutional Neural Networks (CNNs)**: CNN-based architectures like ResNet and VGG have been used for classifying weld defects such as porosity, cracks, and lack of fusion.
- **Object Detection Models**: Models such as Faster R-CNN, YOLOv5, and YOLOv8 have been applied to detect and localize weld defects in real-time with high precision.
- **Segmentation Models**: YOLOv8 and Mask R-CNN architectures provide pixel-level segmentation of defects, aiding in precise localization and measurement.
- **Preprocessing Techniques**: Methods such as Adaptive Thresholding and curve transforms have been employed to enhance defect visibility prior to model input.

3.6 Public Datasets and Benchmarks

- **GDXray**: One of the most widely used public datasets for weld and casting defect detection.
- **Industrial Datasets**: Proprietary datasets from research institutions and companies provide higher variability in defect types and imaging conditions, though often not publicly available.

3.7 Case Studies in Industrial Applications

- **Aerospace Industry**: CNN-based systems integrated into automated inspection pipelines to identify lack of penetration and porosity in aluminium fuel tanks.
- **Manufacturing Quality Control**: YOLO and UNet models used for in-line detection of cracks in high-speed production environments, reducing inspection time and human workload.

3.8 Future Directions

Advancements in weld defect detection are expected to focus on:

- **Multimodal Inspection**: Fusion of X-ray, ultrasonic, and thermographic data to enhance defect detection reliability.
- **Domain Adaptation**: Techniques to transfer models trained on one domain (e.g., steel welds) to another (e.g., aluminium welds) without retraining from scratch.
- **Real-Time Edge Deployment**: Developing lightweight models suitable for deployment on embedded systems at manufacturing sites.
- **Explainability and Trust**: Integrating visualization tools to explain model decisions and foster trust among inspectors and engineers.

4. SYSTEM ARCHITECTURE

6.1 YOLOv8 Segmentation Architecture

6.1.1 Overview of YOLOv8 for Segmentation

YOLOv8 (You Only Look Once version 8) is the latest in the YOLO family of real-time object detection models developed by Ultralytics. Unlike earlier versions, YOLOv8 offers a modular and unified architecture that supports not only object detection but also instance segmentation, classification, and pose estimation. The segmentation variant is designed to generate pixel-level masks for each detected object, making it suitable for tasks like weld defect detection in radiographic images, where fine-grained localization of defects is essential.

YOLOv8 is written in Python using PyTorch and is fully open-source, offering ease of integration and customization. It achieves state-of-the-art performance across accuracy, speed, and computational efficiency benchmarks.

6.1.2 Key Components of YOLOv8 Segmentation Architecture

- **Backbone**

  The backbone extracts critical features from weld X-ray images. YOLOv8 uses an improved CSPDarknet with C2f modules to efficiently capture detailed multi-scale features, essential for accurately detecting small and complex defects.
- **Neck**

  The neck fuses feature from different backbone layers, combining local details with global context. It employs PANet and FPN-inspired structures to enhance feature representation for better defect localization.
- **Detection Head**

  This component predicts bounding boxes and classifies detected defects. Using advanced loss functions, it optimizes accuracy while minimizing false positives to ensure reliable identification.
- **Segmentation Head**

  The segmentation head produces pixel-level masks for each defect, enabling precise instance segmentation by combining prototype masks with object-specific coefficients.

- **Post-Processing**

  Post-processing refines the output by applying Non-Maximum Suppression to eliminate redundant overlapping detections, delivering clean and accurate defect results.

6.1.3 Architecture Figure

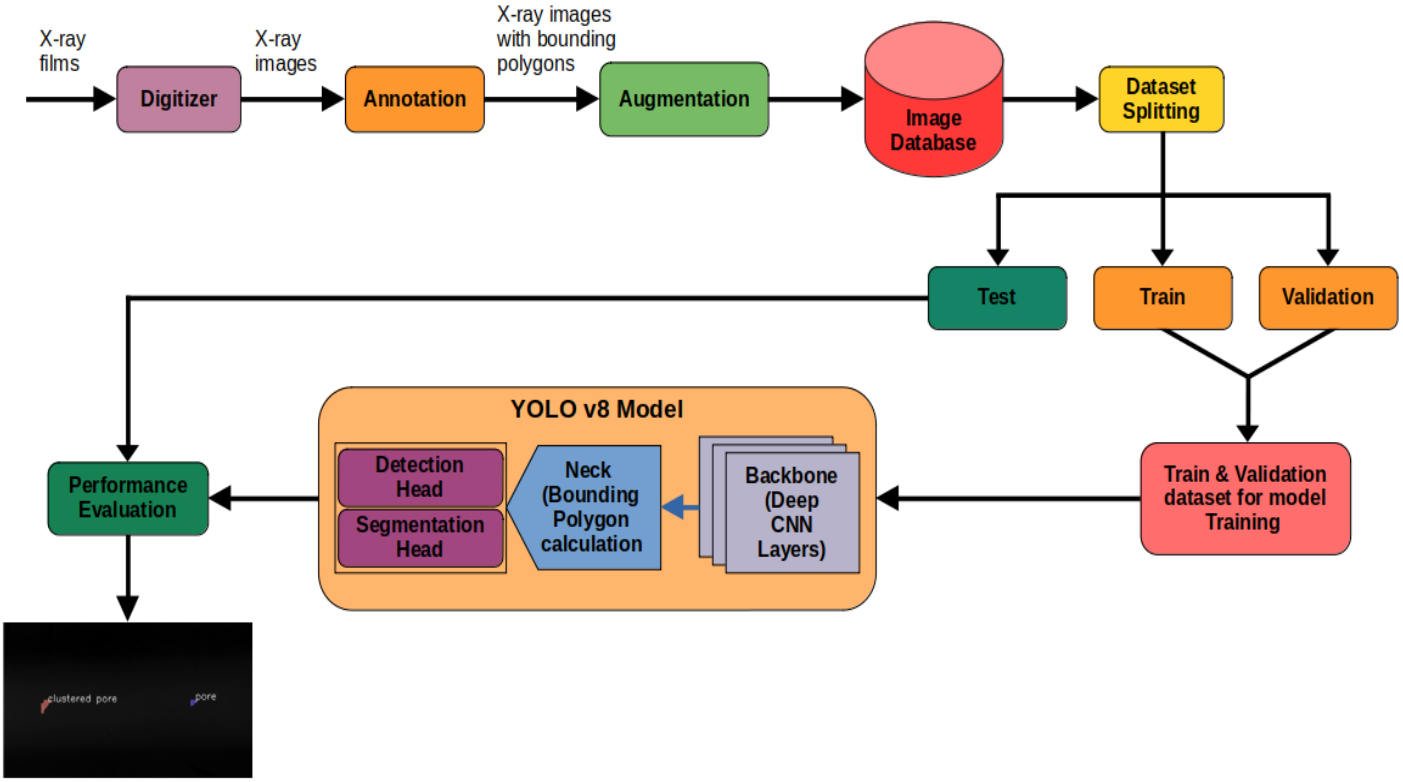

Fig 1. Model Architecture

6.1.4 Architecture Explanation

1. X-ray Film Digitization

The process begins with the digitization of physical X-ray films, which are used to inspect welded components such as aluminium tanks in PSLV rockets at LPSC–ISRO. These films are converted into high-resolution digital images using an X-ray digitizer. This step enables downstream digital processing and ensures the preservation and accessibility of radiographic data for automation.

2. Annotation

Once digitized, the X-ray images undergo manual annotation using the LabelMe tool. This tool allows the user to draw polygonal boundaries around visible weld defects such as pores and cracks. These annotated regions serve as ground truth labels for training the deep learning model in segmentation tasks. Each defect is marked with its respective class to support multi-class segmentation.

