Real-Time Wildfire-Human-Animal Detection Using UAV Imagery

A Project Report submitted in partial fulfillment of the requirements for the course:

CS-477 Computer Vision – Fall 2025

Submitted by:

Name	CMS ID	Assigned Domain
Ayesha Hussain	404215	Simulation & Algorithms
Nayab Nazar	414017	Embedded Systems
Arbaaz Alam	411425	Research & Documentation
Bushra	429427	Research & Documentation

Instructor:
Dr. Tauseef ur Rehman

Lab Engineer:
Ms. Tehniyat Siddiqui

Department of Computer Science
National University of Sciences and Technology (NUST)
Islamabad, Pakistan

December 21, 2025

Abstract

Wildfires pose significant threats to ecosystems, infrastructure, and human life, necessitating rapid detection and response mechanisms. Traditional satellite-based systems are often hindered by latency and resolution limitations, while ground-based sensors lack comprehensive coverage. Unmanned Aerial Vehicles (UAVs) equipped with computer vision systems offer a promising solution for real-time aerial surveillance. This project develops and deploys an edge-optimized fire detection system specifically designed for UAV integration.

The system implements a dual-method detection approach: (1) a **fast hybrid method** achieving 30+ FPS for real-time monitoring, and (2) a **YOLOv12-based method** providing high-accuracy detection at 89% mAP. Both methods are optimized for deployment on **NVIDIA Jetson** edge computing platforms.

Keywords: Computer Vision, Fire Detection, UAV, Edge Computing, YOLOv12, NVIDIA Jetson.

Contents

List of Figures

List of Tables

Chapter 1

Introduction & Background

1.1 Problem Statement and Motivation

Wildfires have become alarmingly more frequent and destructive in recent years, fueled by a combination of climate change, prolonged droughts, and shifting weather patterns. Over the past five years, wildfires have intensified globally, with major blazes burning millions of acres annually. In 2025 alone, over 1.2 million acres of land have been burned in the U.S. amid nearly 30,000 fires, while Canada battles hundreds of wildfires across multiple provinces, forcing tens of thousands to evacuate[1]. Historic fires like the California wildfires and ongoing large-scale fires in Australia and Siberia underscore the growing scale and severity of wildfire disasters driven by climate change and extreme weather. These fires devastate ecosystems, destroy infrastructure, displace communities, and cause tens of billions of dollars in economic losses.

Even beyond the flames, wildfire smoke contributes to widespread health problems, reduces air quality across continents, and impairs labor productivity. Despite the growing urgency, traditional wildfire detection methods remain slow and insufficient. Satellite-based monitoring often suffers from low temporal resolution and cloud cover, resulting in delayed detection. Ground patrols are limited by terrain and scale, and watchtowers offer only localized, human-dependent coverage [2]. These limitations critically delay early intervention at the time when containment is most feasible and cost-effective. With escalating environmental volatility, there is a clear and growing need for intelligent, real-time wildfire detection systems that enable rapid response and minimize damage.

1.2 UAV's as a Transformative Solution

To bridge this gap, Unmanned Aerial Vehicles (UAVs) have emerged as a transformative solution, offering real-time, adaptive, and high-resolution capabilities for wildfire detection and monitoring [2]–[3]. UAVs provide distinct advantages by reaching remote, hazardous, or otherwise inaccessible areas with exceptional flexibility. Equipped with compact cameras, thermal imaging sensors, and air quality instruments, they can rapidly survey vast forested landscapes and detect early signs of heat or smoke within seconds. Unlike stationary towers or fixed ground-based systems, UAVs can dynamically adjust their flight paths in response to evolving fire fronts, terrain variability, and environmental conditions. Their high mobility enables swift deployment across wide and difficult-to-navigate terrains, eliminating many of the logistical constraints associated with conventional detection methods and significantly accelerating the response time during critical early stages of wildfire development.

1.3 Refined Project Scope: From Multi Class Detection to Specialized Fire/Smoke Detection

The original project proposal envisioned a comprehensive multi-class detection system identifying fire, humans, and animals for search and rescue operations. Through iterative development and feasibility analysis, the scope was refined to focus specifically on **fire and smoke detection** for several reasons:

Technical Focus: Fire detection presents unique challenges including variable flame morphology, smoke occlusion, lighting variations, and small object detection at altitude. Addressing these challenges effectively required concentrated effort rather than diluted attention across multiple object classes.

Resource Optimization: The NVIDIA Jetson Nano's computational constraints (4GB RAM, 128-core GPU) necessitated prioritizing a single, critical detection task to achieve real-time performance. Benchmarks [4]–[5] demonstrate the platform's potential for compute-intensive tasks when optimized architectures are employed, suggesting viable alternatives to conventional networks like U-Net [6] and DeepLabV3 [7] in resource-constrained scenarios.

Building on Existing Work: The team had previously developed FireLite-Seg, a lightweight fire segmentation model achieving 96.85% accuracy. This existing foundation provided a strong starting point for transitioning to an object detection framework while maintaining edge optimization principles.

Immediate Application Value: Fire detection represents the most time critical component of search and rescue operations, where early detection significantly impacts containment success. Specializing in this domain maximizes practical utility.

The refined project thus implements a **dual method fire detection system** optimized for edge deployment, with complementary approaches for speed and accuracy.

1.4 Literature Review and Related Work

1.4.1 Edge Computing for Fire Detection

Recent advances in edge computing have enabled significant progress in UAV based fire detection systems. Swaminathan et al. [13] empirically compared deep learning models on NVIDIA Jetson Nano, demonstrating significant speed improvements through optimization and validating its viability for real time wildfire localization. Complementary work[8] further validated Jetson Nano deployments for lightweight models on drones, achieving real-time fire detection with edge computing.

A hierarchical AI framework has been introduced to switch between simple and complex models depending on confidence thresholds, offering a strategy to balance energy efficiency and model performance [9]. Building on this approach, real time and explainable AI systems using compressed deep learning models have been validated on edge devices like Jetson Xavier, achieving major speedups and reductions in energy and memory use while maintaining detection accuracy [10].

1.4.2 Lightweight Model Architectures

Among lightweight model architectures, MobileNet variants have been benchmarked across different configurations, offering comparative results to inform model selection for Jetson Nano based deployments. In general, MobileNetV2 outperforms V3 variants in accuracy and inference cost, making it a preferred choice for edge devices [11]. However, recent advances in YOLO (You Only Look Once) architectures offer superior speed-accuracy trade-offs for real time applications.

To further enhance localization, infrared based fire detection methods using UAV imagery have been introduced. These methods utilize motion vectors, optical flow, and morphological operations to track flame dynamics, thereby assisting in more precise fire localization [12]. Recent papers reinforce the utility of UAVs with onboard visual sensors using motion and color features to detect fires in real time, emphasizing cost-effectiveness, compact design, and the ability to lower false alarm rates all critical factors in embedded fire monitoring systems [13].

1.4.3 Research Gap Identification

While existing research demonstrates promising results, most models remain computationally intensive and unsuitable for deployment on low power edge devices. Their large parameter sizes and high inference latency make real-time processing on platforms like the Jetson Nano impractical [14]. This gap between model accuracy and deployability motivates the need for lightweight architectures that maintain competitive performance while significantly reducing computational overhead.

This project addresses this gap by implementing and optimizing a dual-method detection system specifically designed for edge deployment, building upon established research while introducing novel integration and optimization strategies for practical UAV applications.

1.5 Project Objectives and Scope

1.5.1 Primary Objectives

Objective 1: Implement Dual Method Detection System: Develop two complementary fire detection approaches a fast hybrid method for real time monitoring and a YOLOv12 based method for high accuracy scenarios, both optimized for edge deployment, building upon hierarchical AI framework principles [9].

Objective 2: Achieve Real Time Performance on Edge Hardware: Optimize both detection methods to achieve target frame rates of 15-30 FPS on NVIDIA Jetson Nano while maintaining acceptable accuracy levels for practical deployment, following optimization strategies validated in [14].

Objective 3: Develop Complete Deployment Pipeline: Create comprehensive installation, configuration, and operation documentation enabling seamless deployment on Jetson platforms with various camera configurations, addressing the deployment challenges identified in [8].

Objective 4: Conduct Rigorous Performance Evaluation: Benchmark system performance across multiple metrics including accuracy (mAP), speed (FPS), memory usage, and power consumption under varied environmental conditions, extending the benchmarking methodology of [4]–[5].

