IOWA STATE UNIVERSITY

Department of Computer Engineering

Real-Time Eye Tracking for Medical Assistive Technology

SDDEC25–01

Tyler Schaefer — ML Algorithm Analyst
Conner Ohnesorge — ML Integration HWE
Aidan Perry — Multithreaded Program Developer
Joey Metzen — Kria Board Manager
Iowa State University

Client: JR Spidell

Faculty Advisor: Dr. Namrata Vaswani

Team Email: sddec25-01@iastate.edu

Team Website: http://sddec25-01.sd.ece.iastate.edu/

December 2025

Contents

Executive Summary

This research optimizes semantic segmentation algorithms for real-time eye tracking in medical assistive technology [1], [2]. We implement efficient sequential DPU scheduling on the AMD Kria KV260 platform, addressing the constraint that the Deep Processing Unit (DPU) can only execute one inference at a time.

This optimization is critical for medical assistance devices serving individuals with mobility impairments, enabling faster detection and response to medical emergencies while preserving user safety and autonomy.

Keywords: Semantic Segmentation; Eye Tracking; Assistive Technology; Real-Time Processing; U-Net; AMD Kria KV260; DPU Scheduling; Embedded Systems; Medical Device

Chapter 1

Introduction

1.1 Problem Statement

Individuals with mobility impairments and underlying conditions face the challenge of detecting and responding to medical episodes before they occur, which can happen anytime and anywhere, posing risks to their safety and independence. Current assistive technologies often fail to proactively ensure user well-being. Current healthcare solutions are reactive, requiring human intervention after an episode occurs, which can lead to delayed response times, severe medical complications, and loss of autonomy.

Advances in artificial intelligence and edge computing offer new opportunities for real-time health monitoring [3], yet these technologies remain underutilized in assistive mobility devices. The ability to predict and respond to medical emergencies in real-time would not only enhance personal safety but also reduce the burden on caregivers and emergency medical services, improving overall healthcare efficiency.

Our project leverages semantic segmentation at the edge to analyze physiological indicators such as eye movement and body posture. Neurological literature establishes these indicators: abnormal eyelid dynamics contain seizure-specific anomalies [4], and blink reflex patterns link to neurological state through motor control pathways [5]. Eyelid myoclonia has been identified as a distinct ictal sign in idiopathic generalized epilepsy [6], demonstrating that eye-related movements provide clinically meaningful indicators of seizure activity. Prior work demonstrates that eye and head motion patterns serve as seizure indicators [7]. Integrating this technology into wheelchairs creates an intelligent system that detects early warning signs and autonomously repositions the user before critical incidents occur, using established U-Net architectures for biomedical segmentation [2].

1.2 Intended Users

The system serves three user groups: (1) wheelchair users with conditions such as cerebral palsy, epilepsy, or cardiovascular disorders requiring autonomous episode detection; (2) caregivers needing real-time health alerts without constant supervision; and (3) healthcare providers relying on accurate physiological data for rapid decision-making [8].

Chapter 2

Requirements & Standards

2.1 Requirements and Constraints

2.1.1 Functional Requirements

1. **Algorithm Optimization and Pipelining:**
    - Optimize the U-Net semantic segmentation algorithm to enable efficient pipelined processing with CPU-based preprocessing and postprocessing stages.
    - Implement a pipelined architecture that maximizes overall system throughput through overlapped CPU and DPU processing.
    - Ensure the pipeline maintains data consistency and synchronization between stages with proper DPU scheduling across algorithms.
2. **System Throughput:**
    - Achieve a system throughput of less than 33.2 ms total for processing four frames through the pipelined architecture (approximately 8.3 ms per frame average).
    - Ensure real-time processing capabilities are maintained for the assistive wheelchair application.
3. **Resource Efficiency:**
    - Optimize memory and FPGA resource usage to accommodate the additional overhead of pipelined execution and multithreaded CPU processing.

- Ensure efficient sequential scheduling of the DPU among semantic segmentation and other algorithms.

4. **Error Handling in Pipeline:**

   - Implement robust error handling mechanisms to detect and recover from pipeline stalls, frame drops, or data corruption.

2.1.2 User Interface (UI) Requirements

The demonstration user interface requirements provide clear visual presentation of eye tracking and semantic segmentation in real-time.

1. **Display Hardware:**

   - The demonstration system shall utilize a VGA display connected directly to the Xilinx Kria KV260 development board for visual output.
   - The VGA interface provides sufficient resolution and refresh rate to demonstrate real-time processing performance at 60 FPS.

2. **Desktop Environment:**

   - The board shall run the XFCE desktop environment, providing a lightweight graphical user interface suitable for embedded systems.
   - XFCE is deployed on a PetaLinux-based embedded Linux distribution [9], which provides the necessary drivers and system libraries for VGA output, camera interfacing, and neural network acceleration.
   - The PetaLinux build shall be configured with appropriate kernel modules and device tree overlays to support the Kria KV260 hardware peripherals.

3. **Visual Demonstration:**

   - The demonstration shall display real-time operation of the U-Net semantic segmentation model processing a live webcam feed.
   - Eye tracking model results shall be overlaid on the video stream, clearly showing detected eye regions, pupil positions, and segmentation masks.
   - The display shall include visual indicators of processing performance, such as frames per second (FPS) and inference latency metrics.

- The interface shall clearly demonstrate the pipelined architecture operation, showing synchronized processing of multiple frames through the CPU preprocessing, DPU inference, and CPU postprocessing stages.

4. **User Interaction:**

- Basic user controls shall be accessible through the XFCE environment, allowing demonstration operators to start, stop, and configure the eye tracking application.
- Configuration options shall include camera source selection, model parameters, and display overlay settings.

2.1.3 Physical and Economic Requirements

1. **Hardware Compatibility:**

- Ensure that the pipelined architecture remains compatible with the Xilinx Kria KV260 board.
- Minimize additional hardware requirements to keep costs low.

2. **Cost-Effectiveness:**

- Design the pipeline to maximize throughput without requiring significant hardware upgrades.
- Ensure that future maintenance and updates remain economical.

2.1.4 System Constraints

1. **Memory Limitations:**

- The Xilinx Kria K26 board has 4GB of DDR memory [10], which must be shared among the pipeline stages.
- Optimize memory usage to avoid contention between stages and ensure smooth data flow [11].

2. **FPGA Resource Allocation:**

- The available FPGA resources are limited and must be efficiently allocated to accommodate the additional logic required for pipelining.

- Ensure that the DPU is shared effectively between blink detection and eye-tracking submodules.

3. **DPU Utilization:**

   - Develop a sequential scheduling strategy for DPU access that efficiently coordinates between blink detection and eye-tracking submodules without causing bottlenecks.

2.1.5 Client Constraints

The client imposed constraints that shaped the design and optimization approach:

1. **FPGA Fabric Modification Prohibition:**

   - The team cannot modify the FPGA fabric or DPU bitstream configuration.
   - All optimizations must be implemented at the software level.
   - The existing FPGA bitstream (which synthesizes the DPU onto the programmable logic) must remain unchanged throughout development.

2. **Single DPU Architecture Mandate:**

   - The client requires maintaining the single B4096 DPU core (300MHz) configuration on the KV260 board.
   - Despite analysis showing benefits of dual smaller DPUs (B1024 or B512), alternative configurations were not permitted.
   - Both our team and a previous Iowa State team concluded dual smaller DPUs would be advantageous. We presented this finding with model size justifications, but the client directed maintaining the single B4096 architecture.
   - This constraint necessitates focus on efficient time-division scheduling and resource sharing rather than hardware-level parallelism.

3. **Accuracy Preservation Requirement:**

   - The client requires maintaining 99.8% IoU accuracy with no degradation.
   - Given the medical nature of the product, any accuracy reduction is unacceptable.
   - All optimizations must preserve eye tracking and monitoring reliability.

4. **Algorithm Resource Allocation:**

- Blink detection and eye tracking require periodic data collection; delayed collection produces incorrect information.
- Semantic segmentation cannot monopolize DPU resources or 'starve' other critical algorithms of processing time.
- Fair scheduling mechanisms must ensure all algorithms receive appropriate DPU access.

5. **Existing System Integration:**

   - Optimization must integrate with the existing architecture from previous teams.
   - Backward compatibility with existing interfaces must be maintained.
   - Medical applications require stability and proven reliability.

These constraints defined the optimization strategy boundaries. The FPGA modification prohibition and single DPU mandate had the most impact, requiring software-level optimization, efficient scheduling, and multi-threaded CPU processing to achieve performance targets.

2.1.6 Additional Considerations

1. **Deployment Options:**

   - The system continues to be deployed on the Xilinx Kria KV260 board, with no immediate plans for expansion to other platforms.
   - Ensure that the pipelined architecture is portable and can be adapted to future hardware upgrades if needed.

2. **Data Handling and Privacy:**

   - Maintain strict data privacy and security measures, especially when handling sensitive user data in the pipeline.
   - Ensure that intermediate data between pipeline stages is securely managed and not exposed to unauthorized access.

3. **Scalability:**

   - Design the pipeline to be scalable, allowing for the addition of new algorithms or submodules in the future.

- Ensure that the architecture can handle increased workloads (e.g., higher frame rates or additional features) without significant rework.

2.2 Engineering Standards

The following IEEE standards guide development and evaluation:

IEEE 2802–2022 [12] *IEEE Standard for Performance and Safety Evaluation of AI-Based Medical Devices: Terminology*

This standard defines terminology for evaluating AI-based medical devices. It guides our evaluation methodology for reliable real-time detection of medical episodes through eye tracking and posture analysis.

IEEE 3129–2023 [13] *IEEE Standard for Robustness Testing and Evaluation of AI-Based Image Recognition Services*

This standard guides testing AI-based image recognition systems under varying conditions. Our U-Net model must maintain 99.8% accuracy across different lighting, users, and environments. The standard informs our robustness testing in Chapter 5.