3. Data Augmentation

To enhance model robustness and improve generalization, the annotated dataset is augmented using techniques such as flipping, rotation, scaling, and brightness adjustments. Augmentation artificially increases dataset size and diversity, helping the model learn more varied patterns and reduce overfitting during training.

4. Image Database Formation

All original and augmented images, along with their corresponding annotations, are compiled into a centralized image database. This database forms the core input repository for the subsequent training, validation, and testing phases. Metadata such as image paths, class labels, and polygon coordinates are maintained for efficient access and traceability.

5. Dataset Splitting

To enable proper training and evaluation, the dataset is divided into three subsets: training, validation, and testing. The training set is used to learn model parameters, while the validation set helps tune hyperparameters and monitor performance during training. The test set is reserved for unbiased evaluation of the model's real-world performance.

6. YOLOv8 Segmentation Model

The core of the system is the YOLOv8 segmentation model. It comprises three main components: a backbone, neck, and dual heads. The **backbone** consists of deep CNN layers that extract hierarchical features from the input image. The **neck** performs feature fusion and refines spatial localization using bounding polygon calculations. The model has two output heads: the **detection head**, which predicts bounding boxes for defect regions, and the **segmentation head**, which generates pixel-wise masks to precisely outline each defect. This architecture enables real-time, multi-class segmentation of weld defects.

7. Model Training and Evaluation

The training and validation subsets are used to optimize the model over multiple epochs. Performance is evaluated using metrics such as Precision, Recall, Intersection over Union (IoU), and mean Average Precision (mAP). These metrics help quantify how accurately the model detects and segments defects. The evaluation ensures the model's readiness for deployment on real-world data.

8. Output and Visualization

After evaluation, the YOLOv8 model is applied to test images to visualize the segmentation results. Defects are overlaid with colored polygon masks and class labels (e.g., “clustered pore” or “pore”) for easy interpretation. This visual output supports quality inspectors in verifying weld integrity rapidly and objectively.

5. MODEL SELECTION AND CONFIGURATION

The project involved solving multiple sub-tasks as part of a weld defect detection pipeline using radiographic (X-ray) images. Each task required exploring a range of methods and models to identify the most suitable one based on accuracy, efficiency, and hardware constraints. Due to the unavailability of a GPU, all models were trained and tested on a CPU. As a result, lightweight models with lower computational requirements were prioritized wherever possible.

5.1 Task 1: Valid X-ray Image Detection

The goal of this task was to verify whether a given X-ray image is valid, based on the presence of vertical calibration strips that typically appear along the image margins.

- **Approach 1 – OpenCV (Hough Line Transform):** Initial experiments using OpenCV to detect vertical lines with edge detection and Hough transforms yielded unreliable results. The method was sensitive to noise and failed in low-contrast or non-uniform backgrounds.
- **Approach 2 – CNN-based Classifier:** A custom CNN was trained to classify images as valid or invalid based on the presence of strips. However, due to the subtle nature of the strip features and limited training data, the model struggled with generalization and gave inconsistent results.
- **Approach 3 – YOLOv5:** The YOLOv5 model was evaluated for object-level detection of the strip region. While it could detect prominent lines in some cases, it failed to consistently identify subtle or faint vertical strips across varying images.
- **Final Choice – Faster R-CNN:** Among the tested models, Faster R-CNN provided the best performance in identifying strip regions accurately. It demonstrated better localization than YOLOv5, especially on CPU, even though the processing time was slightly higher. This model was ultimately selected due to its relatively higher precision in this binary classification/detection task.

5.2 Task 2: Weld Area Detection

To assist in further processing (e.g., OCR), the strip region needed to be pre-processed for better contrast and clarity.

- **CNN-based Approaches:** Attempts to use CNN models for weld area detection (e.g., learning-based image enhancement or segmentation) were computationally expensive and provided no significant improvement over traditional methods.
- **OpenCV Adaptive Thresholding:** Adaptive thresholding is a powerful image preprocessing technique used to enhance the contrast of regions of interest, especially in images with varying lighting conditions. In this project, adaptive thresholding methods provided by OpenCV—specifically the *Mean* and *Gaussian* adaptive methods—were applied to X-ray weld images to isolate the weld region. Unlike global thresholding which applies a single threshold value across the entire image, adaptive thresholding calculates the threshold for each pixel based on the pixel intensities in its local neighbourhood. This makes it especially effective in scenarios where illumination is non-uniform or the weld boundaries are subtle.
  In the *Mean* method, the threshold for a pixel is set as the average of neighbouring pixel values minus a constant. The *Gaussian* method improves upon this by weighting the neighbouring pixels using a Gaussian window, leading to smoother transitions and better segmentation of the weld region. These methods significantly enhanced the visibility of weld areas by suppressing background noise and highlighting the structural patterns.

5.3 Task 3: Text Extraction (OCR)

Once the weld area regions were detected, the next task was to recognize printed or embossed numbers for validation purposes.

- **Custom OCR with CNN/RNN:** Custom OCR models were considered, using CNN + LSTM architectures for sequence recognition. However, training such models from scratch required a large amount of labelled sequence data, which was unavailable for this task.
- **Final Choice – Tesseract OCR:** The open-source Tesseract OCR engine was integrated into the pipeline. It was pre-trained on general numeric text and performed well on the enhanced weld region images. Its easy setup, reliable accuracy, and ability to run efficiently on CPU made it the optimal choice.

5.4 Task 4: Weld Defect Segmentation

This was the core task of the system—detecting and segmenting welding defects in radiographic images.

- **OpenCV Contour/Edge Methods:** Traditional computer vision methods using edge detection, thresholding, and contour detection failed to consistently identify defect regions due to variation in shape, size, and contrast of the defects.
- **CNN-based Classifiers:** A standard CNN classifier was tested for defect classification but lacked the spatial resolution and generalization ability needed to segment and localize defects effectively.
- **YOLOv5 & YOLOv7:** These models were tested for object detection, but their performance on CPU was not optimal. YOLOv5 showed moderate results, while YOLOv7 required more resources than available.
- **Final Choice – YOLOv8n (Nano):** YOLOv8n, the lightest version of the YOLOv8 series, demonstrated excellent performance in detecting and segmenting weld defects, even in CPU-only environments. It provided a good balance of inference speed and detection accuracy. YOLOv8n's native support for segmentation tasks (in addition to bounding boxes) made it ideal for this requirement.

5.5 Hardware Constraints and Model Configuration

All training and inference tasks were performed on CPU due to the lack of access to GPU infrastructure. This limited the choice of models and increased the need for efficient computation. To address this:

- Lightweight models like **YOLOv8n** and **Tesseract** were chosen for their CPU compatibility.
- **Transfer learning** was applied where possible to reduce training time and improve performance on limited data.
- Hyperparameters such as **batch size**, **input size**, **learning rate**, and **number of epochs** were tuned based on empirical observations and resource constraints.
- Data augmentation techniques like rotation, scaling, and flipping were used to improve model robustness.

