IOWA STATE UNIVERSITY

Department of Computer Engineering

Senior Design Project - Computer Engineering

Real-Time Eye Tracking Through Optimized Semantic Segmentation for Medical Assistive Technology

Team SDDEC25-01: VisionAssist

[Student Names]
Department of Computer Engineering
Iowa State University

Ames, Iowa
Spring 2025

Contents

Executive Summary

This research presents an innovative approach to optimizing semantic segmentation algorithms for real-time eye tracking in medical assistive technology applications. The proposed system implements efficient resource scheduling to ensure fair access to the Deep Processing Unit (DPU) across multiple concurrent algorithms, eliminating the need for algorithm splitting that would otherwise increase computational overhead. Performance optimization targets a reduction from 160ms to 33.2ms per frame for simultaneous processing of 4 frames while maintaining 99.8% Intersection over Union (IoU) accuracy. This optimization is critical for medical assistance devices serving individuals with mobility impairments, particularly those with cerebral palsy and similar conditions.

The system leverages the AMD Kria KV260 development board's multi-core architecture and specialized DPU capabilities for neural network inference. Key technical innovations include deadline-aware resource scheduling, memory management optimization, and parallel processing strategies designed to maintain algorithmic accuracy while achieving 60 frames per second throughput. The architecture ensures that periodic data collection algorithms operate without interruption, preventing resource starvation that could compromise system reliability.

Initial development milestones include successful establishment of the development environment, validation of existing eye tracking algorithms, and implementation of a preliminary scheduling framework. Current performance metrics demonstrate 98.8% IoU accuracy, indicating feasibility of reaching the target specifications. Subsequent development phases focus on resource management system refinement and inter-component data flow optimization.

This research directly enhances quality of life for individuals with disabilities by enabling more responsive and reliable eye-tracking interfaces. The improved processing capabilities facilitate faster detection and response to potential medical emergencies, providing caregivers and users with enhanced safety and autonomy in daily living activities.

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

Handicapped individuals with underlying conditions face the critical challenge of detecting and responding to medical episodes before they occur, which can happen anytime and anywhere, posing significant risks to their safety and independence. In a broader societal context, individuals with disabilities often encounter inadequate assistive technologies that fail to proactively ensure their 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.

This issue is particularly significant as advancements in artificial intelligence and edge computing offer new opportunities for real-time health monitoring [1]. However, these technologies remain underutilized in the field of 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.

To address this problem, our project focuses on leveraging semantic segmentation at the edge to analyze physiological indicators such as eye movement and body posture. By integrating this technology into wheelchairs, we aim to create an intelligent system that detects early warning signs of medical distress and autonomously moves the user to a safer position before a critical incident occurs. This approach bridges the gap between existing assistive technologies and the urgent need for proactive, real-time health monitoring, using established U-Net architectures for biomedical image segmentation [2], ultimately empowering handicapped individuals to navigate their daily lives with greater security and independence.

1.2 Intended Users

1.2.1 Primary Clients

The primary clients of this product are individuals with mobility impairments, many of whom have underlying physiological conditions such as Cerebral Palsy, epilepsy, or cardiovascular disorders. These individuals depend on wheelchairs for mobility and face heightened risks associated with sudden medical episodes. They require a proactive safety system that detects early signs of medical distress and responds autonomously to relocate them to a safe position.

Maintaining independence is a critical priority for these individuals, as many wish to lead active lives without constant supervision. By integrating real-time monitoring and intervention features, this product empowers users by providing an added layer of security without compromising their autonomy. The benefits of such a system include a significant reduction in medical emergencies, increased confidence in navigating daily life, and an overall improved quality of life.

1.2.2 Caregivers and Family Members

Caregivers and family members form the secondary user group, as they play an essential role in ensuring the well-being of individuals with mobility impairments. Parents, guardians, and professional caregivers are often burdened with the responsibility of constant monitoring, which can be both emotionally and physically demanding. They need a reliable alert system that provides real-time updates on the user's condition, allowing them to respond appropriately without intrusive supervision.

This product alleviates some of the stress associated with caregiving by offering automated alerts and health tracking, enabling caregivers to provide support when necessary while also granting users greater independence. The ability to receive timely notifications about potential medical issues enhances caregivers' ability to act swiftly and effectively, fostering a more sustainable care model.

1.2.3 Healthcare Providers and Emergency Responders

The tertiary user group consists of healthcare providers and emergency responders, including medical professionals, therapists, and paramedics who are responsible for diagnosing,

treating, and responding to medical emergencies among mobility-impaired individuals. These professionals rely on accurate, real-time health data to assess risks and make informed decisions.

An automated system capable of detecting early warning signs of medical distress and transmitting alerts to healthcare providers can significantly improve response times and patient outcomes. Additionally, the ability to integrate this data with existing healthcare monitoring systems enhances the efficiency of medical intervention. By bridging the gap between assistive mobility technology and healthcare services, this project contributes to a more data-driven approach to patient care, ultimately improving medical decision-making and emergency response capabilities.