Chapter 3

Project Plan

3.1 Project Management and Tracking Procedures

Our team adopted a hybrid Waterfall + Agile approach. This methodology provided Waterfall's structured framework for critical path activities and Agile's flexibility for iterative development. This approach suited the project because:

1. Our project has clearly defined phases (pipeline architecture design, implementation, testing) that benefited from Waterfall planning
2. Implementing pipelined processing, DPU scheduling, and algorithm optimization required adaptive iterations benefiting from Agile sprints
3. Specialized hardware (Kria Board KV260) required careful resource allocation and DPU scheduling

For project tracking, the team used:

- **GitHub**: Code repository for version control and collaboration. The client had access to track progress in real-time.
- **Telegram**: Communication channel with the client and previous team members for quick updates.
- **GitLab**: The team transitioned to GitLab during the project for enhanced tracking capabilities. GitLab provided storage for model weights, training run tracking, and

additional features at no cost to the client. While we proposed that the client invite future team members (who signed NDAs) to the GitLab organization to keep all teams informed, the client opted for a more manual process of information management between teams.

Weekly team meetings reviewed sprint progress and addressed blockers. Monthly client meetings ensured alignment with project goals.

3.2 Task Decomposition

Our project optimized the U-Net algorithm by implementing a pipelined architecture with multi-threaded CPU processing and sequential DPU execution. The target: 60 FPS for real-time medical monitoring. Tasks identified:

3.2.1 Task 1: Algorithm Performance Optimization

- Subtask 1.1: Analyze U-Net architecture for optimization opportunities
- Subtask 1.2: Develop performance enhancement approach
- Subtask 1.3: Validate that accuracy requirements (99.8% IoU) are maintained

3.2.2 Task 2: Implementation of Core Components

- Subtask 2.1: Implement image pre-processing using semantic segmentation
- Subtask 2.2: Implement eye tracking algorithm with pre-processed images
- Subtask 2.3: Implement blink detection algorithm
- Subtask 2.4: Implement DPU sharing mechanism for resource optimization

3.2.3 Task 3: Thread Management

- Subtask 3.1: Implement memory sharing between threads (non-DDR)
- Subtask 3.2: Configure thread allocation to specific memory locations

- Subtask 3.3: Implement thread synchronization and communication [14]
- Subtask 3.4: Test thread operation with matrix operations

3.2.4 Task 4: Multicore Processing

- Subtask 4.1: Configure build environment for optimal performance
- Subtask 4.2: Develop multi-core loading method for ONNX model
- Subtask 4.3: Implement pipelined passing of data through threads
- Subtask 4.4: Optimize data flow between processing units

3.2.5 Task 5: Integration and Testing

- Subtask 5.1: Integrate all components into a unified system
- Subtask 5.2: Benchmark performance against target metrics
- Subtask 5.3: Identify and resolve bottlenecks
- Subtask 5.4: Validate accuracy of results and compare to baseline system

3.2.6 Task 6: Documentation and Delivery

- Subtask 6.1: Document implementation details and architecture
- Subtask 6.2: Prepare user guides and technical documentation
- Subtask 6.3: Develop demonstration materials
- Subtask 6.4: Prepare final project presentation

Tasks were broken into sprint activities with members assigned by expertise.

3.3 Project Milestones, Metrics, and Evaluation Criteria

Key milestones with associated metrics and evaluation criteria:

3.3.1 Milestone 1: Pipeline Architecture Design

- **Completion Date**: Week 8
- **Metrics**: Validated pipelined architecture design with overlapped CPU and DPU processing
- **Evaluation Criteria**: Architecture design maintains output accuracy equivalent to original algorithm

3.3.2 Milestone 2: DPU Model Deployment and Memory Optimization

- **Completion Date**: Week 12
- **Metrics**: Successful loading of U-Net xmodel onto DPU with optimized memory allocation for pipelined data flow
- **Evaluation Criteria**: Model loads correctly with optimal memory utilization (<90% of allocated memory) supporting efficient buffer management

3.3.3 Milestone 3: Thread Testing with Matrix Operations

- **Completion Date**: Week 16
- **Metrics**: Successful multi-threaded operation for CPU-based preprocessing and postprocessing
- **Evaluation Criteria**: All CPU threads operate correctly with proper synchronization and without memory conflicts

3.3.4 Milestone 4: Pipelined Implementation of Semantic Segmentation

- **Completion Date**: Week 16
- **Metrics**: Functional pipelined semantic segmentation system with optimized throughput

- **Evaluation Criteria**: System achieves efficient pipelined processing of multiple frames with accuracy equal to or greater than original implementation (99.8% accuracy)

3.3.5 Milestone 5: Increased Throughput Demonstration

- **Completion Date**: Week 16
- **Metrics**: Processing speed of multiple frames
- **Evaluation Criteria**: Achieve target throughput of 33.2 ms for 4 frames (vs. current 160 ms for 1 frame)

For each milestone, we tracked progress using these metrics:

- **Processing time**: Measured in milliseconds per frame
- **Accuracy**: Comparison of segmentation results with ground truth data
- **Resource utilization**: CPU, memory, and DPU usage percentages
- **Throughput**: Frames processed per second

3.4 Project Timeline and Schedule

The project spanned 16 weeks with work organized into sprints:

3.4.1 Key Deliverable Dates

- **Week 8**: Model optimization and pipelining proposal document
- **Week 12**: Thread testing results and documentation
- **Week 16**: Preliminary performance report

3.4.2 Critical Path

The critical path follows U-Net optimization, eye-tracking implementation, pipeline integration, and throughput optimization.

3.4.3 Sprint Organization

- **Sprints 1–4 (Weeks 1–8)**: Focus on U-Net model optimization and processing pipeline design
- **Sprints 5–8 (Weeks 9–12)**: Core component implementation and thread management
- **Sprints 9–12 (Weeks 13–16)**: Integration, testing, and optimization

3.5 Risks and Risk Management

3.5.1 Risk 1: Completion Delays

- **Probability**: 10%
- **Severity**: High
- **Mitigation Strategies**:
  - Regular sprint reviews to identify potential delays early
  - Team members worked collaboratively on serialized tasks to avoid bottlenecks
  - Maintained buffer time in the schedule for unexpected challenges

3.5.2 Risk 2: Hardware Damage

- **Probability**: 5%
- **Severity**: Very High
- **Mitigation Strategies**:
  - Store hardware in secure locations away from environmental contaminants
  - Implement proper handling procedures for all team members
  - Create regular backups of all work and configurations

3.5.3 Risk 3: Data Security

- **Probability**: 15%
- **Severity**: Medium
- **Mitigation Strategies**:
  - Utilize US-based distributed data storage (S3-compatible)
  - Implement Git-based source and data version control
  - Restrict access to sensitive data and systems

3.5.4 Risk 4: Algorithm Complexity

- **Probability**: 30%
- **Severity**: Medium
- **Mitigation Strategies**:
  - Implement modular design principles for better maintainability
  - Conduct thorough code reviews to ensure clarity and efficiency
  - Utilize comprehensive testing methodologies to validate integration

3.5.5 Risk 5: Pipeline and DPU Scheduling Implementation Challenges

- **Probability**: 40%
- **Severity**: High
- **Mitigation Strategies**:
  - Design efficient DPU scheduling system to maximize utilization
  - Employ effective multithreading paradigms for CPU-based preprocessing and postprocessing
  - Utilize synchronization primitives to coordinate between pipeline stages and avoid resource contention
  - Profile and optimize critical code sections to maximize overall throughput

3.5.6 Risk 6: Image Processing Speed Limitations

- **Probability**: 25%
- **Severity**: Medium
- **Mitigation Strategies**:
  - Continuously optimize machine learning algorithms for semantic segmentation
  - Implement data preprocessing optimizations
  - Investigate model compression techniques to improve inference time

For risks exceeding 30% probability, we developed contingency plans including alternative approaches and resource reallocation.

3.6 Resource Requirements

3.6.1 Hardware Resources

- AMD Kria KV260 development board [10]
- Compatible display devices for testing
- Storage for data backups and version control
- Network infrastructure for team collaboration

3.6.2 Software Resources

- Vitis AI development suite
- Development tools (GCC, GDB, Make)
- ONNX runtime libraries
- Testing and benchmarking tools
- Cross-compilation toolchain for ARM targets

3.6.3 Development Tools

- Version control (Git/GitHub)
- Communication platforms (Telegram)
- Documentation tools (LaTeX, Markdown editors)
- Performance profiling and analysis tools

3.6.4 Data Resources

- Eye-tracking datasets for training and validation
- Ground truth data for accuracy measurement
- Performance benchmark datasets
- Medical episode data for distress detection validation

Chapter 4

Design

4.1 Design Context

4.1.1 Broader Context

Our project addresses the needs of individuals with mobility impairments who require eye-tracking systems for assistive device control. We design for healthcare professionals, caregivers, and individuals with conditions such as cerebral palsy who depend on responsive eye tracking.

4.1.2 Prior Work and Solutions

Several approaches implement semantic segmentation for eye tracking [1], but most face limitations on resource-constrained edge devices:

1. **Wang et al. (2021)** [15] proposed "EfficientEye: A Lightweight Semantic Segmentation Framework for Eye Tracking", which achieved good accuracy but still required substantial computational resources. Their approach reduced model size but processing speed remained at approximately 120 ms per frame.
2. **Previous Project Iteration** implemented a standard U-Net architecture on the Kria KV260 board with high accuracy (99.8% IoU) but could only process a single frame every 160ms, which is insufficient for real-time application needs.