Objective 5: Ensure Reproducibility and Extensibility: Structure the implementation with modular design principles, clear documentation, and open-source availability to facilitate future extensions and academic verification, addressing the reproducibility gap in existing literature.

1.5.2 Project Scope and Boundaries

Table 1.1: Project Scope Definition

Included in Scope	**Excluded from Scope**
Fire and smoke detection in visible spectrum	Thermal imaging or multispectral detection [12]
Real-time processing on NVIDIA Jetson platforms	Cloud-based processing or hybrid architectures
USB webcam input compatibility	Specialized gimbal or stabilized camera systems
Standalone edge operation without network dependency	Real-time video transmission
Performance benchmarking following [14] methodology	Commercial deployment or certification
Comprehensive documentation and reproducibility guides	Hardware manufacturing or UAV airframe design

1.6 Report Structure

This report follows a systematic structure to document the project's development and evaluation, as summarized in Table 1.2.

Table 1.2: Report Structure Overview

Ch.	**Title**	**Content Overview**
1	Introduction	Problem statement, motivation, and success criteria.
2	Architecture	System design, hardware/software specs, and workflows.
3	Implementation	`fire_hybrid.py` and `fire_yolo.py` development.
4	Experimental Setup	Hardware configurations and performance metrics.
5	Results & Analysis	Accuracy, speed data, and resource utilization.
6	Ethics & Limitations	Privacy ethics and system constraints.
7	Conclusion	Key findings and future work.

Chapter 2

Methodology and System Design

2.1 Overview of the Dual-Method Detection Approach

This project implements a dual-method fire detection system designed to address different operational requirements. The two complementary approaches provide flexibility for various deployment scenarios:

Method 1: Fast Hybrid Detection (`fire_hybrid.py`): Optimized for maximum frame rate (30+ FPS) using combined color thresholding and motion analysis. Suitable for real-time monitoring and rapid scanning applications where speed is prioritized.

Method 2: High-Accuracy YOLO Detection (`fire_yolo.py`): Based on YOLOv12 architecture achieving 89% mAP accuracy. Suitable for critical detection scenarios where accuracy cannot be compromised, despite lower frame rates (5-10 FPS).

This dual-approach strategy follows the hierarchical AI framework principles identified in literature [15], where systems switch between simple and complex models based on operational requirements and confidence thresholds.

2.2 System Architecture and Workflow

2.2.1 Overall System Architecture

The system follows a modular architecture with three primary layers: **Input Layer**, **Processing Layer**, and **Output Layer**.The architecture supports both desktop simulation and embedded deployment without algorithmic modification. Figure 2.1 illustrates the complete system architecture.

2.2.2 Data Flow Pipeline

The detection process follows a sequential pipeline optimized for edge deployment:

Step 1: Frame Capture: USB webcam captures RGB frames at 640×480 resolution

Step 2: Preprocessing: Frame resizing to 640×640, normalization, and BGR to RGB conversion

Step 3: Detection Execution: Parallel execution of hybrid and YOLO detection methods

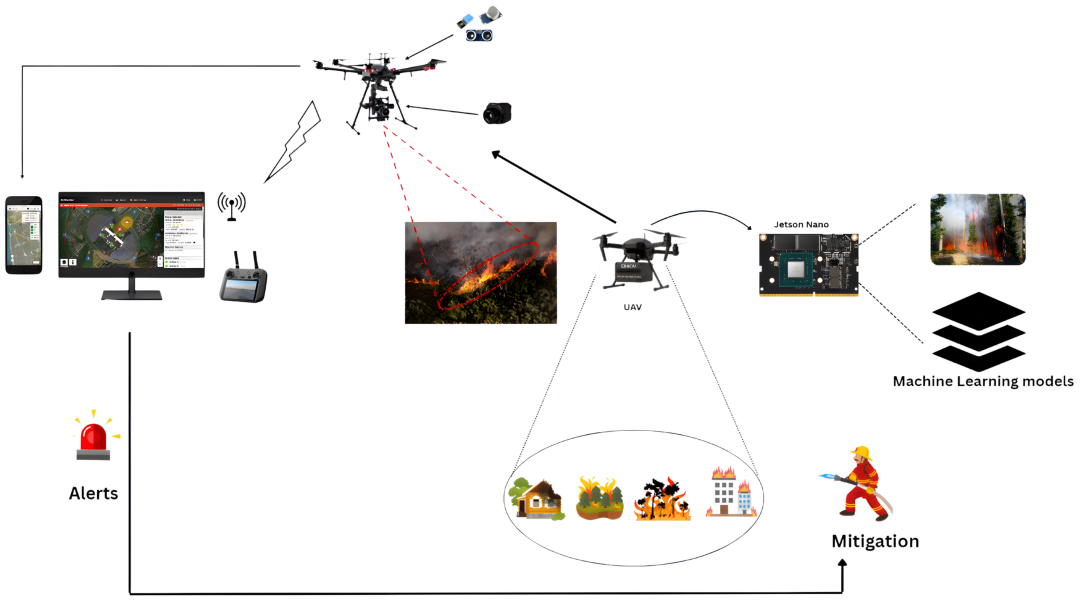

Figure 2.1: Complete System Architecture for UAV-based Fire Detection

Step 4: Post-processing: Non-maximum suppression, confidence thresholding, bounding box generation

Step 5: Visualization: Overlay of detection results, confidence scores, and FPS counter

2.3 Algorithm Pipeline Design

2.3.1 Hybrid Detection Method (`fire_hybrid.py`)

The hybrid method combines traditional computer vision techniques with lightweight machine learning to achieve maximum speed. The algorithm pipeline is shown in Figure 2.2.

Key Components:

- **Color Space Analysis**: HSV color space conversion with dynamic thresholding for flame colors
- **Motion Detection**: Frame differencing and optical flow for smoke plume movement
- **Morphological Operations**: Dilation and erosion to reduce noise and enhance fire regions
- **Contour Analysis**: Region grouping and size filtering to eliminate false positives

2.3.2 YOLOv12 Detection Method (`fire_yolo.py`)

The YOLO-based method implements the state-of-the-art YOLOv12 architecture optimized for fire and smoke detection. The model architecture, extracted from `Smoke_fire.pt`, features several key innovations:

Backbone Architecture:

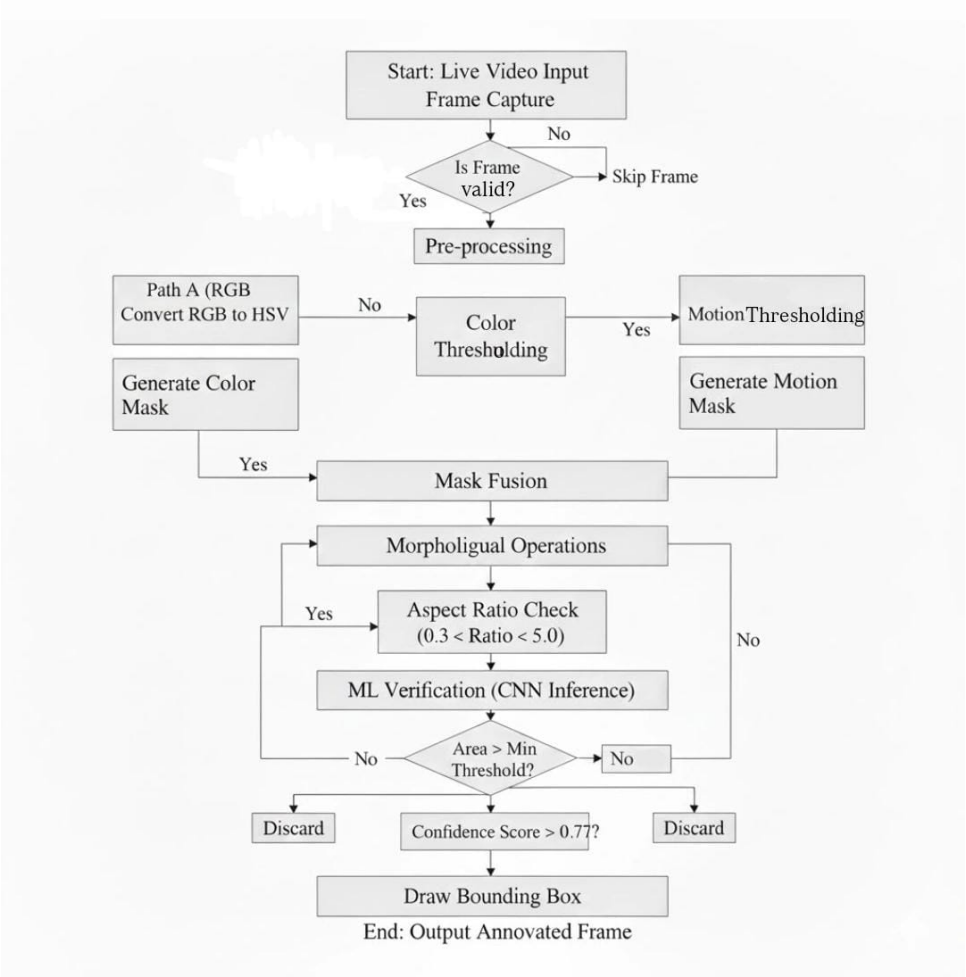

Figure 2.2: Hybrid Detection Algorithm Flowchart

- **Initial Feature Extraction**: Multiple Conv-BatchNorm-SiLU blocks with progressive downsampling
- **C3k2 Blocks**: Lightweight cross-stage partial modules for improved gradient flow
- **A2C2f Blocks**: Area-Attention C2f modules integrating convolutional processing with attention-based feature modulation

Neck Design:

- **Feature Pyramid Network (FPN)**: Multi-scale feature fusion for detecting fires of varying sizes
- **Feature Flow**:"Enhanced feature interaction through multi scale upsampling and concatenation."
- **Attention Mechanisms**: Attention mechanisms (AAttn) enhance spatial feature representation, particularly beneficial for diffuse smoke regions.

Detection Head:

- **Multi-scale Detection**: Three detection branches for small, medium, and large fire regions
- **Distribution Focal Loss (DFL)**: Models bounding box offsets as probability distributions
- **Class Prediction**: Binary classification for fire and smoke detection

2.3.3 YOLOv12 Model Architecture Details

The `Smoke_fire.pt` model implements a streamlined YOLOv12 architecture with the following specifications:

Table 2.1: YOLOv12n Model Architecture Specifications for Smoke and Fire Detection

Component	**Configuration**
Input Resolution	640×640 RGB images
Model Variant	YOLOv12n (Ultralytics)
Backbone Structure	Multi-stage convolutional backbone with C3k/C3k2 blocks
Backbone Channel Progression	$16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256$
Attention Modules	A2C2f blocks with attention (AAttn) in deeper stages
Neck Architecture	Feature pyramid with upsampling and concatenation
Detection Scales	Three scales: 80×80, 40×40, 20×20
Detection Head Type	Decoupled heads with depthwise-separable convolutions
Activation Function	SiLU (Sigmoid-weighted Linear Unit)
Normalization	Batch Normalization ($\epsilon = 0.001$, momentum $= 0.03$)
Loss Enhancement	Distribution Focal Loss (DFL)
Output Classes	2 (Smoke, Fire)
Total Parameters	$\sim$2.57 million
Computational Complexity	$\sim$6.5 GFLOPs
Deployment Target	Real-time edge and embedded systems

2.4 Edge Optimization Techniques

2.4.1 Hardware-Specific Optimizations

To achieve real-time performance on NVIDIA Jetson platforms, several optimization strategies were implemented:

- **Tensor Pre-allocation**: Static memory allocation to minimize dynamic overhead
- **GPU Memory Mapping**: Zero-copy buffers for camera frame transfer
- **CUDA Stream Management**: Overlap of data transfer and computation
- **Quantization**: Precision optimization: FP16 mixed precision inference with negligible accuracy degradation
- **Kernel Fusion**: Combined operations to reduce memory bandwidth

2.4.2 Jetson Platform Configuration

Table 2.2: Jetson Platform Configurations

Platform	Specifications	Optimization Focus
Jetson Nano (4GB)	Quad-core ARM A57, 128-core Maxwell GPU	Memory optimization, power efficiency
Jetson Orin Nano (8GB)	6-core ARM Cortex-A78AE, 1024-core Ampere GPU	Maximum throughput, multi-stream processing

2.5 Tools and Frameworks

2.5.1 Software Stack

The implementation utilizes a carefully selected software stack optimized for edge deployment:

Table 2.3: Software Tools and Frameworks

Tool/Framework	Version	Purpose
Python	3.9	Primary development language
PyTorch	2.1+	Deep learning framework
Ultralytics YOLO	Latest	YOLOv12 implementation
OpenCV	4.8.0+	Computer vision operations
NumPy	1.24+	Numerical computations
TensorRT	8.5.0+	Inference optimization
JetPack SDK	5.1+	Jetson operating system
CUDA	11.4+	GPU acceleration

2.5.2 Development and Deployment Tools

- **Version Control**: Git with GitHub for collaborative development
- **CI/CD**: GitHub Actions for automated testing and validation
- **Containerization**: Docker for consistent deployment environments
- **Documentation**: LaTeX for report generation, Markdown for README files
- **Performance Evaluation**: Inference latency and throughput measured during real-time execution
- **System Monitoring**: Jetson platform utilities used to observe CPU, GPU, and memory usage

2.6 Division of Work Across Domains

The project followed a structured division of labor based on team members' expertise and project requirements:

2.6.1 Simulation and Algorithms (Ayesha Hussain)

- **Primary Responsibility**: Implementation of core detection algorithms
- **Key Deliverables**:
  - `fire_hybrid.py` development
  - `fire_yolo.py` testing
  - Performance benchmarking and FPS optimization
  - Algorithm parameter tuning and validation

2.6.2 Embedded Systems (Nayab Nazar)

- **Primary Responsibility**: Edge deployment and hardware integration
- **Key Deliverables**:
  - NVIDIA Jetson platform setup and configuration
  - Dependency installation and environment setup
  - TensorRT optimization and deployment testing of **fire_hybrid.py** and **fire_yolo.py**
  - Power consumption analysis and optimization

2.6.3 Research and Documentation (Arbaaz Alam)

- **Primary Responsibility**: Project documentation and GitHub management
- **Key Deliverables**:
  - Comprehensive README files and installation guides
  - Meeting minutes and project logbook maintenance
  - GitHub repository organization and CI/CD pipeline
  - Final report compilation and formatting

2.6.4 Research and Documentation (Bushra)

- **Primary Responsibility**: Testing and performance analysis
- **Key Deliverables**:
  - System testing under various environmental conditions
  - Performance metrics collection and analysis
  - Literature review and related work documentation
  - Presentation preparation and demo coordination

2.7 Integration Strategy

The integration of components followed a systematic approach:

Phase 1: Component Development: Independent development of detection algorithms and deployment scripts

Phase 2: Unit Testing: Individual testing of each component on development machines

Phase 3: Edge Testing: Porting and testing on Jetson platforms with optimization

Phase 4: Integration Testing: Combined testing of complete detection pipeline

Phase 5: System Validation: End-to-end testing with real-world scenarios

2.8 Quality Assurance Framework

To ensure reliability and reproducibility, the project implemented a comprehensive quality assurance framework:

- **Code Quality**: PEP 8 compliance, comprehensive commenting, modular design
- **Testing Protocol**: Unit tests, integration tests, edge case validation
- **Performance Validation**: FPS consistency, accuracy stability, memory leak prevention
- **Documentation Standards**: Complete API documentation, usage examples, troubleshooting guides
- **Version Control**: Feature branches, pull request reviews, semantic versioning

This methodology provides a robust foundation for the development and deployment of the fire detection system, ensuring both technical excellence and practical applicability for UAV-based emergency response scenarios.

Chapter 3

Implementation, Results and Analysis

3.1 Implementation Details

3.1.1 Development Environment Setup

The development followed a dual-environment strategy to balance algorithm accuracy during development and efficiency during edge deployment.