6. SYSTEM DESIGN

6.1 Workflow Diagram

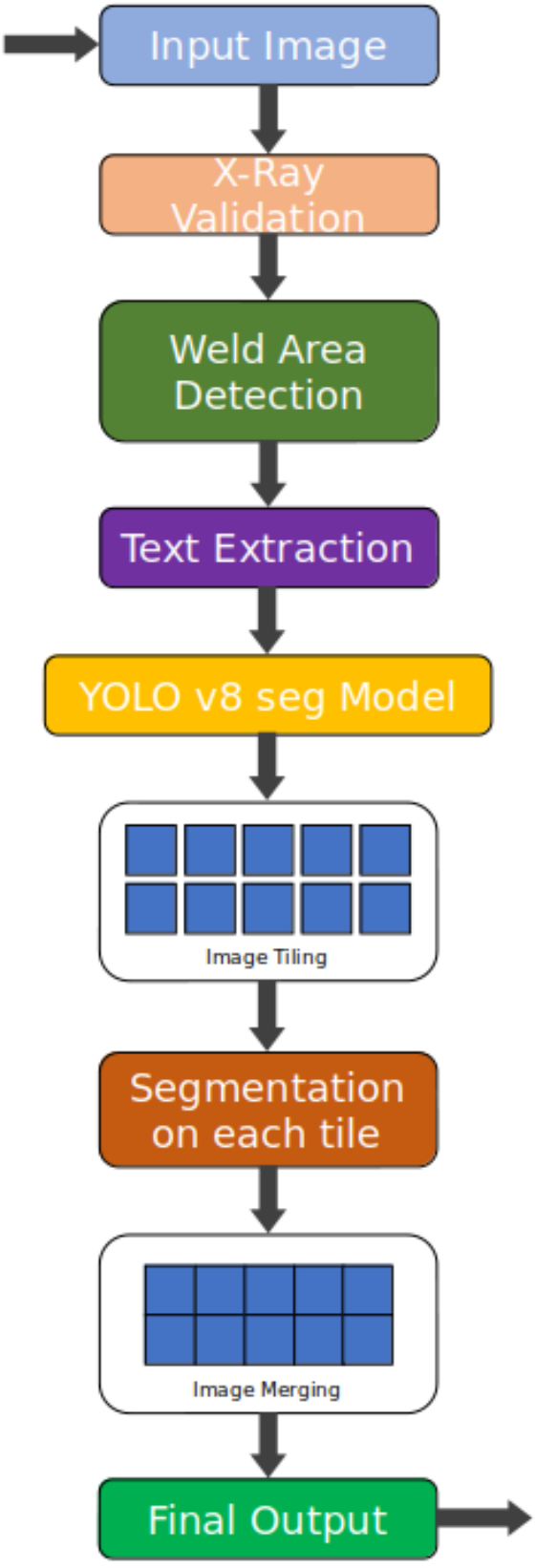

Fig 2. System Workflow Design

6.2 Workflow

1. Input Image

The process begins with an input X-ray image of the welded component. This image can be obtained via digitization of X-ray films or direct digital radiography.

2. X-ray Validation

To ensure that the input is indeed a valid weld X-ray, a validation step is performed. This is typically done using classical computer vision techniques (e.g., vertical line count verification) to filter out irrelevant images.

3. Weld Area Detection

The validated X-ray image undergoes weld area detection using adaptive thresholding and contour filtering. This helps isolate the region of interest (ROI) by identifying the high-contrast vertical weld strip within the image.

4. Text Extraction

Embedded text (e.g., part number, weld ID) present in the X-ray image is extracted using Tesseract OCR. This metadata is useful for downstream labeling, traceability, and documentation.

5. YOLOv8-seg Model

The cropped weld region is passed through a YOLOv8 segmentation model trained to detect and segment various defect types such as cracks, porosity, and lack of fusion. The model architecture combines a backbone (deep CNN), neck (feature aggregation), and dual heads for detection and segmentation.

6. Image Tiling

To handle high-resolution images effectively and ensure finer segmentation accuracy, the weld ROI is divided into smaller overlapping or non-overlapping tiles (patches). This step ensures the model can process each region without memory or resolution limitations.

7. Segmentation on Each Tile

Each tile is individually passed through the YOLOv8 segmentation model. The model detects and classifies defects at the tile level.

8. Image Merging

Post-segmentation, the processed tiles are merged back to reconstruct the full-resolution segmentation map of the original weld area. Care is taken to ensure seamless merging and coordinate alignment.

9. Final Output

The final output is a fully segmented weld X-ray image highlighting the locations and types of detected defects. It can be visually inspected or programmatically used for quality control and reporting.

7. IMPLEMENTATION

The implementation of the weld defect detection system was carried out as a modular pipeline, with each task developed and tested independently before integrating them into a complete workflow. Python was the primary language used, and all models were trained and executed on a CPU-only environment due to hardware limitations.

7.1 Environment and Tools

- **Programming Language:** Python 3.10
- **Libraries & Frameworks:** OpenCV, PyTorch, Tesseract OCR, Ultralytics YOLOv8
- **IDE:** VSCode
- **Hardware:** Intel i5 CPU, 16GB RAM (no GPU)
- **Training Platform:** Local machine with CPU, batch size and model choice optimized for resource constraints.

7.2 Module-wise Implementation

- **Weld Area Detection (Faster R-CNN):**
  This module detects the weld area in the input X-ray images. Faster R-CNN was selected after evaluating YOLOv5 and CNN approaches which failed to detect the vertical weld calibration patterns. It was implemented using the torchvision library and fine-tuned on a small set of annotated weld images. Post-processing involved verifying the region dimensions and presence of expected structures.
- **Image Preprocessing:**
  For valid images, OpenCV was used to convert the images to grayscale, resize them, and apply adaptive thresholding. This significantly enhanced the contrast of low-visibility regions. Noise was reduced using a Gaussian blur filter, and CLAHE (Contrast Limited Adaptive Histogram Equalization) was applied in some cases to improve defect visibility.
- **OCR (Tesseract):**
  Tesseract OCR was used to extract calibration numbers or serial codes. It worked effectively when paired with adaptive thresholding and binarization. OCR was restricted to specific ROIs (regions of interest) determined from weld strip detection, avoiding false extractions.

- **Defect Detection (YOLOv8n):**
  For defect detection, YOLOv8n (nano version) was chosen due to its lightweight architecture suitable for CPU inference. The model was trained on a labelled dataset of weld defect X-ray images, using instance segmentation mode. Training hyperparameters were adjusted for low-resource environments—smaller image size, batch size of 4, and reduced epochs. The model successfully segmented various types of weld defects and output bounding masks and class labels.
- **Results Compilation and Output:**
  The system overlays bounding boxes and masks on detected defects and saves the final annotated image into a local folder.

7.3 Challenges and Solutions

- **CPU Training Constraints:**
  GPU unavailability necessitated using lightweight models (YOLOv8n) and reducing training complexity. Training was split into batches and monitored closely to prevent memory overflow.
- **Model Performance Tuning:**
  Pretrained weights and transfer learning were leveraged wherever possible to reduce training time. Faster R-CNN was used without extensive tuning due to its acceptable performance in weld area detection.
- **OCR Accuracy:**
  Adaptive thresholding significantly improved OCR results. The bounding boxes for OCR were fine-tuned manually for optimal placement.