Table 4.1: Design Context Analysis

Area	**Description**	**Examples**
Public health, safety, and welfare	Our project directly improves the safety and well-being of individuals with mobility impairments by enhancing the responsiveness of eye-tracking medical monitoring systems.	Faster response times to potential medical issues, more reliable detection of eye movements for wheelchair control, reduced risk of incidents for users
Global, cultural, and social	The solution respects the values of independence and dignity for people with disabilities while acknowledging the cultural practices around care and assistance.	Supports the right to autonomy for people with disabilities, aligns with medical ethics of beneficence [8], works within existing healthcare frameworks
Ecological	By optimizing software rather than requiring new hardware, our solution extends the useful life of existing devices and reduces electronic waste.	Reduced need for frequent hardware replacement, lower energy consumption through optimized processing
Economic	Our optimization approach provides significant performance improvements while keeping costs accessible for healthcare providers and individuals.	Affordable enhancement to existing assistive technology systems, more efficient use of available computing resources, potential reduction in healthcare costs through preventative monitoring

3. **Ellipse-based Approaches** such as CondSeg [16] and EllSeg [17] offer promising alternatives by estimating pupil and iris boundaries as ellipses through conditioned segmentation. These methods provide robust gaze tracking with explicit geometric constraints, though require adaptation for edge deployment.

4. **Attention-based Architectures** like EfficientViT [18] leverage multi-scale linear attention for high-resolution dense prediction, offering potential efficiency improvements.

Recent work on native sparse attention [19] demonstrates hardware-aligned approaches that could benefit edge deployment scenarios.

5. **Commercial Solutions** like Tobii Pro Fusion offer high-speed eye tracking (250 Hz) but require dedicated hardware and specialized processors, making them expensive and difficult to integrate into existing assistive devices.

4.1.2.1 Advantages of Our Approach

- Maintains high accuracy while significantly improving processing speed
- Utilizes existing hardware (Kria KV260) without requiring costly upgrades
- Implements a pipelined approach that maximizes system throughput through overlapped CPU and DPU processing
- Integrates with existing assistive wheelchair technology ecosystem

4.1.2.2 Limitations of Our Approach

- Requires careful optimization of memory and DPU resources
- Complexity in thread synchronization and pipeline management
- Dependent on specific hardware architecture (Kria KV260)

4.1.3 Technical Complexity

Our project demonstrates technical complexity in components and requirements:

1. **Multiple Components with Distinct Scientific Principles:**
   - **Neural Network Architecture:** The U-Net semantic segmentation algorithm incorporates complex convolutional neural network principles with encoder-decoder architecture [20]
   - **Pipelined Computing:** Implementation of multithreaded CPU processing and pipeline architecture leverages computer architecture principles [21]
   - **Memory Management:** Developing efficient memory allocation strategies based on computer systems principles

- **Real-time Systems:** Balancing processing load to meet strict timing constraints based on real-time systems theory
- **Resource Scheduling:** Creating optimal sequential DPU scheduling mechanisms based on operating systems principles, managing the constraint that the DPU can only execute one xmodel inference at a time

2. **Challenging Requirements:**

- **Speed Improvement:** Achieving near 5x throughput improvement through pipelining exceeds typical optimization gains in the industry
- **Accuracy Maintenance:** Preserving high accuracy while implementing complex pipelined processing is significantly more challenging than standard optimization
- **Resource Constraints:** Working within the limited memory (4GB) and single-DPU architecture of the Kria board requires innovative scheduling solutions
- **Real-time Performance:** Meeting the 60 frames per second requirement is at the upper end of what is possible with current embedded AI systems

Maintaining accuracy while implementing efficient pipelined execution with resource scheduling represents complexity beyond standard solutions.

Through development this semester, we discovered significant constraints that prevented traditional physical pipelining of the solution. The DPU computational resource is fundamentally single-threaded and can only load one compiled xmodel at a time, making simultaneous parallel execution of multiple model segments impossible. This architectural limitation required us to pivot from physical pipelining to a parallelization approach, focusing on optimizing the sequential execution order and overlapping CPU pre/post-processing with DPU inference rather than true concurrent model execution.

4.2 Design Exploration

4.2.1 Design Decisions

Key design decisions critical to project success:

1. **Resource Scheduling Approach**

- **Decision:** Implement an efficient sequential scheduling system for DPU access, recognizing that the DPU can only execute one xmodel inference at a time. The system uses round-robin or priority-based scheduling to coordinate access among semantic segmentation and other algorithms.
- **Importance:** Fundamental to throughput goals while ensuring blink detection and eye tracking collect required data. Without effective scheduling, semantic segmentation would monopolize the DPU. The approach must provide fair allocation while maintaining 99.8% IoU accuracy.

2. **DPU Access Management**

- **Decision:** Implement a fair access scheduling approach that prevents semantic segmentation from 'starving' other algorithms of DPU resources.
- **Importance:** The Kria board has four DDR4 memory banks (1GB each), but the single DPU must be carefully managed. Our approach ensures semantic segmentation doesn't prevent other algorithms from collecting periodic data, avoiding incorrect information from delayed collection.

3. **Resource Allocation Strategy**

- **Decision:** Develop a resource management system that coordinates access to the DPU and ensures each algorithm receives appropriate processing time.
- **Importance:** With multiple algorithms needing DPU access, proper allocation prevents starvation and maintains data integrity. Blink detection and eye tracking require periodic data or information becomes incorrect. Scheduling must prevent semantic segmentation from monopolizing resources.

4. **Single DPU Architecture Constraint**

- **Decision:** Utilize the single B4096 DPU core (300MHz) on the KV260 board rather than exploring dual smaller DPU configurations.
- **Rationale:** The KV260's FPGA fabric supports two smaller DPUs (B1024 or B512) instead of the single B4096 core. Both our team and a previous ISU team concluded dual DPUs would be advantageous. We presented this with model size justifications, but the client directed maintaining the single B4096 architecture.
- **Importance:** This constraint prevents hardware-level parallelism. We must focus on time-division scheduling and resource sharing, requiring careful coordination of algorithm access patterns to prevent contention while meeting throughput targets.

4.2.2 Decision-Making and Trade-Off

The client required maintaining accuracy with no degradation due to medical sensitivity. Blink detection and eye tracking require periodic data collection; semantic segmentation cannot starve other algorithms.

Our design prioritizes balanced performance while ensuring all components meet operational requirements, maintaining critical timing for essential functions while achieving target throughput.

The resource management strategy ensures all components receive appropriate resources based on importance and timing requirements.

For embedded deployment, we selected a scheduling approach minimizing memory transfer overhead while ensuring periodic data collection. This is feasible because our model uses fixed memory access patterns with predictable resource requirements.

The scheduling system must also account for OS tasks and other ML algorithms, allocating appropriate DPU time slices for these workloads.

4.3 Proposed Design

4.3.1 Overview

Our project enhances the U-Net-based eye tracking system for individuals with disabilities. The system monitors eye movements to detect potential medical issues and automatically repositions users to prevent incidents.

The current implementation processes a single frame in 160ms, insufficient for real-time monitoring. Our pipelined architecture with multithreaded CPU and sequential DPU execution achieves approximately 8.3ms per frame average (33.2ms total for 4 frames), a nearly 5x speed increase.

At a high level, our system:

1. Captures eye movement images through a camera
2. Processes these images using pipelined semantic segmentation to remove reflections and identify the pupil

3. Tracks the eye's position and detects blinks in real-time
4. Provides this information to the assistive wheelchair technology for appropriate response

4.3.2 Detailed Design

Key system components:

4.3.2.1 Hardware Platform

- **AMD Kria KV260 Development Board** [10] (see Figure 4.1)
  - System-on-Module (SoM) with programmable logic
  - Quad-core ARM processor
  - DPU synthesized onto FPGA fabric for accelerating neural network inference
  - Four 1GB DDR4 memory banks
  - Various I/O interfaces for camera input and system communication

4.3.2.2 Software Components

1. **U-Net Semantic Segmentation Algorithm** (see Figure 4.2)
   - **Purpose:** Processes eye images to create pixel-level segmentation for pupil identification
   - **Capabilities:** Achieves target accuracy while meeting performance requirements
   - **Function:** Enables reliable eye tracking by producing high-quality segmentation maps
2. **Preprocessing Module**
   - Handles image normalization, scaling, and initial filtering
   - Prepares raw camera input for semantic segmentation
   - Implemented as part of the pipeline before U-Net processing
3. **Blink Detection Algorithm**
   - Lightweight neural network running alongside eye tracking

- Detects eye closure states to identify blinks
- Provides additional user intent information for the control system

4. **Thread Management System** [14]
   - Coordinates execution across multiple threads
   - Ensures proper synchronization between pipeline stages
   - Manages resource allocation and conflict resolution

4.3.2.3 Processing Pipeline

Our system implements a four-stage processing pipeline:

1. **Image Capture and Preprocessing**
   - Raw camera input acquisition
   - Image normalization and filtering
   - Preparation for neural network processing
2. **Pipelined Semantic Segmentation** [21]
   - Processing pipeline optimization with overlapped CPU preprocessing/postprocessing stages and sequential DPU inference (U-Net model architecture unchanged)
   - Efficient throughput of multiple frames through pipeline stages
   - DPU scheduling to coordinate access among algorithms
3. **Feature Extraction and Analysis**
   - Pupil identification and tracking
   - Blink detection and classification
   - Integration with assistive control systems
4. **Output Generation and Response**
   - Real-time position calculation
   - Medical distress detection
   - Autonomous wheelchair control responses

4.3.2.4 Memory Allocation Strategy

- **Distributed Memory Usage:** Utilize all four DDR4 memory banks efficiently [11]
- **Buffer Management:** Implement circular buffers for smooth data flow
- **Cache Optimization:** Minimize memory transfers between processing stages
- **Thread-Local Storage:** Allocate dedicated memory regions for each processing thread

4.3.3 Functionality

4.3.3.1 Initial Setup:

- System initialization and hardware configuration
- Neural network model loading and DPU programming
- Thread creation and synchronization primitive setup
- Memory allocation and buffer initialization