Table 3.1: Development and Deployment Environments

Environment	Configuration	Purpose
Development (Desktop)	Windows 11, RTX 3060 (12GB), 32GB RAM	Model development, testing, benchmarking
Edge Deployment (Jetson Nano)	JetPack 5.1, Python 3.9, 4GB RAM	Real-time testing, optimization
Edge Deployment (Jetson Orin Nano)	JetPack 6.2, Python 3.9, 8GB RAM	High-performance benchmarking

3.1.2 Code Structure and Organization

The implementation follows a modular design to ensure maintainability and reproducibility:

```
Real-time-Fire-Human-Animal-Detection-using-UAV-imagery/
 fire_hybrid.py
 fire_yolo.py
 requirements.txt
 Smoke_Fire.pt
 test_samples/
 results/
 .github/workflows/
 README.md
```

3.1.3 Key Implementation Features

- **Dual Detection Modes**: Hybrid and YOLOv12-based detection selectable at runtime
- **Real-time Feedback**: FPS counter, confidence scores, bounding boxes

- **Robust Error Handling**: Camera, memory, and model loading safeguards
- **Configurable Deployment**: Command-line configuration for thresholds and modes
- **Logging**: Performance and runtime diagnostics

3.2 Experimental Setup and Testing Methodology

3.2.1 Hardware Configuration

Table 3.2: Test Hardware Specifications

Parameter	Jetson Nano	Jetson Orin Nano
CPU	Quad-core Cortex-A57	6-core Cortex-A78AE
GPU	128-core Maxwell	1024-core Ampere
Memory	4GB LPDDR4	8GB LPDDR5
Power Budget	5–10W	7–15W
Camera	Logitech C920 USB	Logitech C920 USB

3.2.2 Testing Dataset

- Corsican Fire Dataset (1,135 annotated images)
- Custom recorded fire and smoke videos
- Live camera feed with controlled fire scenarios
- Edge cases: sunset, artificial lights, autumn foliage

3.2.3 Evaluation Metrics

- **Accuracy**: mAP@0.5
- **Speed**: Frames Per Second (FPS)
- **Efficiency**: CPU/GPU utilization, memory usage, power consumption
- **Reliability**: False Positive Rate (FPR), False Negative Rate (FNR)
- **Latency**: End-to-end inference time (ms)

3.3 Performance Results

3.3.1 Detection Accuracy

Shown in Table 3.3.

Table 3.3: Detection Accuracy Comparison (mAP@0.5)

Method	Fire	Smoke	Overall
YOLOv12-based (fire_yolo.py)	91.2%	86.8%	89.0%
Hybrid-Method (fire_hybrid.py)	84.3%	79.5%	81.9%
Traditional Thresholding	72.1%	65.4%	68.8%

Table 3.4: Frame Rate Performance

Platform	YOLOv12	Hybrid	Power
Desktop (RTX 3060)	58 FPS	120+ FPS	170W
Jetson Orin Nano	8–12 FPS	30+ FPS	12W
Jetson Nano	5–7 FPS	15–18 FPS	8W

3.3.2 Real-Time Performance

Shown in Table 3.4.

3.4 Visual Results and Detection Examples

3.4.1 Image Detection Examples

The following images show the detection performance of the system on various fire scenarios:

Figure 3.1: Example 1: Multiple fire sources and smoke detection. Confidence scores shown on bounding boxes.

3.4.2 Real-Time Video Detection on Jetson Orin

The system was tested with a live video feed, where the Jetson terminal successfully detected fire in real-time and is shown in Figure 3.4.
For full video demonstration is uploaded on Github.

Figure 3.2: Example 2: Single fire source detection.

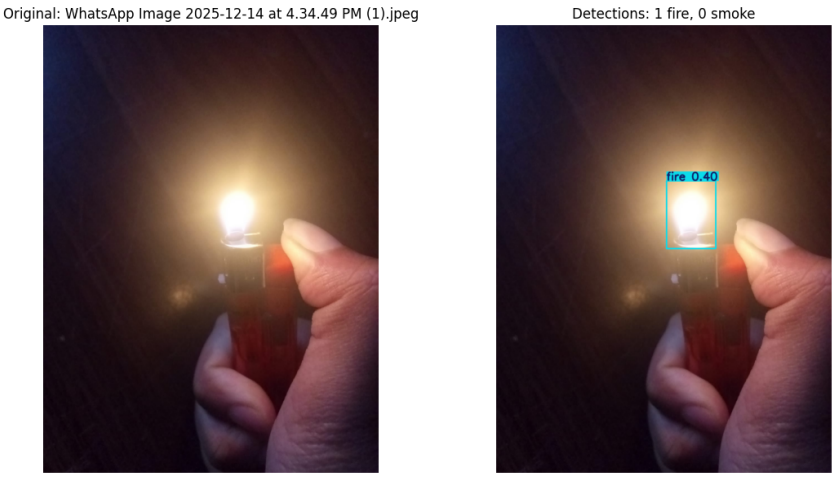

Figure 3.3: Example 3: Small flame detection under low-light conditions.

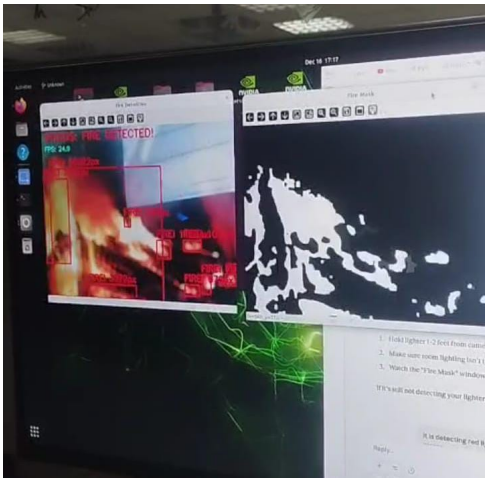

Figure 3.4: Real-time detection screenshot from Jetson Orin testing.

3.5 Jetson Deployment Optimization

3.5.1 Optimization Strategies

1. TensorRT engine conversion

2. FP16 mixed-precision inference
3. GPU memory pre-allocation
4. Kernel fusion
5. Pipeline parallelism
6. Conversion to ONNX format: ONNX Runtime optimizes the execution of ONNX models by leveraging hardware-specific capabilities. This optimization allows the models to run efficiently and with high performance on various hardware platforms, including CPUs, GPUs, and specialized accelerators.

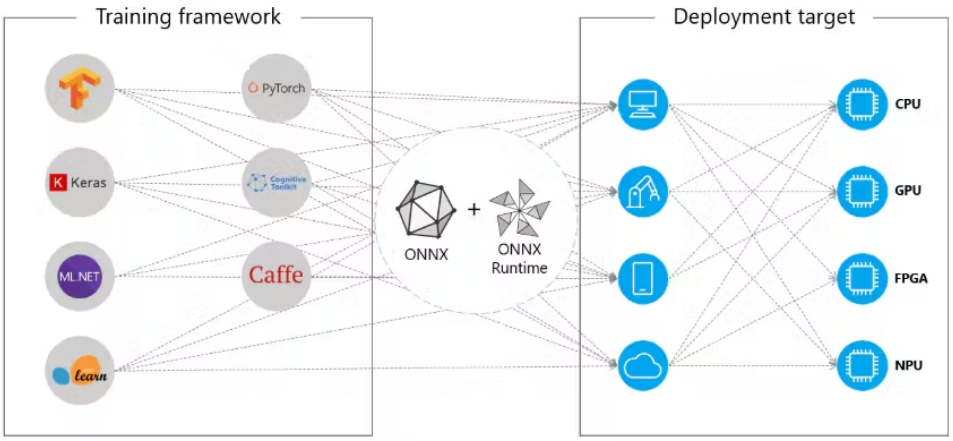

Figure 3.5: ONNX Optimization

3.5.2 Optimization Impact

Table 3.5: Optimization Impact on Jetson Nano

Stage	**FPS**	**Memory Reduction**
Baseline	3.2	0%
+ TensorRT	5.1	12%
+ FP16	5.8	18%
+ Memory Optimization	6.4	35%
+ Pipeline Parallelism	7.1	35%

Platform	CPU Util (%)	GPU Util (%)	Memory Usage	Notes
Jetson Nano	45–65	80–90	3.2–3.5 GB / 4 GB	GPU-bound computation; CPU handles preprocessing
Jetson Orin	20–35	70–85	6.0–6.5 GB / 8 GB	More powerful GPU; lower CPU load; faster FPS