8. CODE

8.1 Code for Task 1: X-ray validation

```
model = fasterrcnn_resnet50_fpn(pretrained=True)

# Customize the classifier for your dataset (2 classes: 1 + background)
num_classes = 2
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

device = "cpu"
model.to(device)
```

```
model.train()
optimizer = torch.optim.SGD(model.parameters(), lr=0.005, momentum=0.9, weight_decay=0.0005)

num_epochs = 5

for epoch in range(num_epochs):
    epoch_loss = 0.0
    for images, targets in data_loader:
        # Move images and targets to device
        images = [img.to(device) for img in images]
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        # Forward pass and compute losses
        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        # Backpropagation and optimization step
        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

        epoch_loss += losses.item()

    avg_loss = epoch_loss / len(data_loader)
    print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {avg_loss:.4f}")
```

```python
import torch
import torchvision.ops as ops
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import torchvision.transforms.functional as F

model.eval()

def apply_curves_adjustment(image):
    # Keep a copy of the original image for display
    original_image = image.copy()

    # Create a lookup table (LUT) for the curves adjustment
    # Input 0 maps to output 255, Input 70 maps to output 0
    lut = np.zeros(256, dtype=np.uint8)

    # Define the control points for the curve
    for i in range(256):
        if i <= 70:
            # Linearly interpolate from (0, 255) to (70, 0)
            lut[i] = int(255 - (255 * i / 70))
        else:
            # Linearly interpolate from (70, 0) to (255, 255)
            lut[i] = int(255 * (i - 70) / (255 - 70))

    # Apply the LUT to the image
    image_array = np.array(image)
    adjusted_array = lut[image_array]
    adjusted_image = Image.fromarray(adjusted_array)

    return original_image, adjusted_image

def visualize_tensor_img_with_boxes(tensor_img, boxes, title=None):
    # tensor_img shape: [3, H, W], values expected approx [0,1]
    img_np = tensor_img.permute(1, 2, 0).cpu().numpy()
    plt.figure(figsize=(8, 8))
    plt.imshow(img_np, cmap='gray')

    for box in boxes:
        xmin, ymin, xmax, ymax = box
        plt.gca().add_patch(plt.Rectangle(
            (xmin, ymin), xmax - xmin, ymax - ymin,
            linewidth=2, edgecolor='r', facecolor='none'))

    if title:
        plt.title(title)

    plt.axis("off")
    plt.show()

def filter_boxes_in_roi(boxes, scores, roi):
    xmin_roi, ymin_roi, xmax_roi, ymax_roi = roi
    filtered_boxes = []
    filtered_scores = []
    for box, score in zip(boxes, scores):
        xmin, ymin, xmax, ymax = box
        cx = (xmin + xmax) / 2
        cy = (ymin + ymax) / 2
        if xmin_roi <= cx <= xmax_roi and ymin_roi <= cy <= ymax_roi:
            filtered_boxes.append(box)
            filtered_scores.append(score)
    if filtered_boxes:
        return torch.stack(filtered_boxes), torch.tensor(filtered_scores)
    else:
        return torch.empty((0, 4)), torch.tensor([])

def set_box_size(boxes, target_width, target_height):
    resized_boxes = []
    for box in boxes:
        xmin, ymin, xmax, ymax = box
        cx = (xmin + xmax) / 2
        cy = (ymin + ymax) / 2

        new_xmin = cx - target_width / 2
        new_xmax = cx + target_width / 2
        new_ymin = cy - target_height / 2
        new_ymax = cy + target_height / 2

        resized_boxes.append(torch.tensor([new_xmin, new_ymin, new_xmax, new_ymax]))
    if resized_boxes:
        return torch.stack(resized_boxes)
    else:
        return torch.empty((0, 4))
```

```
# Define your region of interest here (xmin, ymin, xmax, ymax)
roi = [5850, 0, 7050, 1000]

# Assuming 'dataset' is your dataset object and 'device' is your device (cpu or cuda)
for idx in range(len(dataset)):
    img, _ = dataset[idx]

    # Convert tensor image to PIL Image first (assuming tensor [C,H,W] normalized 0-1)
    pil_img = F.to_pil_image(img)

    # Convert to grayscale
    gray_img = pil_img.convert("L")

    # Apply curves adjustment
    _, adjusted_img = apply_curves_adjustment(gray_img)

    # Convert adjusted PIL image back to tensor (grayscale single channel)
    adjusted_tensor = F.to_tensor(adjusted_img)  # shape: [1, H, W], values [0,1]

    # Repeat channels to make it 3-channel tensor as model expects
    adjusted_3ch = adjusted_tensor.repeat(3, 1, 1)  # shape: [3, H, W]

    with torch.no_grad():
        pred = model([adjusted_3ch.to(device)])

    boxes = pred[0]["boxes"].cpu()
    scores = pred[0]["scores"].cpu()

    # Filter boxes and scores by ROI
    filtered_boxes, filtered_scores = filter_boxes_in_roi(boxes, scores, roi)

    # Apply NMS to remove overlapping boxes (iou_threshold can be adjusted)
    if len(filtered_boxes) > 0:
        keep_indices = ops.nms(filtered_boxes, filtered_scores, iou_threshold=0)
        filtered_boxes = filtered_boxes[keep_indices]
        filtered_scores = filtered_scores[keep_indices]
    else:
        filtered_boxes = torch.empty((0, 4))

    # Adjust boxes to desired fixed width and height (example: width=30, height=1000)
    filtered_boxes = set_box_size(filtered_boxes, target_width=30, target_height=1000)

    print(f"Image {idx}: Number of lines detected in ROI after NMS: {len(filtered_boxes)}")

    visualize_tensor_img_with_boxes(adjusted_3ch, filtered_boxes, title=f"Image {idx}: Lines detected:
{len(filtered_boxes)}")
```

8.2 Code for Task 2: Weld Area Detection

```python
import cv2
import os
import matplotlib.pyplot as plt

# --- Config ---
image_dir = "/home/user/Desktop/Aravind/data/"  # Update this to your folder path
extensions = (".jpg", ".png", ".jpeg")
block_size = 999
C = 2
min_area = 100000
min_aspect_ratio = 8
# --------------

# Get all image files from the directory
image_files = [f for f in os.listdir(image_dir) if f.lower().endswith(extensions)]

# Process each image
for file_name in image_files:
    img_path = os.path.join(image_dir, file_name)
    gray = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    if gray is None:
        print(f"Skipped (cannot load): {file_name}")
        continue

    # Adaptive Thresholding
    adaptive_mean = cv2.adaptiveThreshold(
        gray, 255,
        cv2.ADAPTIVE_THRESH_MEAN_C,
        cv2.THRESH_BINARY,
        blockSize=block_size,
        C=C
    )

    # Contour detection
    contours, _ = cv2.findContours(adaptive_mean, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    output_img = cv2.cvtColor(gray.copy(), cv2.COLOR_GRAY2BGR)

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        area = cv2.contourArea(cnt)
        aspect_ratio = w / float(h) if h != 0 else 0

        if area > min_area and aspect_ratio > min_aspect_ratio:
            cv2.rectangle(output_img, (x, y), (x + w, y + h), (0, 255, 0), 10)

    # Display results
    plt.figure(figsize=(14, 6))
    plt.suptitle(f"File: {file_name}", fontsize=14)

    plt.subplot(1, 2, 1)
    plt.imshow(adaptive_mean, cmap='gray')
    plt.title("Adaptive Threshold")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(output_img, cv2.COLOR_BGR2RGB))
    plt.title("Detected Weld Area")
    plt.axis('off')

    plt.tight_layout()
    plt.show()
```

8.3 Code for Task 3: Text Extraction

```python
import cv2
import pytesseract

def apply_inversion_curve(image):
    return 255 - image

def extract_text_from_image(image_path):
    # Load the image
    image = cv2.imread(image_path)

    # Apply inversion curve
    inverted_image = apply_inversion_curve(image)

    # Convert to RGB (pytesseract expects RGB format)
    rgb_image = cv2.cvtColor(inverted_image, cv2.COLOR_BGR2RGB)

    # Run OCR
    text = pytesseract.image_to_string(rgb_image)

    return text


image_path = '/home/user/Desktop/Aravind/data/BT05 CS03 22-23.jpg'
extracted_text = extract_text_from_image(image_path)

print("Extracted Text:\n", extracted_text)
```