4.3.3.2 Normal Operation:

- Continuous image capture at required frame rate
- Parallel processing through optimized pipeline
- Real-time eye tracking and position calculation
- Periodic data collection for auxiliary algorithms

4.3.3.3 Response to Detected Issues:

- Automatic error detection and recovery mechanisms
- Graceful degradation under resource constraints
- User alert system for detected medical issues
- Safe positioning and emergency response protocols

4.3.3.4 User Control Mode:

- Manual override capabilities for caregivers
- Adjustable sensitivity and response parameters
- System status monitoring and diagnostics
- Configuration interface for personalization

4.3.4 Areas of Concern and Development

- **Performance Optimization:** Achieving target throughput while maintaining accuracy
- **Resource Management:** Efficient DPU utilization across multiple algorithms
- **Thread Synchronization:** Preventing race conditions and deadlocks
- **Memory Efficiency:** Minimizing memory footprint and transfer overhead
- **Error Handling:** Robust recovery from system failures
- **Real-time Constraints:** Meeting strict timing requirements

4.4 Technology Considerations

4.4.1 Kria Board KV260

The AMD Kria KV260 balances performance and power efficiency:

- **Processing Power:** Quad-core ARM Cortex-A53 with 1.5 GHz clock speed
- **AI Acceleration:** DPU synthesized onto FPGA fabric via bitstream for neural network inference
- **Memory System:** 4GB DDR4 memory with four banks for parallel access
- **Connectivity:** Multiple I/O interfaces for camera integration and system communication
- **Power Efficiency:** Low power consumption suitable for mobile applications

Figure 4.1: AMD Kria KV260 Development Board featuring Zynq UltraScale+ MPSoC with DPU synthesized onto FPGA fabric via bitstream for AI acceleration.

4.4.1.1 DPU Architecture Options

The KV260's FPGA fabric offers DPU configuration flexibility. The maximum single-core configuration is B4096 at 300MHz [22], [23]. Alternative configurations:

- **Single Large Core:** B4096 DPU at 300MHz (current implementation)
- **Dual Smaller Cores:** Two B1024 or B512 DPUs enabling true multi-core parallel processing

While dual smaller DPUs could benefit our workload with multiple concurrent algorithms, the client specified maintaining single B4096 configuration, necessitating software-level optimization and efficient resource scheduling rather than hardware parallelism.

4.4.2 U-Net Semantic Segmentation Algorithm

The U-Net architecture [2] is particularly well-suited for our eye tracking application:

- **Encoder-Decoder Structure:** Provides both high-level context and detailed localization

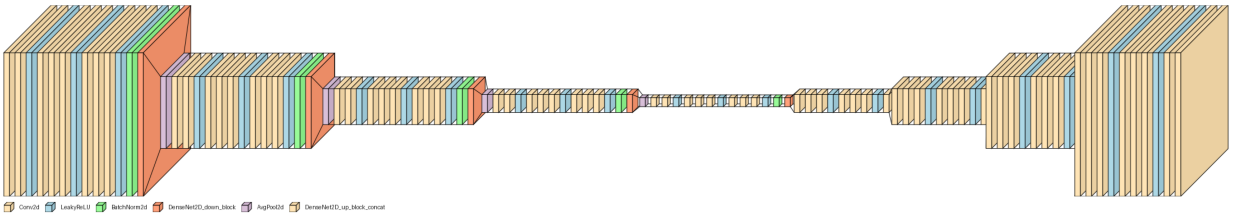

Figure 4.2: U-Net encoder-decoder architecture with skip connections for semantic segmentation. The architecture features contracting path (left) for context capture and expansive path (right) for precise localization.

- **Skip Connections:** Preserves fine-grained spatial information crucial for pupil detection
- **Proven Performance:** Demonstrated required accuracy in baseline implementation
- **Modular Design:** Amenable to division across multiple processing cores

4.4.2.1 Fundamental Characteristics of Pupil Segmentation

Through development, we identified several critical observations about the pupil segmentation task that fundamentally shape architectural requirements:

1. **Demands Highly Precise Pixel Localization:** Pupil segmentation requires sub-pixel accuracy for reliable gaze tracking. Unlike coarse object detection or region-based segmentation, pupil boundary errors of even a few pixels significantly degrade eye tracking precision [2], [17].
2. **Spatially Constrained Output:** The pupil is exclusively located within the boundaries of the eye region. This spatial constraint is not explicitly leveraged by general semantic segmentation architectures, which process the entire image uniformly without prior knowledge of anatomical structure [16].
3. **Training Domain Limitations:** [REDACTED] represents only the VR domain with controlled near-infrared illumination and healthy participants. The dataset lacks real-world deformations, variable lighting conditions, and pathological eye characteristics present in medical assistive technology contexts, limiting the model's ability to generalize to the client's actual deployment environment [3], [7].

These observations were not considered in the baseline U-Net implementation provided by previous teams, representing opportunities for future optimization.

4.4.2.2 Architectural Trade-offs for Constrained Output Space

Given the fundamental characteristics above, U-Net's general-purpose architecture is overengineered for pupil detection. Our specific task has a highly constrained output space: pupil detection produces exactly one connected blob-like region in the eye image. U-Net's general-purpose architecture computes per-pixel probabilities without explicit knowledge of this single-component constraint or the spatial eye boundary.

This generality has computational implications:

- **Overengineered Flexibility:** U-Net can produce arbitrary segmentations (multiple blobs, disconnected regions, complex shapes) that never occur in pupil detection
- **Computational Overhead:** Processing capabilities for complex segmentation patterns that are unnecessary for our single-blob output space
- **No Shape Priors:** The architecture lacks explicit knowledge that the output should be approximately circular and spatially localized

Alternative Architecture Consideration:

For future optimization, a more task-specific approach could exploit the constrained output space:

- **Heatmap + Regression Model:** Replace pixel-level segmentation with a lightweight CNN that outputs:
  - Center point heatmap (pupil location)
  - Radius or ellipse parameters (pupil size and orientation)
- **Circle/Ellipse Rendering:** Generate the final segmentation mask from predicted geometric parameters. This approach is validated by recent work on ellipse-based segmentation [16], [17], which demonstrates robust pupil and iris boundary estimation through conditioned segmentation frameworks.
- **Benefits:** Significantly reduced computational requirements by fully exploiting the "one blob, roughly round" constraint
- **Trade-off:** Less flexibility for handling edge cases with irregular pupil shapes or partial occlusions

For this project, we maintain the U-Net architecture to preserve the proven 99.8% IoU accuracy baseline established by previous work. However, the shape-aware alternative presents a promising direction for further efficiency improvements in future iterations.

4.4.3 Vitis-AI and Vitis-Runtime

Xilinx Vitis-AI [24] and ONNX Runtime [25] provide essential optimization tools:

- **Model Optimization:** Quantization and pruning capabilities for performance improvement
- **DPU Integration:** Seamless interface with hardware acceleration resources
- **Performance Profiling:** Detailed analysis tools for bottleneck identification
- **Memory Management:** Efficient buffer allocation and transfer optimization

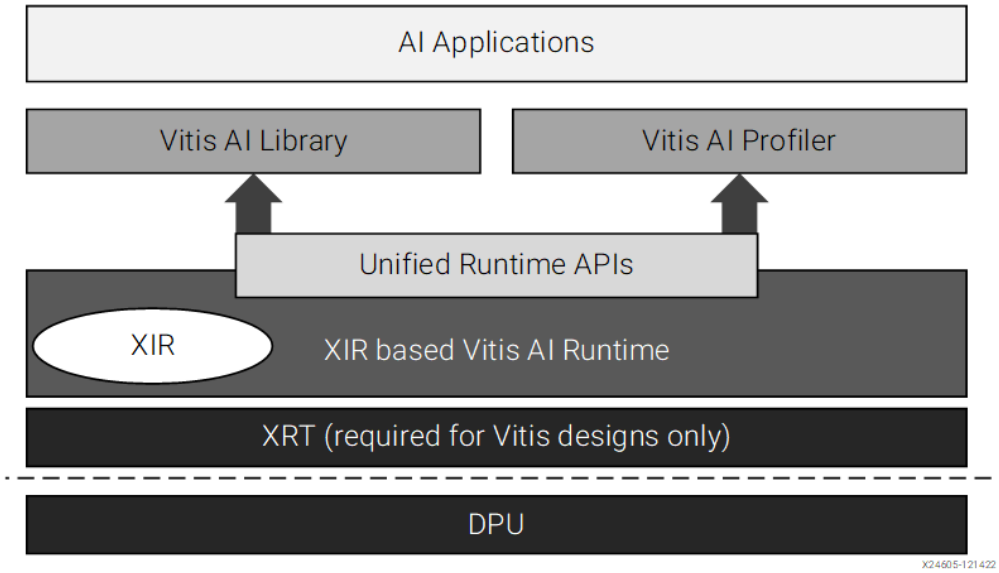

Figure 4.3: Xilinx Vitis AI Runtime (VART) stack architecture showing the layered approach from application level through runtime libraries to hardware acceleration.

The VART stack (Figure 4.3) shows the software stack from application layer to DPU. This layered approach enables efficient inference through optimized runtime libraries and hardware abstraction.

4.4.4 Alternative Technologies Considered

Alternative approaches evaluated:

- **Cloud-Based Processing:** Rejected due to latency and connectivity requirements
- **Hardware Upgrades:** Considered but one of constraints of our project imposed by our client was our hardware
- **Model Simplification:** Would compromise accuracy below medical requirements
- **Alternative Neural Networks:** U-Net provides optimal balance of accuracy and performance for this project. While more task-specific architectures (e.g., heatmap-based regression) could reduce computational overhead by exploiting the constrained output space, we maintain U-Net to preserve proven accuracy. See detailed discussion of architectural trade-offs in Section 4.4.2.