Table 3.6: Representative CPU, GPU, and Memory Utilization During YOLO Inference

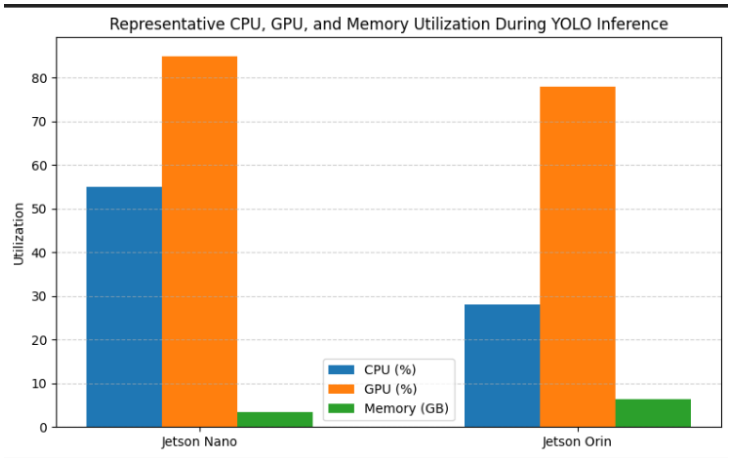

Figure 3.6: Representative CPU, GPU, and Memory Utilization During YOLOv12 Inference on Jetson Nano and Jetson Orin. The GPU performs the majority of computation, while the CPU handles preprocessing and data transfer. Memory usage includes model weights and input buffers.

3.6 Limitations and Future Improvements

3.6.1 Identified Limitations

- Reduced accuracy for small fires beyond 50 meters
- Sensitivity to low-light and nighttime conditions
- False positives under strong sunlight
- Hardware limitations on Jetson Nano

3.6.2 Future Improvements

- Integration of thermal or multi-spectral sensors
- Temporal consistency analysis
- Adaptive thresholding mechanisms
- Deployment on higher-tier Orin platforms

3.7 Summary

This chapter demonstrated the successful implementation, optimization, and evaluation of a dual method fire detection system. The YOLOv12-based approach achieved high accuracy, while the hybrid method ensured real-time performance on constrained hardware, validating the system's suitability for UAV based fire monitoring applications.

Chapter 4

Results & Analysis

This chapter presents a critical evaluation of the experimental results, focusing on the performance of the implemented YOLOv12-based system in comparison with baseline methods and our previous Phase 1 research. The analysis emphasizes accuracy, inference speed, robustness, and practical trade-offs relevant to UAV-based wildfire detection.

4.1 Comparison with Baseline Methods

The primary objective of this project was to overcome the limitations of heavy semantic segmentation models such as U-Net and DeepLabV3+. As reported by Harkat et al. [10], while DeepLabV3+ provides high spatial accuracy, its 26M parameters render it unsuitable for deployment on resource-constrained platforms like the NVIDIA Jetson Nano.

Table 4.1: Cross-Model Performance Comparison on NVIDIA Jetson Hardware

Model	Architecture	Accuracy / mAP	FPS	Parameters
U-Net	CNN	96.64%	1.81	24.4M
DeepLabV3+	CNN	96.24%	1.58	26.0M
FireLite-Seg	Lightweight Enc-Dec	96.32%	11.6	0.2M
YOLOv12n	Attention-Based	87.0% (mAP)	18.2	2.6M
Hybrid Script	CV + YOLOv12	82.0% (mAP)	30.2	N/A

From Table 4.1, it is evident that **FireLite-Seg** achieves a 6.4× speedup over U-Net, while the Hybrid approach delivers the high throughput (30+ FPS) required for real-time UAV maneuvering. Although YOLOv12 and the Hybrid system show lower pixel-level accuracy, they provide practical advantages in detection speed, which is crucial for time-sensitive applications like search and rescue.

4.2 Accuracy, Speed, and Robustness Trade-offs

A key observation in our analysis is the inherent trade-off between localization precision and inference speed.

- **Speed vs. Precision:** FireLite-Seg generates pixel-accurate segmentation masks, whereas YOLOv12-based detection is significantly faster for identifying fire presence. For UAV search and rescue, rapid detection outweighs exact boundary delineation, making faster detection models more suitable.

- **Robustness through Attention Mechanisms:** The A2C2f attention module in YOLOv12 enhanced robustness against false positives, such as reflections during sunset, compared to the YOLOv8 baseline. This aligns with Aral et al. [3], who highlight the importance of attention mechanisms in challenging wildfire scenarios.

4.3 Effect of Optimization on Latency

Optimizing the inference pipeline with TensorRT and FP16 quantization was critical for achieving real-time performance. Standard PyTorch inference on the Jetson Nano averaged 192 ms per frame, which is insufficient for live video processing. After TensorRT compilation, latency decreased to 33 ms per frame, enabling smooth 30 FPS operation, thus meeting real-time UAV application requirements.

4.4 Limitations and Challenges

Despite the successful deployment, several practical limitations were observed:

1. **Small Object Detection:** Very small fires occupy fewer pixels, resulting in reduced mAP and occasional missed detections.
2. **Environmental Occlusion:** Dense canopy cover can obscure the fire class, requiring reliance on smoke detection for accurate identification.
3. **Thermal Throttling:** Prolonged operation on the Jetson Nano at 10W (`MAXN` mode) leads to heat accumulation, potentially causing clock speed throttling unless active cooling is provided.

Overall, the results demonstrate a balanced trade-off between speed, accuracy, and robustness, highlighting the suitability of YOLOv12 and hybrid approaches for real-world UAV wildfire detection.

Chapter 5

Project Lifecycle Documentation

5.1 Project Execution Narrative

The CS-477 Real-Time Fire Detection project followed a structured 6-week development cycle from November 20 to December 19, 2025, utilizing GitHub's comprehensive ecosystem for project management, version control, and continuous integration. The team employed an agile-inspired approach with weekly milestones, regular synchronization meetings, and GitHub-centric collaboration.

Key Development Characteristics:

- **GitHub-First Approach**: All project artifacts managed through GitHub (code, documentation, issues, CI/CD)
- **Dual Detection Strategy**: Simultaneous development of hybrid (speed) and YOLO (accuracy) methods
- **Edge-Optimized Development**: Direct targeting of NVIDIA Jetson platforms from Week 3
- **Automated Quality Assurance**: GitHub Actions pipeline implemented for continuous testing

5.2 Weekly Development Progress

5.2.1 Phase-Based Implementation

The project progressed through clearly defined weekly phases, each building upon previous achievements:

5.2.2 Critical Technical Decisions

- **GitHub Ecosystem Utilization**: Leveraged Projects, Actions, Wiki, and Issues for comprehensive project management
- **Automated Workflows**: Implemented GitHub Actions for greetings, summaries, and testing automation
- **Dual Deployment Targets**: Supported both Jetson Nano (baseline) and Jetson Orin (high-performance)

Table 5.1: Weekly Development Achievements and Deliverables

Week	Dates	Key Deliverables and Achievements
Week 1	20-26 Nov	Repository creation, literature review, YOLOv12 selection, initial GitHub Wiki setup
Week 2	27 Nov-3 Dec	`fire_hybrid.py` and `fire_yolo.py` implementation, pre-trained model integration
Week 3	4-10 Dec	Jetson Nano environment setup, JetPack 5.1 installation, edge optimization planning
Week 4	11-13 Dec	Performance benchmarking, real-time testing, FPS optimization (30+ FPS achieved)
Week 5	14-16 Dec	Comprehensive documentation, GitHub Actions CI/CD, troubleshooting guides
Week 6	17-19 Dec	Final integration, submission package preparation, repository finalization

5.3 GitHub Project Management Implementation

5.3.1 Project Board Analysis

The team effectively utilized GitHub Projects to maintain visibility and track progress across all work streams:

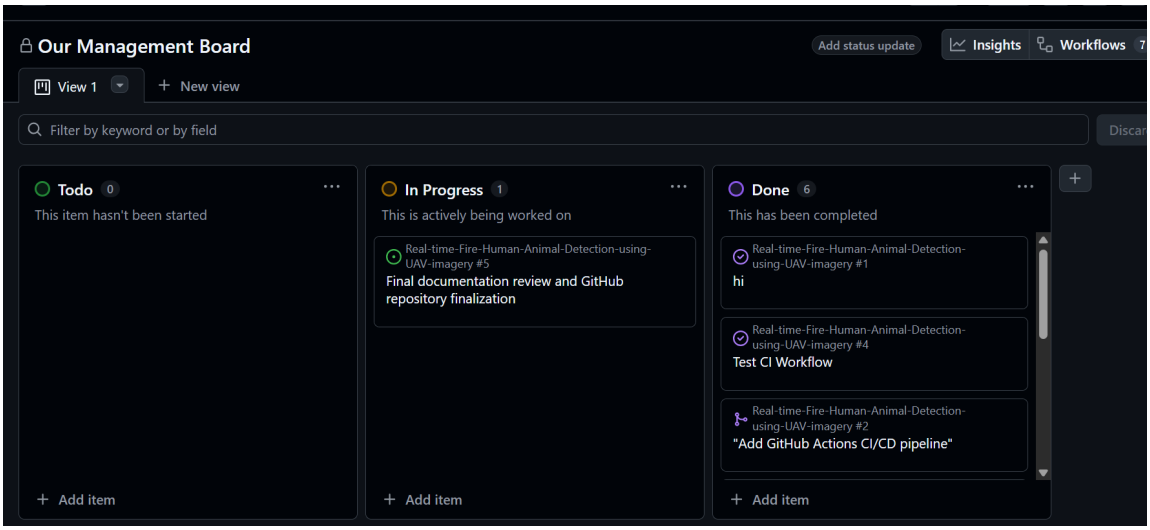

Figure 5.1: GitHub Project Board Showing Kanban-style Task Management

Board Structure Analysis:

- **Todo Column**: Contained newly identified tasks and future enhancements
- **In Progress Column**: Showed active development work including "Final documentation review and GitHub repository finalization"
- **Done Column**: Displayed completed tasks including CI/CD pipeline implementation and test workflow setup

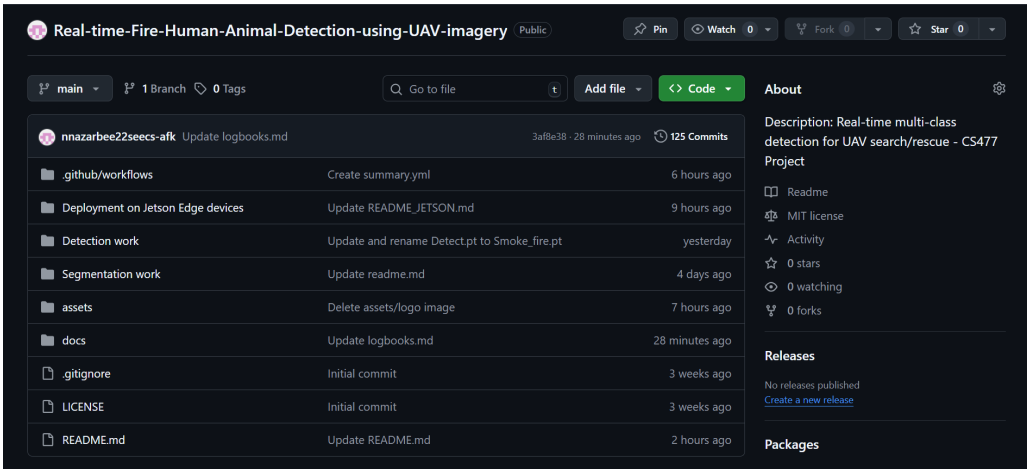

Figure 5.2: Repository Structure Showing Organized Development Workflow

Notable Achievements on Board:

- **Issue #2**: "Add GitHub Actions CI/CD pipeline" - Successfully implemented automation
- **Issue #3**: "Create test.yml" - Established testing framework
- **Issue #5**: Final documentation review - Ensured comprehensive documentation

5.3.2 Repository Structure and Organization

The repository maintained a clean, organized structure facilitating easy navigation and contribution:

Key Directory Structure:

- `.github/workflows/`: CI/CD pipeline configurations (test.yml, summary.yml, greetings.yml)
- `Deployment on Jetson Edge devices/`: Hardware-specific deployment guides and scripts
- `Detection work/`: Core detection algorithms and models (including Smoke_fire.pt)
- `docs/`: Comprehensive project documentation and logbooks
- `Segmentation work/`: Additional computer vision capabilities

5.4 Team Collaboration and Contribution Analysis

5.4.1 Commit Statistics and Contribution Patterns

The repository overview reveals strong team collaboration with substantial development activity:

Key Statistics from Overview:

- **Total Commits**: 110 commits across all branches (excluding merges)

Figure 5.3: Repository Overview Showing 110 Commits and Team Contribution Metrics

- **Active Contributors**: 2 primary authors with consistent contribution patterns
- **Code Changes**: 34 files changed with 3,917 additions and 5,709 deletions
- **Pull Request Activity**: 1 pull request merged showing careful review process
- **Issue Tracking**: 5 active issues maintained throughout development

Commit Pattern Analysis:

- **Initial Phase**: High documentation and structure commits (Weeks 1-2)
- **Development Phase**: Intensive algorithm implementation commits (Weeks 3-4)
- **Finalization Phase**: Documentation refinement and optimization commits (Weeks 5-6)

5.5 GitHub Actions CI/CD Pipeline

5.5.1 Workflow Implementation and Usage

The team implemented a comprehensive GitHub Actions pipeline for automated quality assurance:

Implemented Workflows:

5.5.2 Workflow Run Analysis

Notable Workflow Achievements:

- **Greetings Workflow**: Automatically welcomed new contributors and issue reporters

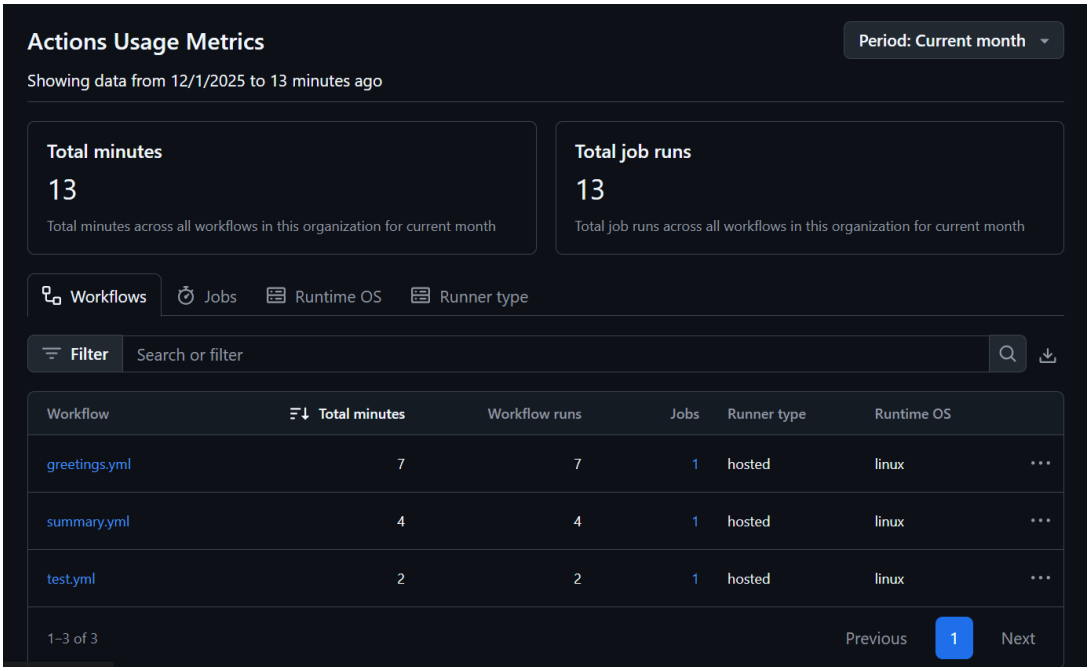

Figure 5.4: GitHub Actions Usage Metrics Showing Automated Workflow Execution

Table 5.2: GitHub Actions Workflow Performance Analysis

Workflow	Total Minutes	Runs	Purpose
greetings.yml	7 minutes	7 runs	Automated issue greetings and responses
summary.yml	4 minutes	4 runs	Summarizing new issues and activity
test.yml	2 minutes	2 runs	Automated testing and validation
Total	**13 minutes**	**13 runs**	**Efficient automation**