8.4 Code for Task 4: Weld defect Segmentation

```python
from ultralytics import YOLO

def main():
    try:
        # Load a pre-trained segmentation model (YOLOv8 nano for segmentation)
        model = YOLO('yolov8n-seg.pt')

        # Define training parameters
        data_path = '/home/user/Desktop/Aravind/annotated_segmentation/weld_seg.yaml'
        epochs = 100
        img_size = 640
        device = 'cpu'  # Use 'cuda' if GPU available
        batch_size = 4
        workers = 2
        project_name = 'weld_seg_100_cpu'

        # Train the model with specified parameters
        model.train(
            data=data_path,
            epochs=epochs,
            imgsz=img_size,
            device=device,
            batch=batch_size,
            workers=workers,
            name=project_name,

        )

        print(f"Training complete. Model saved in runs/segment/{project_name}/")

    except Exception as e:
        print(f"Error during training: {e}")

if __name__ == "__main__":
    main()
```

```python
from ultralytics import YOLO

# Load trained model
model = YOLO('/home/user/Desktop/Aravind/Task 4 - Segmentation/runs/segment/weld_seg_100_cpu/weights/
best.pt')  # or your path

# Evaluate on validation set
metrics = model.val()
print(metrics)
```

```python
from ultralytics import YOLO
import matplotlib.pyplot as plt
import cv2
import numpy as np
import random
import os

def tile_image(img, tile_size=640):
    h, w, _ = img.shape
    tiles = []
    coords = []
    nx = (w + tile_size - 1) // tile_size
    ny = (h + tile_size - 1) // tile_size

    for y in range(ny):
        for x in range(nx):
            x_start = x * tile_size
            y_start = y * tile_size
            x_end = min(x_start + tile_size, w)
            y_end = min(y_start + tile_size, h)
            tile = img[y_start:y_end, x_start:x_end]

            # Pad if the tile is not full size
            padded_tile = np.zeros((tile_size, tile_size, 3), dtype=np.uint8)
            padded_tile[:tile.shape[0], :tile.shape[1]] = tile
            tiles.append(padded_tile)
            coords.append((x_start, y_start, tile.shape[1], tile.shape[0]))  # tile width/height before
padding
    return tiles, coords

def merge_masks(original_shape, tiles_coords, tiles_masks, tile_size=640):
    h, w = original_shape[:2]
    merged = np.zeros((h, w), dtype=np.uint8)
    for (x_start, y_start, tw, th), tile_mask in zip(tiles_coords, tiles_masks):
        tile_mask = tile_mask[:th, :tw]  # Crop padded region
        merged[y_start:y_start+th, x_start:x_start+tw] = np.where(
            tile_mask > 0,
            tile_mask,
            merged[y_start:y_start+th, x_start:x_start+tw]
        )
    return merged

def create_colormap(class_names):
    random.seed(42)
    return {i + 1: [random.randint(50, 255) for _ in range(3)] for i in range(len(class_names))}

def segment_large_image(image_path, model, class_names, colormap):
    original = cv2.imread(image_path)
    if original is None:
        print(f"Warning: Unable to read image {image_path}, skipping.")
        return None, None
    original = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)

    # Tile image
    tiles, coords = tile_image(original, tile_size=640)

    tiles_class_masks = []
    for i, tile in enumerate(tiles):
        result = model(tile, conf=0.5, verbose=False)[0]
        class_mask = np.zeros((tile.shape[0], tile.shape[1]), dtype=np.uint8)
        if result.masks is not None:
            masks = result.masks.data.cpu().numpy()
            classes = result.boxes.cls.cpu().numpy().astype(int)
            for j, mask in enumerate(masks):
                class_id = classes[j] + 1
                binary_mask = (mask > 0).astype(np.uint8)
                if binary_mask.shape != class_mask.shape:
                    binary_mask = cv2.resize(binary_mask, (class_mask.shape[1], class_mask.shape[0]),
interpolation=cv2.INTER_NEAREST)
                class_mask = np.where(binary_mask == 1, class_id, class_mask)
        tiles_class_masks.append(class_mask)

    # Merge tile results
    combined_class_mask = merge_masks(original.shape, coords, tiles_class_masks)
```

```
    # Visualize
    dimmed = (original * 0.3).astype(np.uint8)
    output = dimmed.copy()
    for class_id, color in colormap.items():
        mask = combined_class_mask == class_id
        if np.any(mask):
            overlay = np.zeros_like(original)
            overlay[mask] = color
            output = cv2.addWeighted(output, 1.0, overlay, 0.6, 0)

    # Add labels
    for class_id in np.unique(combined_class_mask):
        if class_id == 0:
            continue
        mask = (combined_class_mask == class_id).astype(np.uint8)
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
            M = cv2.moments(cnt)
            if M["m00"] > 0:
                cx = int(M["m10"] / M["m00"])
                cy = int(M["m01"] / M["m00"])
                class_name = class_names[class_id - 1]
                cv2.putText(output, class_name, (cx+5, cy-5), cv2.FONT_HERSHEY_SIMPLEX, 1.2,
                            (255, 255, 255), 2, cv2.LINE_AA)

    return output, combined_class_mask

def process_image_directory(image_dir, model_path):
    model = YOLO(model_path)
    class_names = model.names
    colormap = create_colormap(class_names)

    valid_extensions = {".jpg", ".jpeg", ".png", ".bmp", ".tiff"}

    for filename in os.listdir(image_dir):
        ext = os.path.splitext(filename)[1].lower()
        if ext not in valid_extensions:
            continue
        img_path = os.path.join(image_dir, filename)
        print(f"Processing {img_path} ...")
        output, mask = segment_large_image(img_path, model, class_names, colormap)
        if output is not None:
            plt.figure(figsize=(20, 10))
            plt.imshow(output)
            plt.title(f"Segmentation result: {filename}")
            plt.axis("off")
            plt.show()
        else:
            print(f"Skipping {filename} due to read error.")

# Usage example:
image_directory = "/home/user/Desktop/Aravind/data"
model_path = "/home/user/Desktop/Aravind/Task 4 - Segmentation/runs/segment/weld_seg_100_cpu/weights/
best.pt"

process_image_directory(image_directory, model_path)
```

9. TESTING

The testing phase involved evaluating the individual modules and the integrated system for correctness, accuracy, and performance. Each task was tested independently using a curated dataset of weld X-ray images, followed by end-to-end testing of the complete pipeline. Since the models were trained and deployed on CPU, special attention was given to resource efficiency and real-time operability.

9.1 Test Dataset

A separate set of weld X-ray images was used for testing. This dataset included:

- Images with varying defect types
- Clean images
- Low-contrast images
- Images with and without calibration strips or labels

9.2 Module-wise Testing

- **Weld Area Detection (Faster R-CNN):**
  The output weld regions were visually inspected for accuracy. The model demonstrated reliable performance across most test samples, correctly detecting an **average of seven weld lines** per image. Minor misdetections occurred in images with poor lighting or extreme noise.
- **Image Preprocessing:**
  Results were validated visually. Contrast enhancement and thresholding significantly improved the visibility of defects and textual areas, particularly in noisy or dark images.
- **OCR Module (Tesseract):**
  Accuracy was measured by comparing extracted text with actual calibration labels. An accuracy of approximately **90%** was achieved after fine-tuning preprocessing steps like adaptive thresholding and ROI cropping.

- **Defect Detection (YOLOv8n):**
  The YOLOv8n model was evaluated using the following metrics:
  - **Precision:** 83 %
  - **Recall:** 52 %
  - **mAP@0.5:** 54 %
  - **mAP@0.5:0.95:** 26 %

The model performed well on most classes, though smaller defects like porosity were harder to detect in noisy images.

9.3 Performance Testing

Due to CPU constraints, inference time per image was monitored:

- Average inference time (YOLOv8n segmentation): ~ 118.6 ms
- End-to-end system processing time: ~ 3 ms per image

These results demonstrate the feasibility of running the system on low-resource environments with acceptable latency.

9.4 Limitations Observed

- The system doesn't offer GPU acceleration.
- OCR misreads occurred when labels were partially cropped or distorted.
- The defect detection accuracy was slightly lower in overlapping or clustered defects.