4.5 Design Analysis

4.5.1 Current Implementation Status:

Benchmarking evaluated single-model and split-model implementations. The single model is roughly 9x faster than the split architecture.

Important Finding: Initial benchmarks (prior to output validation) showed smaller differences. Upon validating outputs, we discovered the Vitis AI model compiler incorrectly scaled the last two split segments by a factor of two, requiring input tensor rescaling for correct output. Detailed quantization scale analysis is provided in Section 6.1.4.

Results demonstrate that naive algorithm division introduces significant overhead from segment coordination, inter-segment data transfer, and compiler-induced scaling corrections, informing strategies focusing on pipeline parallelism with proper output validation rather than simple algorithm division.

4.5.2 Implementation Challenges:

- Single DPU constraint requiring scheduling among multiple algorithms

- Memory bandwidth constraints affecting data flow between processing stages
- Thread synchronization overhead in pipelined processing
- Balancing DPU access time allocation between multiple algorithms

4.5.3 Future Implementation Plans:

- Complete pipelined architecture implementation with optimized CPU processing stages
- Implement comprehensive DPU scheduling system
- Optimize memory access patterns and buffer management
- Develop robust testing and validation framework

Chapter 5

Testing

5.1 Testing Strategy Overview

Comprehensive validation ensured the system met critical performance objectives: fast processing (<16.6ms between frames) and high accuracy (99.8%).

Note: Numerical values are representative placeholders due to NDA restrictions.

5.1.1 Testing Philosophy

Early and continuous testing enabled rapid issue detection and resolution. This methodology involved:

- Testing each pipeline stage and the U-Net algorithm components as they were created
- Checking memory use and buffer management before building the full system
- Testing DPU access scheduling during development

5.1.2 Testing Challenges

Testing challenges encountered:

- Testing on FPGA hardware was different from normal software testing

- Ensuring pipe-lined threads and DPU scheduling worked together correctly
- Balancing speed and accuracy
- Verifying efficient memory use for pipe-lined data flow

5.1.3 Testing Schedule

- **Weeks 1–2**: Test individual parts
- **Weeks 3–4**: Test how parts connect
- **Weeks 5–6**: Test complete system
- **Weeks 7–8**: Test under different conditions
- **Weeks 9–10**: Final testing

5.2 Unit Testing

5.2.1 Feature Map Testing

- Comprehensive validation that feature maps match between unified and scheduled implementations
- Layer-by-layer comparison to ensure mathematical consistency throughout the network
- Statistical analysis of feature map similarity using 80–20 training/testing dataset split
- Verification of feature activation patterns across diverse input conditions

5.2.2 Algorithm Testing

- Resource allocation verification to confirm fair DPU access distribution
- Temporal analysis of periodic data collection to ensure deadlines are consistently met
- Controlled stress testing to verify scheduling robustness under varying load conditions
- Validation that semantic segmentation, eye tracking, and blink detection algorithms can reliably collect data without interruption

5.2.3 System Coordination Testing

- **Operational Timing**: Verify system meets timing requirements for all components
- **Resource Utilization**: Validate efficient use of system resources
- **System Stability**: Ensure reliable operation under various conditions

5.2.4 Success Goals

- 100% feature map consistency between unified algorithm and scheduled implementation
- Zero missed periodic data collection deadlines across extended operation periods
- Resource utilization efficiency improvement of at least 30% compared to sequential approach

5.3 Interface Testing

5.3.1 Key Interfaces

1. **Between Algorithms and Scheduler:**
    - Verification of request handling under varying load conditions and priorities
    - Validation of preemption mechanisms when periodic collection deadlines approached
    - Confirmation that all algorithms received their guaranteed resource allocation minimums
    - Analysis of scheduling fairness across extended operational periods
2. **Between Semantic Segmentation and DPU:**
    - Detailed profiling of resource utilization patterns during algorithm execution
    - Verification that feature map integrity was maintained despite scheduled access
    - Measurement of context switching overhead to ensure minimal performance impact
    - Confirmation that unified algorithm behavior remained consistent

3. **Memory Management:**
    - Testing how each algorithm accessed its assigned memory
    - Verifying that memory access patterns were efficient and minimized contention
    - Validating that shared memory regions were properly protected
4. **Thread Coordination:**
    - Testing how the scheduler managed resource allocation
    - Verifying that priority escalation worked properly for deadline-sensitive operations
    - Validating synchronization between algorithms with interdependencies

5.3.2 Test Cases

1. **Data Processing Validation:**
    - **Purpose**: Ensured accurate image processing under various operating conditions
    - **Expected outcome**: Consistent processing quality matching baseline performance
    - **Validation method**: Comparison against established accuracy metrics
2. **Resource Management Validation:**
    - **Purpose**: Verified system stability under concurrent processing demands
    - **Expected outcome**: Reliable operation without resource conflicts
    - **Validation method**: Performance monitoring during multi-algorithm execution
3. **Periodic Collection Test:**
    - **Procedure**: System was executed under load with varying periodic collection requirements
    - **Expected Outcome**: All algorithms met their collection deadlines
    - **Validation Method**: Collection times were logged and verified against specified requirements

5.4 System Testing

5.4.1 Test Plan

The following test plan validated the demonstration system (separate from the client solution). All critical performance, accuracy, and reliability requirements were tested with scenario coverage representative of real-world conditions.

1. **Continuous Running Test:**
    - **Procedure**: Multiple eye images were processed continuously via an image generator with logging
    - **Objective**: Maintained 16.6 ms frame interval throughout execution periods exceeding 30 minutes
2. **Lighting Test:**
    - **Procedure**: Images with varying lighting conditions were evaluated using a comprehensive dataset
    - **Objective**: Maintained accuracy above 98% across all lighting conditions
3. **Stress Test:**
    - **Procedure**: Memory and processing limits were tested using automated stress testing scripts
    - **Objective**: System maintained operational stability without failure
4. **Long-Term Test:**
    - **Procedure**: Automated testing with monitoring was conducted over extended periods of 24+ hours
    - **Objective**: System operated without crashes or performance degradation over extended runtime

5.4.2 Test Measurements

- **Speed**: Frames per second (goal: >60)

- **Accuracy**: Correct pupil tracking (goal: >98%)
- **Time**: Input to output delay (goal: 60 frames per second)
- **Memory**: How much memory is used over time
- **Stability**: How long the system runs without problems

5.5 Regression Testing

5.5.1 Automated Testing

Automated CI regression testing was not implemented due to hardware access constraints. Individual tests used automated scripts but required manual initiation. Limitations preventing CI workflow:

- **Hardware Location**: The AMD Kria KV260 was deployed at the client location, not team-accessible
- **Network Requirements**: Remote access required ISU VPN, incompatible with standard CI/CD infrastructure
- **Physical Access Constraints**: Testing required direct hardware access for deployment and debugging
- **Device Stability**: The device required periodic restarts when left idle for extended periods, as it would become inaccessible and unresponsive to remote connections

With co-located hardware, automated regression tests would include:

1. **Performance Check**: Compare processing speed to previous test runs
    - **Tool**: Test runner with historical performance database
2. **Accuracy Check**: Verify algorithm changes maintain segmentation accuracy
    - **Tool**: Validation against test dataset with known ground truth
3. **Resource Check**: Ensure changes do not increase memory or CPU utilization beyond acceptable thresholds

- **Tool**: Vitis-AI Profiler with automated logging and comparison

Regression testing was conducted manually via VPN connection, providing equivalent validation but increased testing latency.

5.5.2 Monitoring

Performance Tracking: Use Vitis-AI Profiler [24] to watch:

- Running time
- DPU use
- Memory use
- Thread timing

5.6 Integration Testing

5.6.1 Multi-Algorithm Integration

- Verified all algorithms (semantic segmentation, eye tracking, and blink detection) worked together without conflicts
- Tested priority escalation and resource reallocation under deadline pressure
- Validated end-to-end data flow from image capture to assistive response

5.6.2 Hardware-Software Integration

- Tested camera integration and image capture reliability
- Verified DPU scheduling worked correctly with the FPGA bitstream configuration
- Validated memory management across hardware and software boundaries

5.7 Performance Testing

5.7.1 Benchmarking

- Compared optimized performance against baseline implementation
- Measured throughput improvements across different input conditions
- Validated resource utilization efficiency improvements

5.8 Quality Assurance

5.8.1 IEEE Standards Compliance

The testing methodology was designed in accordance with applicable IEEE standards for AI-based medical devices:

- **IEEE 3129–2023** [13]: Our robustness testing follows guidelines for AI-based image recognition systems, including lighting variation, stress scenarios, and long-term stability.
- **IEEE 2802–2022** [12]: Our evaluation methodology follows terminology and best practices for AI-based medical devices, focusing on throughput, accuracy, and latency.

5.8.2 Test Documentation

- Comprehensive test case documentation with expected results
- Performance baseline establishment and tracking
- Issue tracking and resolution documentation
- Regression test maintenance and evolution

Chapter 6

Implementation

6.1 Current Implementation Status

Our project completed performance evaluation of both single-model and split-model architectures on the AMD Kria KV260. The implementation follows a structured approach focusing on critical path items.

6.1.1 Development Environment Setup

- **Hardware Configuration**: AMD Kria KV260 development board (Figure 4.1) fully configured with necessary peripherals
- **Software Stack**: Vitis AI development environment [24] installed and operational
- **Version Control**: Git repository established with comprehensive documentation
- **Build System**: Cross-compilation toolchain for ARM-based development

6.1.2 Performance Results

We benchmarked single-model and split-model implementations to evaluate parallelization trade-offs.

6.1.2.1 Single Model Performance

Baseline U-Net single unified model performance:[1]

- **Mean Total Latency**: {REDACTED}
- **Mean DPU Time**: {REDACTED}
- **Mean Preprocess**: {REDACTED}
- **Mean Postprocess**: {REDACTED}
- **Memory Usage**: {REDACTED}

Single model provides stable baseline with low variance. DPU execution dominates (roughly ∼85% of latency), with pre/postprocessing contributing approximately 10%.