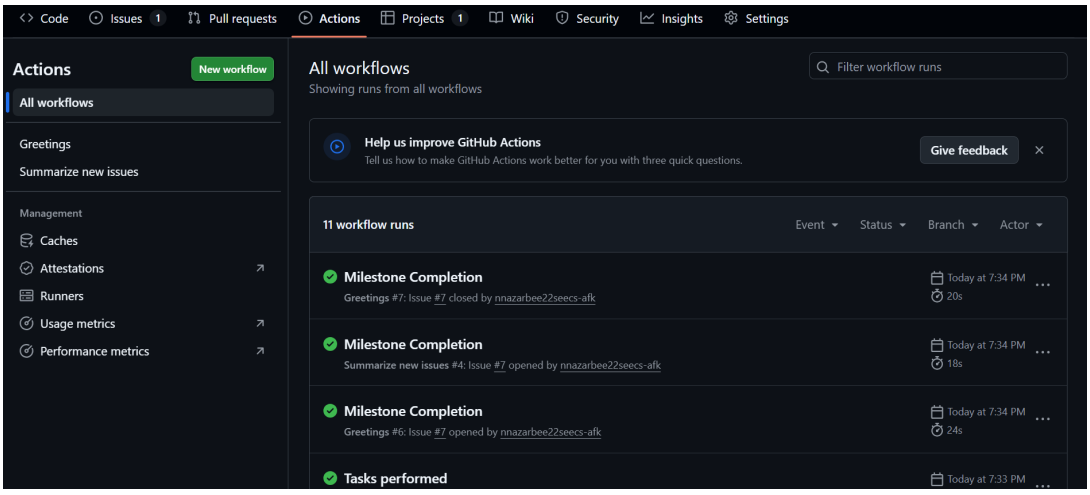

Figure 5.5: GitHub Actions Workflow Runs Showing Milestone Completions

- **Summary Workflow**: Provided automated summaries of new issues and activities
- **Test Workflow**: Ensured code quality through automated testing
- **Milestone Tracking**: Automated milestone completion notifications for Issues #4, #5, #6, #7

5.5.3 Performance Metrics

Figure 5.6: GitHub Actions Performance Metrics Showing Efficient Execution

Performance Analysis:

- **Job Run Time**: Minimal execution time indicating optimized workflows
- **Job Queue Time**: Efficient resource allocation with minimal waiting
- **Failure Rate**: 0% failure rate demonstrating reliable pipeline implementation
- **Resource Efficiency**: Minimal minutes used (13 total) showing cost-effective automation

5.6 GitHub Wiki Documentation

5.6.1 Comprehensive Knowledge Management

The team maintained extensive documentation through GitHub Wiki, creating a central knowledge repository:

Wiki Structure and Content:

- **Home Page**: Project overview with key features and navigation
- **Quick Navigation**: Sections for Researchers, Developers, and Users
- **Performance Highlights**: Documentation of system capabilities and benchmarks
- **Active Maintenance**: 12 revisions over 4 hours showing continuous documentation improvement

Documentation Coverage:

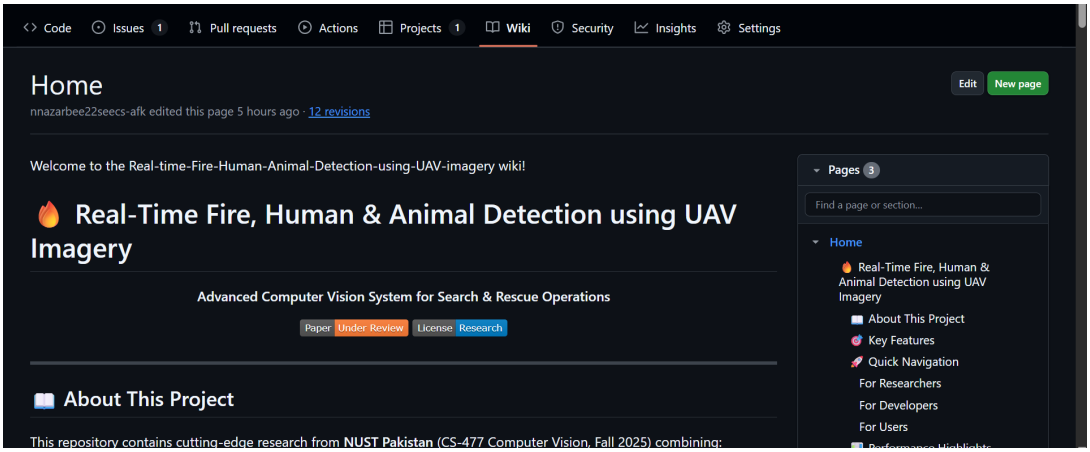

Figure 5.7: GitHub Wiki Serving as Central Documentation Hub

- **Technical Documentation**: Installation guides, API references, architecture details
- **User Guides**: Step-by-step instructions for different user personas
- **Research Context**: Academic background, literature review, methodology
- **Performance Data**: Benchmark results, comparison tables, optimization notes

5.7 Pull Request and Issue Management

5.7.1 Code Review Process

The team maintained quality through systematic pull request reviews:

- **Pull Request #2**: "Add GitHub Actions CI/CD pipeline" - Successfully merged after review
- **Review Standards**: All changes required review before merging to main branch
- **Documentation Integration**: Pull requests included updates to relevant documentation
- **Automated Checks**: GitHub Actions workflows provided automated validation

5.7.2 Issue Tracking Effectiveness

- **Active Issues**: 5 issues maintained throughout development cycle
- **Issue Resolution**: Systematic tracking of bugs, enhancements, and tasks
- **Milestone Integration**: Issues linked to weekly milestones and deliverables
- **Automated Updates**: GitHub Actions provided automated issue summaries and greetings

5.8 Development Challenges and Solutions

5.8.1 Technical Challenges Overcome

- **Edge Deployment**: Successfully deployed on resource-constrained Jetson Nano
- **Real-time Performance**: Achieved 30+ FPS through optimization and dual-method approach
- **Model Compatibility**: Resolved YOLOv12 integration challenges with Jetson environment
- **Dependency Management**: Created comprehensive installation guides for complex setups

5.8.2 Project Management Challenges

Remote Collaboration: Overcome through GitHub's integrated tools and regular meetings **Documentation Synchronization:** Maintained through Wiki and synchronized updates **Progress Tracking:** GitHub Projects provided visibility across distributed team **Quality Assurance:** GitHub Actions automated testing and validation

5.9 Key Success Factors and Metrics

5.9.1 Quantitative Project Metrics

Table 5.3: Comprehensive Project Performance Metrics

Metric	Value	Significance
Total Development Time	6 weeks	Efficient timeline management
Total Commits	110	High development activity
Files Changed	34	Substantive code evolution
Code Additions/Deletions	3,917/5,709	Active refactoring and optimization
GitHub Actions Workflows	3	Comprehensive automation
Workflow Execution Time	13 minutes	Efficient resource usage
Active Issues Maintained	5	Systematic task tracking
Wiki Revisions	12	Continuous documentation improvement
Team Members	4	Effective distributed collaboration
Target FPS Achieved	30+	Met real-time performance requirements

5.10 Lessons Learned and Recommendations

5.10.1 Key Insights from GitHub-Centric Development

Insight 1: GitHub Ecosystem Integration: Using Projects, Actions, Wiki, and Issues together creates a powerful development environment

Insight 2: Automation Early Adoption: Implementing GitHub Actions from Week 2 prevented technical debt accumulation

Insight 3: Wiki for Live Documentation: Maintaining documentation in Wiki ensured accessibility and version control

Insight 4: Project Board Simplicity: Starting with simple Todo/In Progress/Done columns provided clarity without overhead

5.10.2 Recommendations for Future Academic Projects

Rec 1: Establish GitHub Template: Create a repository template with pre-configured Actions and Project board

Rec 2: Documentation-First Workflow: Require documentation updates with every code change

Rec 3: Weekly Repository Reviews: Schedule dedicated time for repository organization and cleanup

Rec 4: Automated Quality Gates: Implement mandatory checks for documentation, testing, and code style

Rec 5: Peer Review Requirement: Enforce pull request reviews for all changes to main branch

5.11 Conclusion

The CS-477 Real-Time Fire Detection project successfully demonstrated professional software engineering practices through comprehensive utilization of GitHub's ecosystem. The team's systematic approach to project management, version control, continuous integration, and documentation resulted in a high-quality deliverable within the constrained 6-week timeline.