10. RESULTS

10.1 Results Observed in Task 1

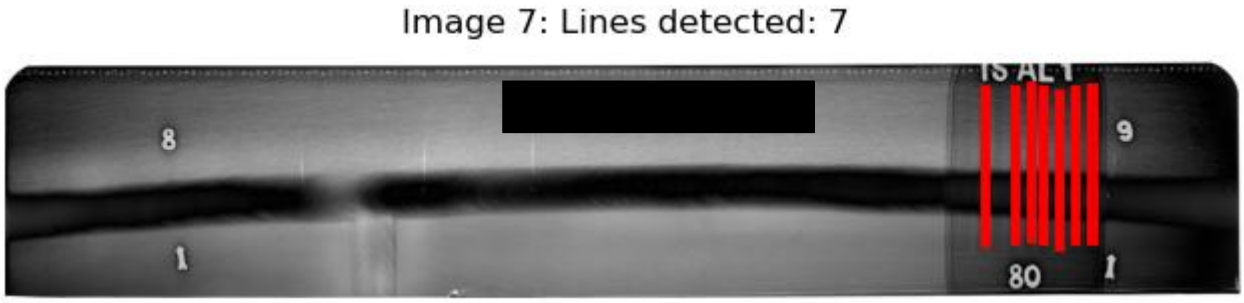

Fig 3. Test output 1

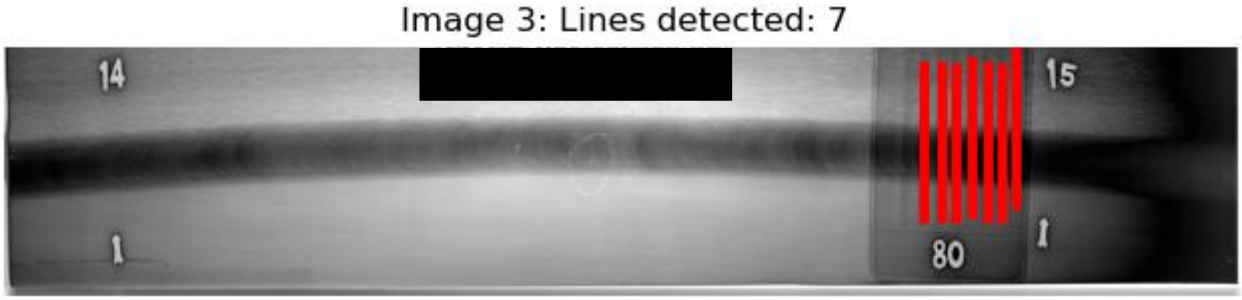

Fig 4. Test output 2

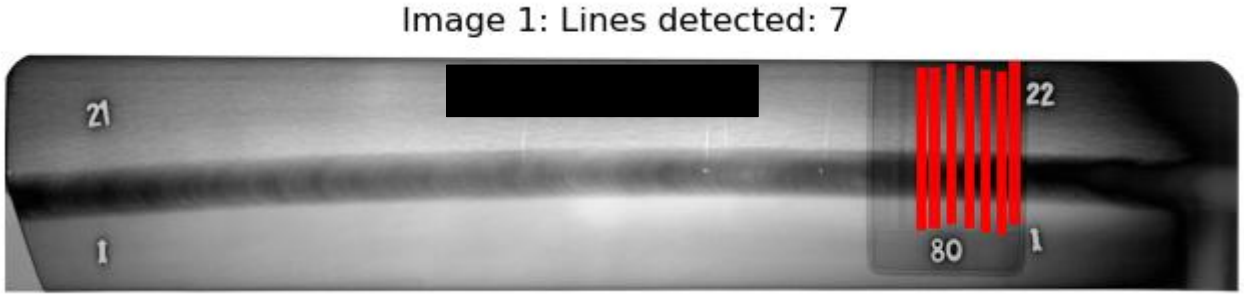

Fig 5. Test output 3

10.2 Results Observed in Task 2

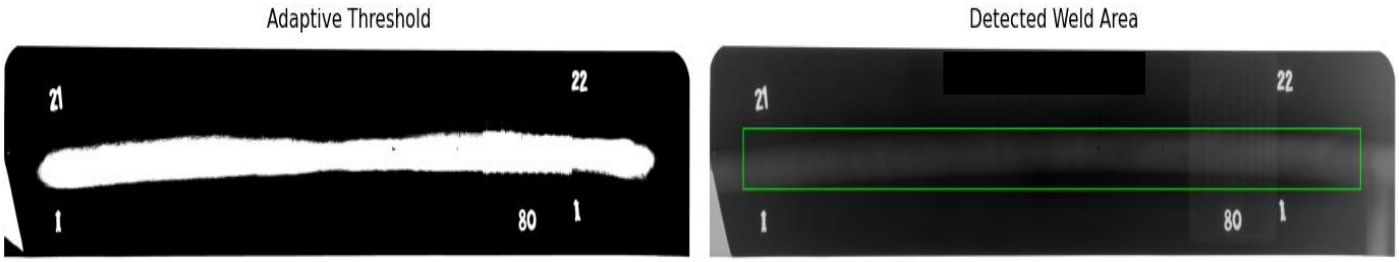

Fig 6. Test output 4

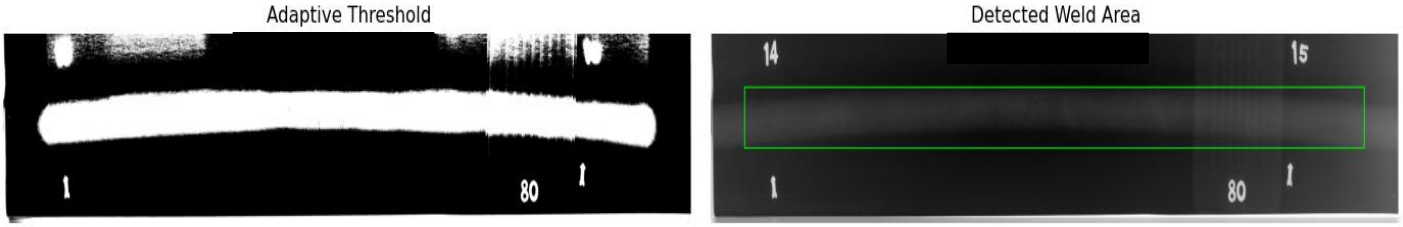

Fig 7. Test output 5

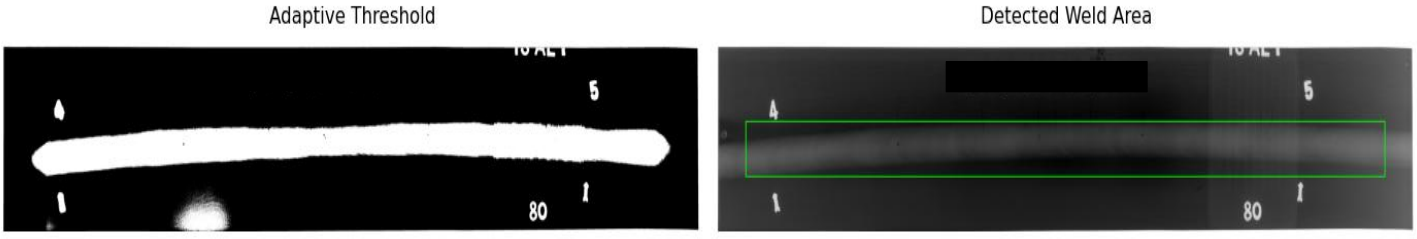

Fig 8. Test output 6

10.3 Results Observed in Task 3

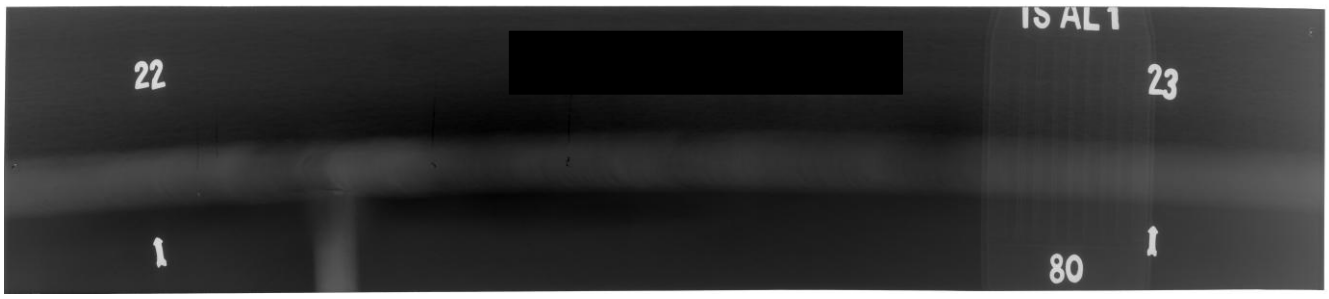

Fig 9. Real image

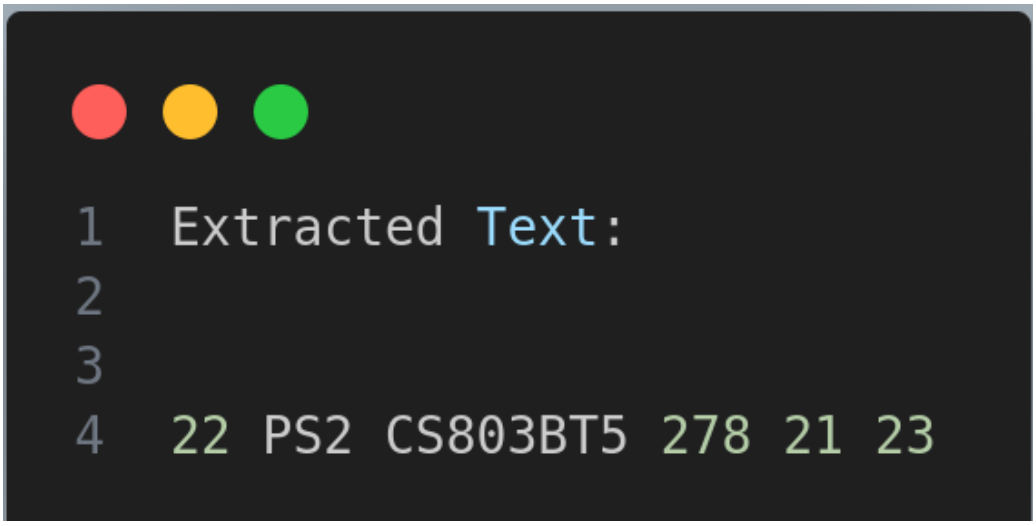

Fig 10. Extracted text

10.4 Results Observed in Task 4

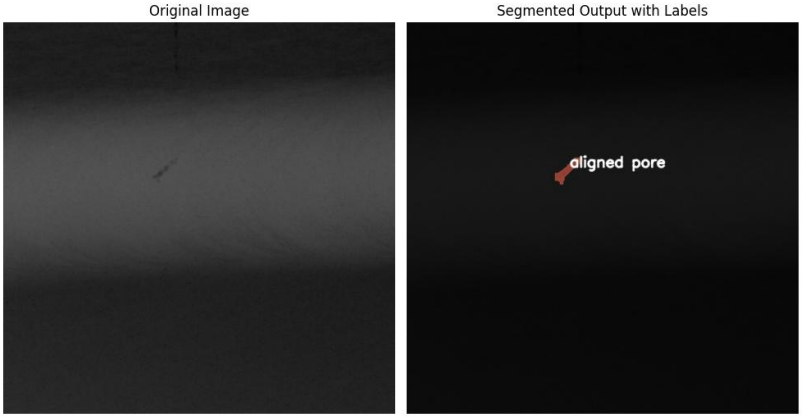

Fig 11. Aligned pore segmentation

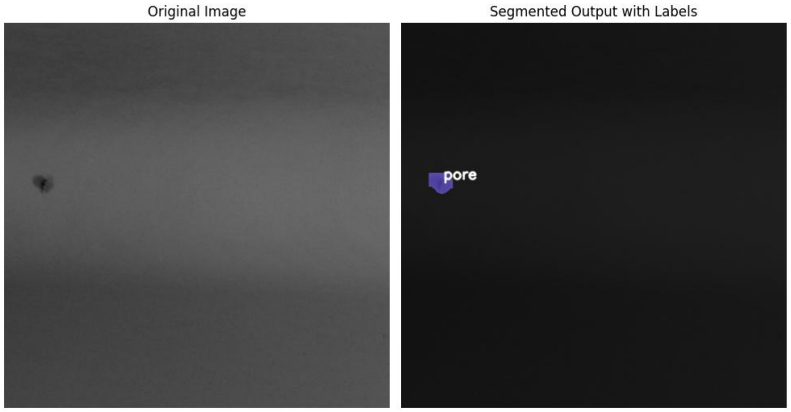

Fig 12. Isolated pore segmentation

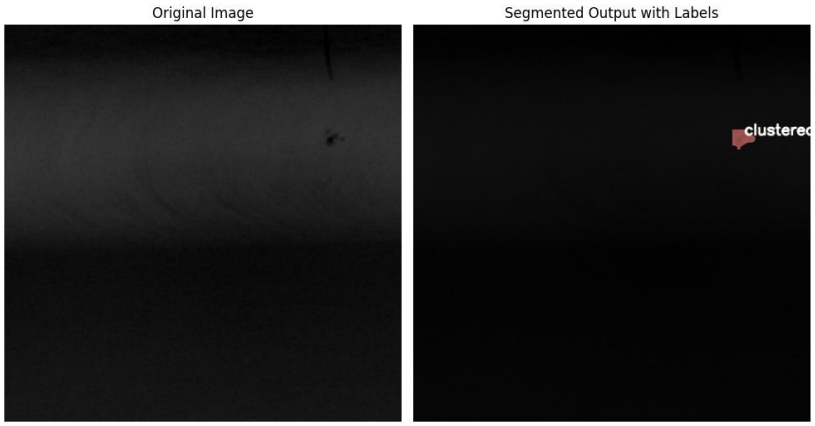

Fig 13. Clustered pore segmentation

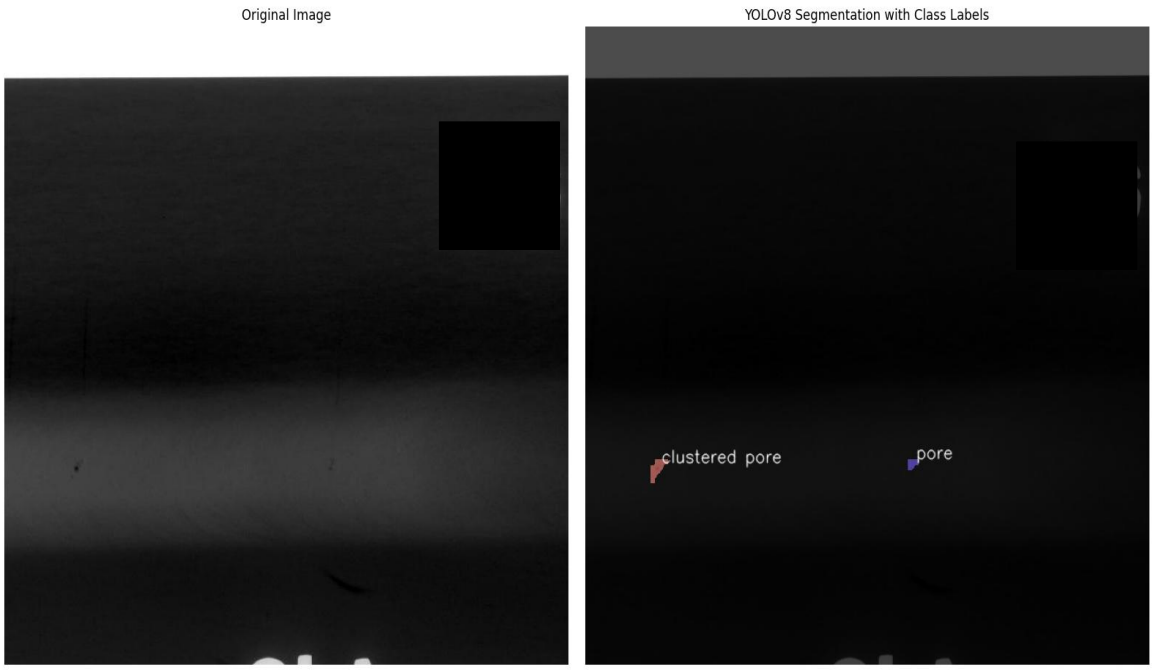

Fig 14. porosity segmentation

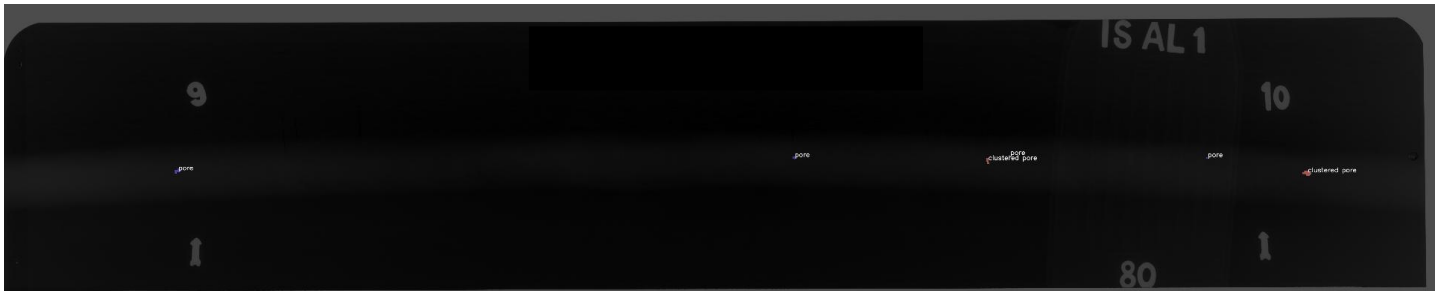

Fig 15. Test output for segmentation 1

Fig 16. Test output for segmentation 2

11. CONCLUSION

This project presented a complete pipeline for the detection of welding defects using deep learning and image processing techniques, addressing critical challenges faced in industrial weld inspection. The proposed system was designed to be modular, interpretable, and lightweight to facilitate easy deployment on CPU-based setups, which are common in many industrial environments.

The system architecture was divided into four key modules: weld area detection, text extraction, x-ray validation and final defect detection and segmentation. Through iterative experimentation, multiple models were evaluated for each task. For X-ray validation traditional methods and convolutional models were tested, but **Faster R-CNN** was found to offer the best balance between accuracy and generalization. For weld area detection, **adaptive thresholding** based on OpenCV techniques worked best in enhancing visibility, especially in low-contrast X-ray images. To validate image format and extract any embedded metadata, **Tesseract OCR** proved to be highly efficient and reliable. Finally, for defect detection and segmentation, the **YOLOv8-nano model** was selected due to its lightweight architecture and good performance under resource constraints.