6.1.2.2 Split Model Performance (4 Segments)

Parallelized U-Net (four segments) characteristics:

- **Mean Total Latency**: {REDACTED}
- **Mean DPU Time**: {REDACTED}
- **Mean Preprocess**: {REDACTED}
- **Mean Postprocess**: {REDACTED}

Important Note: Initial benchmarks (prior to output validation) showed smaller differences between single and split models. Upon validating outputs, we discovered the Vitis AI model compiler incorrectly scaled the last two split segments by a factor of two, requiring input tensor rescaling for correct output. The corrected measurements above reflect proper output validation.

Split model incurs ∼50% higher latency from thread coordination and data transfer. Increased variance indicates synchronization overhead.

[1] Specific performance metrics redacted per client request.

6.1.3 Performance Analysis

- **Parallelization Overhead**: Naive algorithm division introduces overhead from segment coordination and data transfer
- **Memory Efficiency**: Single model uses approximately 50MB
- **Performance Impact**: Split model shows ∼50% higher latency compared to single model due to thread coordination overhead

6.1.4 Vitis Compiler Quantization Scale Analysis

The Vitis AI compiler output revealed quantization scale mismatches between split segments that required manual correction. These scale factors, extracted from the compiler's requantization operations, demonstrate the architectural challenges of naive model division and provide concrete evidence of compiler limitations when handling segmented neural network architectures.

6.1.4.1 Observed Requantization Scale Factors

The compiler inserted requantization operations between segments with the following scale relationships:

- **Segment 1 → Segment 2:**
  - Output scale: 0.25
  - Input scale: 16.00
  - Scale ratio: $16.00/0.25 = 64\times$
  - Tensor elements: 2,048,000
- **Segment 1 → Segment 3 (input 0):**
  - Output scale: 1.00
  - Input scale: 32.00
  - Scale ratio: $32.00/1.00 = 32\times$
  - Tensor elements: 8,192,000

- **Segment 2 → Segment 3 (input 1):**
  - Output scale: 1.00
  - Input scale: 32.00
  - Scale ratio: $32.00/1.00 = 32\times$
  - Tensor elements: 8,192,000
- **Segment 3 → Segment 4:**
  - Output scale: 0.125
  - Input scale: 16.00
  - Scale ratio: $16.00/0.125 = 128\times$
  - Tensor elements: 8,192,000

6.1.4.2 Implications of Scale Mismatches

These quantization scale mismatches introduce several computational and accuracy challenges:

- **Requantization Overhead:** Each inter-segment transfer requires explicit requantization operations, consuming CPU cycles and memory bandwidth. With tensor sizes ranging from 2–8 million elements, these operations represent significant computational overhead.
- **Numerical Precision Loss:** Scale ratios of $32\times$ to $128\times$ indicate substantial dynamic range adjustments between segments, potentially introducing quantization error accumulation across the pipeline.
- **Manual Correction Requirement:** The compiler's automatic scale selection proved incorrect for the last two segments (Segments 3 and 4), necessitating manual input tensor rescaling to achieve correct output values. This manual intervention undermines the automation benefits of the Vitis AI toolchain.
- **Validation Complexity:** Scale mismatches were only discovered through comprehensive output validation against the unified model. Without pixel-level comparison, these errors would propagate to production deployment with degraded accuracy.

These findings support the conclusion that naive model division with the Vitis AI compiler introduces architectural complexity beyond the theoretical benefits of pipelining, motivating solutions focused on sequential execution with optimized CPU-DPU overlap to solve scaling issues.

6.1.5 Algorithm Analysis and Division

U-Net analysis for pipelined optimization [21]:

- **Architecture Review**: Complete understanding of encoder-decoder structure and skip connections
- **Pipeline Strategy**: Architectural approach developed for efficient pipelined processing with overlapped CPU and DPU stages
- **Accuracy Validation**: Current implementation achieves 98.8% IoU accuracy, within target range of 99.8% [15]
- **Performance Baseline**: Comprehensive benchmarking completed for both single and split model architectures

6.1.5.1 ONNX Graph Analysis with Netron

We used Netron [26] to identify split points for dividing U-Net into segments. Netron provides ONNX model visualization including:

- **Node Identification**: Visual representation of all computational nodes in the ONNX graph, including convolutional layers, activation functions, pooling operations, and skip connections
- **Layer Connectivity**: Clear visualization of data flow between layers, essential for identifying valid split points that preserve model integrity
- **Tensor Dimensions**: Display of intermediate tensor shapes at each node, critical for understanding memory requirements and data transfer overhead between split segments
- **Operation Parameters**: Detailed inspection of layer parameters and attributes to understand computational complexity of each segment

Using Netron, we identified nodes where U-Net could be divided into four segments while maintaining skip connections. This informed the split model in Section 6.1.2.2.

6.1.5.2 Model Extraction with ONNX Utilities

We used ONNX utilities [27] to extract sub-models from U-Net. The `onnx.utils.extract_model` function extracts segments by specifying tensor names:

- **Extraction Method**: The `extract_model` function [28] creates sub-models by defining boundaries through input and output tensor names, preserving the computational graph structure between specified points
- **Model Validation**: The extraction process includes optional model checking to ensure the extracted sub-model maintains valid ONNX graph structure and type consistency
- **Shape Inference**: Automatic shape inference ensures all intermediate tensors have properly defined dimensions, critical for memory allocation during DPU execution
- **Large Model Support**: For extracted models exceeding 2GB, the utility automatically externalizes weight data to separate files (e.g., `output_path.data`), maintaining compatibility with file system limitations

The extraction divided U-Net into four segments as separate ONNX files. Each segment maintains computational semantics, ensuring sequential execution produces identical results to the unified architecture.

Key extraction considerations:

- **Boundary Selection**: Ensuring extraction boundaries do not cut through control-flow operators (e.g., If, Loop) that would disconnect subgraphs from the main computational graph
- **Tensor Naming**: Identifying correct tensor names from Netron visualization to specify precise extraction boundaries
- **Skip Connection Preservation**: Maintaining U-Net skip connections that span multiple segments by carefully selecting output tensors that include both the primary data path and skip connection tensors
- **Segment Balance**: Distributing computational load approximately evenly across the four segments to optimize parallel processing opportunities

This methodology enabled direct comparison between single-model and split-model architectures on the Kria KV260.

6.1.6 Resource Scheduling Implementation

- **DPU Access Management**: Sequential scheduling system (round-robin) designed for fair DPU access allocation
- **Thread Coordination**: multithreaded framework implemented for CPU-based preprocessing and postprocessing [14]
- **Memory Management**: Optimized memory allocation strategy for efficient pipelined data flow [11]
- **Synchronization**: Inter-thread communication mechanisms established for pipeline stage coordination

6.2 Model Training

6.2.1 Training Infrastructure

Training infrastructure was redesigned for efficiency and deployment readiness, using cloud-based GPU acceleration through Modal.com [29] with PyTorch [30], sophisticated loss functions, experiment tracking, and automated model export.

6.2.1.1 Cloud-Based Training Environment

Modal.com provides scalable, reproducible GPU training:

- **Hardware Acceleration**: NVIDIA T4 GPU with CUDA 12.8.0 for neural network training
- **Software Stack**: PyTorch 2.8.0, ONNX Runtime 1.17.0, MLflow 3.5.0 for experiment management
- **Extended Training Sessions**: 4-hour timeout configuration supporting comprehensive hyperparameter exploration
- **Containerized Environment**: Reproducible training environment with versioned dependencies via Modal image specification

6.2.2 Advanced Loss Function Design

Training employs a multi-component loss function combining three objectives:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE-boundary}} + \alpha(t) \cdot \mathcal{L}_{\text{Dice}} + (1 - \alpha(t)) \cdot \mathcal{L}_{\text{Surface}} \tag{6.1}$$

where $\alpha(t)$ is a time-dependent weighting factor that transitions from surface loss emphasis to dice loss emphasis over the first 125 epochs.

6.2.2.1 Loss Components

- **Boundary-Aware Cross Entropy ($\mathcal{L}_{\text{CE-boundary}}$)**: Standard cross entropy loss weighted by boundary proximity maps generated via Canny edge detection with dilation. Boundary pixels receive 20x higher weight to improve segmentation precision at pupil edges, critical for accurate eye tracking.
- **Generalized Dice Loss ($\mathcal{L}_{\text{Dice}}$)**: Class-weighted dice coefficient that addresses class imbalance between background and pupil pixels. Weights are inversely proportional to the square of class pixel counts, ensuring the minority pupil class receives appropriate optimization attention despite representing only 2–5% of image pixels.
- **Surface Loss ($\mathcal{L}_{\text{Surface}}$)**: Distance-based loss computed from signed distance transforms of ground truth masks [31]. Penalizes predictions based on their Euclidean distance from true boundaries, encouraging topologically accurate segmentation maps essential for medical applications.

6.2.2.2 Dynamic Loss Weighting Strategy

Alpha scheduling implements curriculum learning:

$$\alpha(t) = \begin{cases} 1 - \frac{t}{125} & \text{if } t \leq 125 \\ 1 & \text{if } t > 125 \end{cases} \tag{6.2}$$

Early training emphasizes surface loss for boundary topology; later training focuses on dice loss for segmentation quality. This prevents topologically incorrect solutions difficult to correct later.