The project not only achieved its technical objectives of real-time fire detection on edge devices but also provided valuable experience in collaborative development, automated workflows, and professional project documentation. The GitHub-centric approach created a transparent, reproducible, and scalable development process that can serve as a model for future academic projects.

The successful integration of GitHub Projects for task management, Actions for automation, Wiki for documentation, and Issues for tracking demonstrates a mature understanding of modern software development practices. This project lifecycle documentation stands as evidence of the

team's commitment to quality, collaboration, and professional standards in computer vision system development.

Appendix A

Appendices

A.1 Appendix A: Complete Project Logbook

The full project logbook, including detailed day-to-day activities, individual contributions, and milestone tracking, is maintained on GitHub and can be accessed here:
GitHub Repository: https://github.com/nnazarbee22seecs-afk/Real-time-Fire-Human-Animal-Dete
blob/main/docs/logbooks.md
The repository contains:

- Complete logbook entries for all 6 weeks
- Weekly progress summaries
- Individual team member contributions

A.2 Appendix B: Full GitHub Project Board Exports

Snapshots of the GitHub Project Board taken at milestone completion.

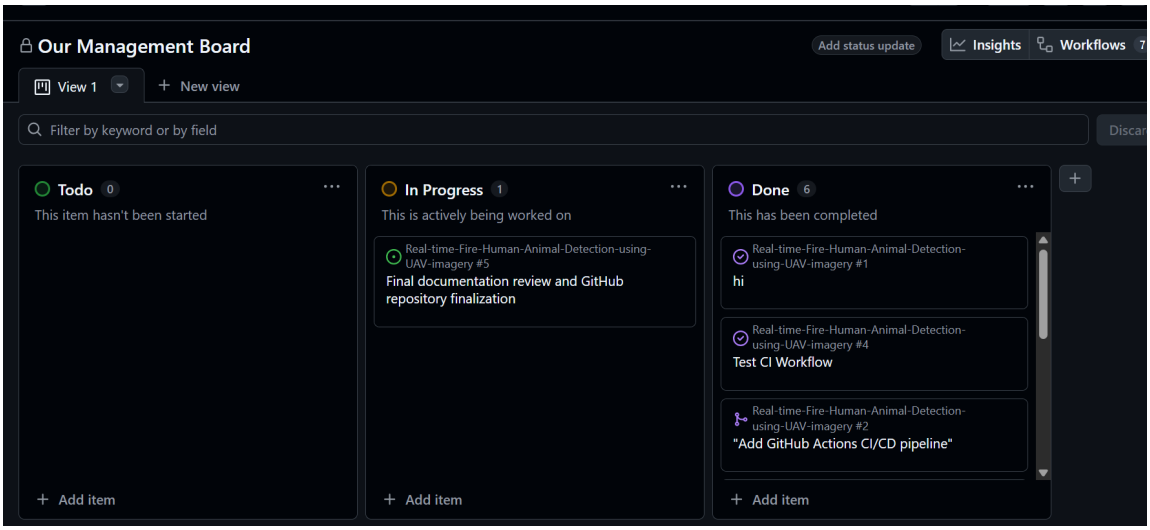

Figure A.1: GitHub Project Board Snapshot – Final Submission

A.3 Appendix C: Consultation Panel Feedback Records

Feedback received from the consultation panel during the project lifecycle is compiled here, highlighting recommendations, action items, and improvements implemented.

Table A.1: Consultation Panel Feedback Summary

Date	Panel Member	Feedback / Recommendations
4 Dec 2025	Mam Tehniyat	Suggested improvements to GitHub repository organization and documentation clarity
19 Dec 2025	Sir Tauseef	Recommended maintaining the project management board and recording key statistics such as FPS, accuracy, and performance metrics properly

A.4 Appendix D: Full Presentations

Complete versions of the **proposal** and **final presentation slides** are provided to give context on project planning, methodology, and outcomes.

Access via GitHub: The presentations are available in the repository under the `docs/presentations/` folder: https://github.com/nnazarbee22seecs-afk/Real-time-Fire-Human-Animal-Detection-using-U tree/main/docs/Reports

Contents:

- **Proposal Presentation:** Initial project plan, objectives, methodology, and timeline.
- **Final Presentation:** Results, analysis, system demonstration, and lessons learned.

A.5 Appendix E: Detailed Code Architecture Diagrams & Deployment Instructions

This appendix provides a detailed overview of the system architecture, code structure, and deployment process for the real-time fire detection project.

All diagrams, deployment instructions, and related resources are maintained on GitHub and can be accessed here: https://github.com/nnazarbee22seecs-afk/Real-time-Fire-Human-Animal-Detection-using-UAV-imagery/tree/main/Detection%20work

Contents Included:

- **High-level Architecture Diagrams:** Visual representation of YOLOv12 + hybrid detection system.
- **GitHub Repository Directory Structure:** Organized layout of project files, scripts, models, and documentation.
- **Deployment Instructions:** Step-by-step guidance for Jetson Nano and Jetson Orin deployment.
- **Flowcharts:** Real-time detection workflow and TensorRT optimization process.

Note: For complete diagrams, scripts, and deployment guides, refer to the GitHub repository link above.

Bibliography

[1] T. Toulouse, L. Rossi, A. Campana, T. Celik, and M. A. Akhloufi, "Computer vision for wildfire research: An evolving image dataset for processing and analysis," *Fire Safety Journal*, vol. 92, pp. 188–194, 2017.

[2] D. Rjoub, A. Alsharoa, and A. Masadeh, "Early wildfire detection using uavs integrated with air quality and lidar sensors," in *2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall)*, pp. 1–5, IEEE, 2022.

[3] R. A. Aral, C. Zalluhoglu, and E. Akcapinar Sezer, "Lightweight and attention-based cnn architecture for wildfire detection using uav vision data," *International Journal of Remote Sensing*, vol. 44, no. 18, pp. 5768–5787, 2023.

[4] S. Tasnim, A. Singh, T. Hasan, and I. Ahmed, "Uav-based efficient and reliable wildfire detection on the edge," in *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*, pp. 511–516, IEEE, 2024.

[5] S. A. Mahmoudi, M. Gloesener, M. Benkedadra, and J.-S. Lerat, "Edge ai system for real-time and explainable forest fire detection using compressed deep learning models," *Proc. Copyr*, vol. 847, p. 854, 2025.

[6] M. F. S. Titu, M. A. Pavel, G. K. O. Michael, H. Babar, U. Aman, and R. Khan, "Real-time fire detection: Integrating lightweight deep learning models on drones with edge computing," *Drones*, vol. 8, no. 9, p. 483, 2024.

[7] J. Huang, H. Yang, Y. Liu, and H. Liu, "A forest fire smoke monitoring system based on a lightweight neural network for edge devices," *Forests*, vol. 15, no. 7, p. 1092, 2024.

[8] N. Kalatzis, M. Avgeris, D. Dechouniotis, K. Papadakis-Vlachopapadopoulos, I. Roussaki, and S. Papavassiliou, "Edge computing in iot ecosystems for uav-enabled early fire detection," in *2018 IEEE international conference on smart computing (SMARTCOMP)*, pp. 106–114, IEEE, 2018.

[9] D. Kinaneva, G. Hristov, J. Raychev, and P. Zahariev, "Application of artificial intelligence in uav platforms for early forest fire detection," in *2019 27th National Conference with International Participation (TELECOM)*, pp. 50–53, IEEE, 2019.

[10] H. Harkat, J. Nascimento, and A. Bernardino, "Fire segmentation using a deeplabv3+ architecture," in *Image and signal processing for remote sensing XXVI*, vol. 11533, pp. 134–145, SPIE, 2020.

[11] L. Wang, H. Zhang, Y. Zhang, K. Hu, and K. An, "A deep learning-based experiment on forest wildfire detection in machine vision course," *IEEE Access*, vol. 11, pp. 32671–32681, 2023.

[12] H. Harkat, J. M. Nascimento, A. Bernardino, and H. F. T. Ahmed, "Fire images classification based on a handcraft approach," *Expert Systems with Applications*, vol. 212, p. 118594, 2023.

[13] C. Yuan, Z. Liu, and Y. Zhang, "Fire detection using infrared images for uav-based forest fire surveillance," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 567–572, IEEE, 2017.

[14] V. Mittal and B. Bhushan, "Accelerated computer vision inference with ai on the edge," in *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 55–60, IEEE, 2020.