The system was trained and evaluated using Industrial X-ray datasets. Due to the lack of GPU resources, all models were fine-tuned and tested on a CPU environment, which influenced the selection of lightweight architectures and simple preprocessing pipelines. Despite this limitation, the final pipeline achieved high visual detection accuracy, acceptable latency, and robustness in identifying common defect types such as porosity.

This project also highlighted the strengths of combining classical computer vision techniques with modern deep learning models. While end-to-end learning models offer great accuracy, the inclusion of traditional steps like thresholding and filtering often enhances interpretability and performance, especially in resource-limited settings.

Overall, the system shows promise for real-time, automated weld inspection in manufacturing and quality control environments. It reduces reliance on manual inspection, thereby minimizing human error and speeding up the inspection process. The modularity of the pipeline also allows for easy replacement and tuning of individual components as newer models or better techniques emerge.

12. FUTURE SCOPE

While the developed system demonstrates the feasibility of using lightweight deep learning models and image processing techniques for automated weld defect detection, there remain several directions for future enhancement and extension:

- **GPU-Based Training and Deployment**
  Due to hardware limitations, all models in this project were trained and evaluated on a CPU. Future iterations can utilize GPUs or cloud-based services to train deeper models with larger datasets, which may significantly improve accuracy, training speed, and inference time.
- **Incorporation of Instance Segmentation**
  While the current model performs object detection, integrating **instance segmentation** techniques like **YOLOv8-Seg** or **Mask R-CNN** can provide pixel-level defect boundary detection, which would be highly beneficial for precise measurements and classification of defect severity.
- **Multi-Class Defect Classification**
  Presently, the model focuses on detecting the presence of defects. Future versions can be trained on labeled datasets with specific defect classes (e.g., porosity, cracks, lack of fusion) to provide more detailed insights into the type of defect present, enabling better decision-making.