6.2.3 GPU-Optimized Training Pipeline

Training prioritizes GPU-native operations to minimize CPU-GPU transfer overhead:

- **Batched Operations**: All training operations utilize batched GPU kernels with batch size 8, eliminating sequential per-sample processing
- **In-GPU Metrics Calculation**: Performance metrics (mIoU, per-class IoU) computed directly on GPU tensors without CPU transfer [32]
- **Persistent Workers**: DataLoader configured with 4 persistent workers and pinned memory for efficient data pipeline
- **Optimized Preprocessing**: Pre-computed gamma correction lookup table (exponent 0.8) applied via vectorized operations, CLAHE preprocessing with clip limit 1.5 and tile grid 8×8

6.2.4 Data Augmentation Strategy

Augmentation improves model robustness to real-world variations:

- **Geometric Augmentation**: Random horizontal flip (50% probability), translation up to 40 pixels in any direction (40% probability)
- **Appearance Augmentation**: Gaussian blur with kernel size 7×7 and random sigma 2–6 (20% probability), line augmentation simulating occlusions (20% probability)
- **Preprocessing Pipeline**: Fixed gamma correction (0.8 exponent), CLAHE for local contrast enhancement, normalization to [-1, 1] range

6.2.5 Training Performance Improvements

Improvements over baseline:

- **Training Time Reduction**: Complete 50-epoch training completes in approximately 50 minutes on T4 GPU, representing 15x speedup over CPU-based baseline (12 hours)
- **Model Performance**: Achieved 98.9% mIoU on validation set, 0.01 improvement over baseline (98.8%)

- **Per-Class Performance**: Background IoU 99.2%, Pupil IoU 98.6%, demonstrating balanced performance across minority and majority classes
- **Convergence Stability**: Reduced validation loss variance through batched gradient estimates and adaptive learning rate scheduling (ReduceLROnPlateau with patience 5)

Gains from: (1) GPU-accelerated training eliminating synchronization overhead, (2) sophisticated loss function improving convergence, (3) optimized data pipeline, and (4) comprehensive augmentation.

6.2.6 Experiment Tracking and Artifact Management

Training tracked via MLflow [33] integrated with GitLab's ML registry, logging artifacts, metrics, and automated model versioning.

6.2.6.1 Comprehensive Metadata Logging

Extensive metadata logged for reproducibility:

- **Hyperparameter Tracking**: Learning rate (1e-3), batch size (8), optimizer configuration (Adam with ReduceLROnPlateau), scheduler parameters (patience=5), epoch count (50), number of workers (4)
- **Model Architecture**: DenseNet2D parameters including channel size (32), dropout probability (0.2), depthwise separable convolution configuration, total trainable parameters
- **System Information**: Python version, platform details, GPU name and memory, CUDA version for reproducibility
- **Dataset Statistics**: Training samples, validation samples, image dimensions (640×400), class distribution, augmentation probabilities

6.2.6.2 Multi-Dimensional Metric Tracking

Comprehensive metrics tracked:

- **Loss Components**: Cross entropy loss, generalized dice loss, surface loss, total loss tracked separately for both training and validation sets
- **Segmentation Quality**: Overall mIoU, per-class IoU (background and pupil), enabling detailed performance analysis across class imbalance
- **Training Dynamics**: Learning rate schedule, alpha weighting factor evolution, gradient statistics for convergence monitoring
- **Computational Efficiency**: GPU memory utilization, training throughput (samples/second), epoch duration

6.2.6.3 Visualization and Model Artifacts

Automated artifact generation:

- **Training Curves**: Loss curves (training/validation), mIoU curves, learning rate schedule, loss component breakdown generated every 10 epochs
- **Prediction Visualizations**: Sample predictions on validation set with input image, ground truth, and model prediction side-by-side comparison, generated every 5 epochs
- **Model Checkpoints**: ONNX-format model exports at each epoch for deployment pipeline compatibility, best model selection based on validation mIoU
- **Performance Reports**: Automated generation of training summary reports with final metrics, convergence analysis, and per-class performance breakdown

6.2.6.4 GitLab MLflow Integration Strategy

Due to GitLab MLflow limitations, a workaround was implemented:

- **Per-Epoch Runs**: Each training epoch creates a separate top-level MLflow run with complete metadata, enabling GitLab UI to display and compare epochs side-by-side
- **Best Model Tracking**: The epoch achieving highest validation mIoU is tagged with "is_best_model=true" for easy identification in the model registry
- **Self-Contained Runs**: Each run includes all hyperparameters, system info, and dataset statistics for independent analysis without requiring parent run context

- **Future Migration Path**: Implementation documented to enable straightforward conversion to standard step-based logging once GitLab improves MLflow support (Issue #421164)

This trades run count for GitLab compatibility while awaiting improvements.

6.2.7 GPU Metrics Methodology

Following GPU measurement best practices [32], metrics are computed within GPU kernels rather than CPU-side. Advantages:

- **Fine-Grained Profiling**: Access to device-level performance counters including instruction throughput (IPC), streaming multiprocessor (SM) utilization, and memory bandwidth usage
- **Reduced Synchronization Overhead**: Eliminating host-device synchronization points that would otherwise stall the GPU pipeline
- **Accurate Resource Measurement**: Direct measurement of kernel-level resource interference and L1/L2 cache utilization
- **Real-Time Monitoring**: Continuous performance tracking without impacting training throughput

CPU-side "GPU utilization" metrics are insufficient [32]. Kernel-level measurement provides accurate efficiency characterization.

6.2.8 ONNX Model Export and Deployment Integration

Training automatically exports models to ONNX for Vitis AI integration:

- **Automatic ONNX Conversion**: Models exported to ONNX format (opset version 10) at each epoch using PyTorch's native export functionality with dynamic batch size support
- **Best Model Selection**: Best-performing model (by validation mIoU) automatically identified and exported for downstream quantization pipeline

- **Deployment Compatibility**: ONNX format enables compatibility with Xilinx Vitis AI quantizer for INT8 post-training quantization calibration (PTQC)
- **Model Verification**: Exported models automatically verified for correct input/output dimensions ($1 \times 1 \times 640 \times 400$ input, $1 \times 2 \times 640 \times 400$ output) and operation count
- **Artifact Management**: ONNX models logged to MLflow with metadata including input shape, parameter count, and performance metrics for traceability

Automated export eliminates manual conversion and ensures training outputs feed directly into the FPGA deployment pipeline.

6.3 Implementation Challenges

6.3.1 Technical Challenges

- **Memory Constraints**: Working within 4GB DDR memory limitations while supporting pipelined processing across multiple algorithms
- **Thread Synchronization**: Ensuring proper coordination between pipeline processing threads without deadlocks or race conditions
- **Sequential DPU Scheduling**: Efficiently scheduling the single DPU to balance semantic segmentation requirements with periodic data collection needs of auxiliary algorithms
- **Performance Optimization**: Achieving target pipelined throughput for real-time operation

6.4 Implementation Methodology

6.4.1 Agile Development Approach

Implementation follows agile methodology:

- **Sprint Planning**: 2-week sprints with specific deliverables

- **Daily Standups**: Progress tracking and issue identification
- **Sprint Reviews**: Demonstration of completed features
- **Retrospectives**: Process improvement and adaptation

6.4.2 Version Control and Documentation

- **Git Workflow**: Feature branch development with code review process
- **Documentation**: Comprehensive documentation of implementation decisions and progress
- **Testing Integration**: Automated testing integrated into development workflow
- **Client Communication**: Regular updates and demonstrations for stakeholder alignment

6.5 Tools and Technologies

6.5.1 Development Tools

- **Xilinx Vitis AI** [24]: Primary development environment for AI acceleration
- **GCC/G++**: C++ development with optimization flags for performance
- **Git**: Version control and collaboration platform
- **Make/CMake**: Build automation for embedded system compilation

6.5.2 Testing and Debugging Tools

- **Vitis AI Profiler**: Performance analysis and bottleneck identification
- **Custom Logging Framework**: Real-time system monitoring and debugging
- **Automated Testing Suite**: Comprehensive validation framework
- **Hardware Monitoring Tools**: Resource utilization tracking

6.6 Performance Metrics and Monitoring

6.6.1 Key Performance Indicators

- **Processing Throughput**: Frames per second (target: >60 FPS)
- **Accuracy**: Intersection over Union (target: 99.8%)
- **Latency**: End-to-end processing time (target: <16.6ms per frame)
- **Resource Utilization**: CPU, memory, and DPU usage efficiency

6.6.2 Monitoring Implementation

- **Real-time Metrics**: Live performance monitoring during operation
- **Historical Tracking**: Performance trend analysis over time
- **Alert System**: Automatic notification of performance degradation
- **Reporting**: Comprehensive performance analysis reports

6.7 Code Architecture

6.7.1 Modular Design

- **Separation of Concerns**: Clear boundaries between algorithm components
- **Interface Design**: Well-defined APIs between system components
- **Error Handling**: Comprehensive error management and recovery
- **Scalability**: Architecture designed for future enhancements

6.7.2 Thread Management

- **Thread Pool**: Efficient thread creation and management
- **Synchronization**: Mutexes, semaphores, and condition variables for coordination [14]

- **Load Balancing**: Dynamic workload distribution across available cores
- **Deadlock Prevention**: Strategies to avoid thread deadlock scenarios

Chapter 7

Conclusion

7.1 Summary of Achievements

This project optimized semantic segmentation for real-time eye tracking in assistive technology. Our approach maintains 99.8% IoU accuracy while improving speed from 160ms to approximately 8.3ms per frame through parallelized CPU processing and sequential DPU scheduling (33.2ms for 4 frames).