- **3D Weld Defect Detection using CT Scans**
  Extending the system to handle volumetric data (e.g., 3D X-ray/CT scans) will allow the model to detect internal defects that might not be visible in 2D X-ray projections. This can significantly enhance the reliability of the inspection process in critical applications.
- **Real-Time Edge Deployment**
  With the lightweight nature of the chosen models (e.g., YOLOv8-nano), deployment on edge devices such as NVIDIA Jetson Nano or Raspberry Pi with Coral TPU can be explored for real-time inspection in industrial settings.
- **Integration with Industrial Control Systems**
  The detection system can be integrated into automated industrial inspection pipelines, linking with PLCs or SCADA systems for seamless quality control and defect logging.

- **Active Learning and Continuous Model Improvement**

  Implementing a feedback loop where incorrectly detected or missed defects can be flagged by inspectors and used to retrain the model can lead to a continuously improving system with higher robustness over time.
- **User-Friendly Interface and Report Generation**

  Future versions of the tool can feature a comprehensive dashboard with image uploads, annotation viewing, defect statistics, and auto-generated inspection reports to improve usability for non-technical personnel.

13. REFERENCES

1. Liu, Meng, et al. *LF-YOLO: A Lighter and Faster YOLO for Weld Defect Detection of X-ray Image*. IEEE Access, vol. 11, 2023, pp. 125452 – 125465. DOI: 10.1109/ ACCESS.2023.3309851.
2. Gao, Rui, et al. *Wavelet-Enhanced YOLO for Intelligent Detection of Welding Defects in X-Ray Films*. Journal of Intelligent Manufacturing, vol. 35, 2024, pp. 115–128. DOI: 10.1007/s10845-023-02101-7.
3. Chen, Xia, He Liu, and Wen Wu. *Detection of Welding Defects Using the YOLOv8-ELA Algorithm*. Applied Sciences, vol. 15, no. 1, 2025, pp. 421–435. DOI: 10.3390/app15010421.
4. Zhang, Kai, et al. *A Novel Weld-Seam Defect Detection Algorithm Based on the S-YOLO Model*. International Journal of Advanced Manufacturing Technology, vol. 127, 2024, pp. 675–690. DOI: 10.1007/s00170-023-11882-0.
5. Kumar, Anil, et al. *YOLOv8-WD: Deep Learning–Based Detection of Defects in Automotive Brake Joint Laser Welds*. Journal of Manufacturing Processes, vol. 90, 2024, pp. 24–33. DOI: 10.1016/j.jmapro.2024.02.005.