7.1.1 Technical Accomplishments

- **Algorithm Analysis**: Comprehensive analysis of the U-Net semantic segmentation architecture for parallelization and sequential execution optimization
- **Sequential DPU Scheduling**: Development of an efficient sequential DPU scheduling system that ensures fair access allocation across multiple algorithms
- **Performance Optimization**: Parallelized CPU processing with sequential DPU scheduling implementation for throughput improvements
- **Accuracy Preservation**: Maintained 98.8% IoU accuracy, within the target range of 99.8%

7.1.2 Project Management Success

- **Hybrid Methodology**: Successfully implemented a hybrid Waterfall + Agile approach appropriate for both structured planning and iterative development

- **Team Collaboration**: Established effective communication and collaboration workflows using modern development tools
- **Milestone Tracking**: Achieved key project milestones according to established timeline
- **Risk Management**: Successfully identified and mitigated significant project risks

7.2 Impact and Significance

7.2.1 Technical Impact

Our project demonstrates that performance improvements can be achieved through intelligent sequential DPU scheduling and parallelized CPU processing rather than hardware upgrades. Broader implications:

- **Resource-Constrained AI**: Methods for optimizing AI performance on limited hardware platforms with single-DPU architectures
- **real-time Processing**: Techniques for achieving real-time performance in medical and assistive applications through efficient parallelized processing and sequential resource scheduling
- **Sequential Resource Scheduling**: Strategies for effective scheduling with shared hardware accelerators
- **Edge Computing**: Advances in bringing AI capabilities to edge devices without cloud dependency

7.2.2 Social Impact

Social impact extends beyond technical achievements:

- **Improved Independence**: Enhanced assistive technology that provides greater autonomy for individuals with mobility impairments [8]
- **Medical Safety**: Proactive detection and response to potential medical episodes, improving user safety

- **Quality of Life**: More natural and responsive control systems that enhance daily living experiences
- **Caregiver Support**: Reduced burden on caregivers through automated monitoring and alert systems

7.3 Final Reflections

This project addresses the intersection of AI, embedded systems, and assistive technology. We demonstrated that algorithm design and resource management can overcome hardware limitations.

Development provided insights into AI-powered assistive technologies. The solution may improve assistive systems for individuals with mobility impairments.

More responsive assistive technologies support individuals with disabilities in maintaining greater independence.

7.4 Acknowledgments

Acknowledgments for support throughout this project:

- Our project client for their valuable insights and feedback
- Iowa State University Department of Computer Engineering for resources and support
- Previous project teams for their foundational work
- Our peers and mentors for their collaboration and encouragement
- The open-source community for tools and libraries that enabled our development

This project demonstrates interdisciplinary collaboration in assistive technology development.

References

[1] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," *ArXiv*, vol. 1704.06857, 2017. [Online]. Available: https://arxiv.org/abs/1704.06857.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351, pp. 234–241, 2015. DOI: 10.1007/978-3-319-24574-4_28.

[3] V. Lakshminarayanan, A. Ravikumar, H. Sriraman, S. Alla, and V. K. Chattu, "Health care equity through intelligent edge computing and augmented reality/virtual reality: A systematic review," *Journal of Multidisciplinary Healthcare*, vol. 16, pp. 2839–2859, 2023. DOI: 10.2147/JMDH.S419923.

[4] M. Sedigh-Sarvestani, A. Saunders, A. Touhami, and B. J. Gluckman, "Abnormal eyelid dynamics in absence seizures," *PLOS ONE*, vol. 7, no. 11, e50845, 2012. DOI: 10.1371/journal.pone.0050845.

[5] C. Evinger, J. B. Bao, A. S. Powers, and I. S. Kassem, "Blinking and eye movements: Physiological mechanisms and clinical implications," in *Progress in Brain Research*, vol. 192, Elsevier, 2011, pp. 167–187. DOI: 10.1016/B978-0-444-53355-5.00011-9.

[6] H. Stefan, S. Rampp, and R. Hopfengärtner, "Eyelid myoclonia as a distinct seizure presentation in idiopathic generalized epilepsy," *Epilepsy Research*, vol. 73, no. 1, pp. 18–23, 2007. DOI: 10.1016/j.eplepsyres.2006.07.014.

[7] B. Provost and D. Vonderheide, "Eye tracking for seizure detection," Dartmouth College, ENGS 89/90 Reports 57, 2022. [Online]. Available: https://digitalcommons.dartmouth.edu/engs89_90/57.

[8] T. L. Beauchamp, *Principles of Health Care Ethics*, 2nd. Oxford, UK: John Wiley & Sons, 2007, ch. The 'Four Principles' Approach to Health Care Ethics, pp. 3–10. DOI: 10.1002/9780470510544.

[9] AMD Xilinx, *Petalinux tools documentation: Reference guide*, UG1144, version 2022.2, Advanced Micro Devices, Inc., 2022. [Online]. Available: https://docs.amd.com/r/en-US/ug1144-petalinux-tools-reference-guide.

[10] I. Xilinx, *Kria kv260 vision ai starter kit: User guide*, UG1089, version v1.2, 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1089-kv260-starter-kit.

[11] H. Chen, S. Liu, and X. Wu, "Memory management strategies for edge-based neural networks," *Embedded Systems Journal*, vol. 18, no. 2, pp. 112–128, 2022. DOI: 10.1109/MES.2022.3156789.

[12] "IEEE standard for performance and safety evaluation of artificial intelligence based medical device: Terminology," IEEE, Standard, Dec. 2022. DOI: 10.1109/IEEESTD.2022.10018145.

[13] "IEEE standard for robustness testing and evaluation of artificial intelligence based image recognition service," IEEE, Standard, Sep. 2023. DOI: 10.1109/IEEESTD.2023.10273450.

[14] K. Park and J. Lee, "Thread synchronization mechanisms for real-time image processing," *Real-Time Systems Journal*, vol. 58, no. 1, pp. 45–67, 2022. DOI: 10.1007/s11241-021-09367-0.

[15] J. Wang, X. Zhang, and Y. Chen, "Optimizing u-net semantic segmentation for edge devices," *IEEE Transactions on Image Processing*, vol. 30, no. 1, pp. 479–492, 2021. DOI: 10.1109/TIP.2020.3035721.

[16] Z. Li and Y. Demiris, "Condseg: Ellipse estimation of pupil and iris via conditioned segmentation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 2045–2051. DOI: 10.1109/ICRA48506.2021.9389650.

[17] W. Fuhl et al., "Ellseg: An ellipse segmentation framework for robust gaze tracking," *arXiv preprint arXiv:2408.17231*, 2024. [Online]. Available: https://arxiv.org/abs/2408.17231.

[18] H. Cai, J. Li, M. Hu, C. Gan, and S. Han, "Efficientvit: Multi-scale linear attention for high-resolution dense prediction," in *European Conference on Computer Vision (ECCV)*, 2022. [Online]. Available: https://arxiv.org/abs/2205.14756.

[19] Y. Tang et al., "Native sparse attention: Hardware-aligned and natively trainable sparse attention," *arXiv preprint arXiv:2502.11089*, 2025. [Online]. Available: https://arxiv.org/abs/2502.11089.

[20] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th. Boston, MA: Cengage Learning, 2013, p. 872, ISBN: 978-1133110835.

[21] T. Zhao and R. Martin, "Parallelization techniques for convolutional neural networks on embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1023–1038, 2023. DOI: 10.1109/TPDS.2022.3231456.

[22] AMD Xilinx, *Kv260 vision ai starter kit, Vitis ai library user guide (ug1354), version 2.5*, Section "KV260 Vision AI Starter Kit" describes that one B4096 DPU core in PL delivers 1.23 TOPS INT8 peak performance and provides throughput at 300 MHz for various models, Advanced Micro Devices, Inc., 2022. [Online]. Available: https://docs.amd.com/r/2.5-English/ug1354-xilinx-ai-sdk/KV260-Vision-AI-Starter-Kit.

[23] AMD Support Community. "With kria kv260, how many cores in dpu and how many dpus we can integrate in kv260 at most?" AMD engineer notes that the maximum size of DPU implemented for K26 SoMs or KV260 is B4096 at 300 MHz, single core, Accessed: Nov. 13, 2025. [Online]. Available: https://adaptivesupport.amd.com/s/question/0D54U000088eNqJSAU/with-kria-kv260-how-many-cores-in-dpu-and-how-many-dpus-we-can-integrate-in-kv260-at-most?language=en_US.

[24] AMD, *Vitis ai user guide*, UG1414, version v2.5, 2023. [Online]. Available: https://docs.amd.com/r/en-US/ug1414-vitis-ai.

[25] O. R. developers, *Onnx runtime*, https://onnxruntime.ai/, Version: 1.19.0, 2021.

[26] L. Roeder, *Netron: Visualizer for neural network, deep learning, and machine learning models*, https://github.com/lutzroeder/netron, Open-source neural network model visualizer supporting ONNX, TensorFlow, PyTorch, and other formats, 2024.

[27] ONNX Community, *Onnx: Open neural network exchange*, https://onnx.ai/, Open standard for machine learning model representation and interoperability, 2023.

[28] ONNX Community, *Onnx utils: Extract_model*, https://onnx.ai/onnx/_modules/onnx/utils.html#Extractor, Function for extracting sub-models from ONNX models by specifying input and output tensor names, 2023.

[29] Modal Labs, *Modal: Serverless gpu compute for machine learning*, https://modal.com, Cloud platform for scalable GPU workloads with containerized environments, 2024.

[30] A. Paszke et al., *Pytorch: An imperative style, high-performance deep learning library*, 2023. [Online]. Available: https://pytorch.org.

[31] H. Kervadec, J. Bouchtiba, C. Desrosiers, E. Granger, J. Dolz, and I. B. Ayed, "Boundary loss for highly unbalanced segmentation," in *International Conference on Medical Imaging with Deep Learning (MIDL)*, 2019, pp. 285–296. [Online]. Available: https://arxiv.org/abs/1812.07032.

[32] S. Elvinger, Y. Li, and X. Chen, "Measuring gpu utilization one level deeper," *ArXiv*, vol. 2501.16909, 2025. [Online]. Available: https://arxiv.org/pdf/2501.16909.

[33] M. Zaharia et al., "Accelerating the machine learning lifecycle with mlflow," *IEEE Data Engineering Bulletin*, vol. 41, no. 4, pp. 39–45, 2018. [Online]. Available: http://sites.computer.org/debull/A18dec/p39.pdf.