1.3 Summary

Each of these user groups plays a critical role in the success and impact of this project. By addressing the needs of handicapped individuals, caregivers, and healthcare professionals, this product aims to create a safer, more autonomous, and more efficient system for managing mobility and health-related challenges. The integration of real-time monitoring and autonomous intervention not only enhances the quality of life for individuals with disabilities but also eases the burden on caregivers and improves medical response strategies. In doing so, this project contributes to a broader movement toward proactive, technology-driven healthcare solutions that prioritize safety, independence, and well-being.

Chapter 2

Requirements & Standards

2.1 Requirements and Constraints

2.1.1 Functional Requirements

1. **Algorithm Splitting and Pipelining:**
    - Split the U-Net semantic segmentation algorithm into four equal parts to enable parallel processing across multiple cores.
    - Implement a pipelined architecture to allow concurrent execution of the split U-Net segments and other algorithms (e.g., image preprocessing, blink detection, eye tracking).
    - Ensure the pipeline maintains data consistency and synchronization between stages.
2. **System Throughput:**
    - Achieve a system throughput of less than 33.2 ms per frame when processing four frames concurrently.
    - 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 pipelining and parallel execution.

- Ensure efficient sharing of the DPU between the split U-Net segments 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

1. **Command Line Interface (CLI):**

   - Retain the existing user-friendly CLI for both technical and non-technical users.
   - Add new commands to allow users to:
     (a) Configure pipeline settings (e.g., number of threads, buffer sizes).
     (b) Monitor pipeline performance (e.g., throughput, latency, resource usage).
   - Include help commands to describe new pipeline-related functionalities.

2. **Command Feedback:**

   - Provide real-time feedback on pipeline performance, including throughput, latency, and error rates.
   - Display warnings or errors if the pipeline encounters issues (e.g., buffer overflow, frame drops).

3. **Error Handling and Logging:**

   - Enhance error logging to include pipeline-specific issues (e.g., stage delays, synchronization errors).
   - Provide detailed logs to assist users in debugging pipeline performance and resource allocation.

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, which must be shared among the pipeline stages.
   - Optimize memory usage to avoid contention between stages and ensure smooth data flow.

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 Deep Learning Processing Unit (DPU) is shared effectively between blink detection and eye-tracking submodules.

3. **DPU Utilization:**

   - Develop a scheduling strategy to allow the DPU to be shared between blink detection and eye-tracking submodules without causing bottlenecks.

2.1.5 Additional Considerations

1. **Deployment Options:**

   - The system will continue 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 are applicable to this project and guide the development and evaluation processes:

IEEE 2952-2023 *IEEE Standard for Secure Computing Based on Trusted Execution Environment*

Trusted Execution Environments (TEEs) are used to protect sensitive data and computations. This standard ensures that systems using TEEs follow security best practices, reducing the risk of unauthorized access or tampering.

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

This standard provides clear terms and definitions for evaluating the performance and safety of AI-based medical devices. It helps ensure these devices are reliable and effective in real-world medical settings.

IEEE 7002-2022 *IEEE Standard for Data Privacy Process*

This standard outlines best practices for protecting user data and ensuring privacy. It helps organizations comply with regulations and build trust with users when handling sensitive information.

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

This standard provides guidelines for testing AI-based image recognition systems to

ensure they work reliably under different conditions. It helps identify and fix issues that could arise from unexpected inputs or scenarios.

IEEE 3156-2023 *IEEE Standard for Requirements of Privacy-Preserving Computation Integrated Platforms*

Privacy-preserving computation allows data to be processed without exposing sensitive information. This standard defines the requirements for platforms that support such computations, ensuring they protect user privacy.

IEEE 2842-2021 *IEEE Recommended Practice for Secure Multi-Party Computation*

Secure multi-party computation lets multiple parties work together on shared data without revealing their individual inputs. This standard provides guidance for implementing these protocols, making collaborative computing safer for sensitive applications like healthcare and finance.

IEEE 1484.1-2003 *IEEE Standard for Learning Technology - Learning Technology Systems Architecture (LTSA)*

This standard defines a framework for designing and integrating educational software and systems. It ensures that learning technologies can work together seamlessly, supporting innovation in online education.

Chapter 3

Project Plan

3.1 Project Management and Tracking Procedures

Our team has adopted a hybrid Waterfall + Agile project management approach for this project. This methodology provides us with both the structured framework of Waterfall for critical path activities and the flexibility of Agile for iterative development and testing. This hybrid approach is particularly well-suited for this project because:

1. The semantic segmentation optimization has clearly defined phases (mathematical division, implementation, testing) that benefit from Waterfall planning
2. The technical nature of implementing parallelism and optimizing algorithms requires adaptive iterations that benefit from Agile sprints
3. Working with specialized hardware (Kria Board KV260) requires careful planning of resource allocation and access

For project tracking, the team will utilize the following tools:

- **GitHub**: Primary code repository for version control, documentation, and collaboration. Our client also has access to this repository to track progress in real-time.
- **Telegram**: Main communication channel with our client and previous years' team members for quick updates and questions.
- **Discord**: Team communication for internal discussions and virtual meetings.

Weekly team meetings will be held to review sprint progress, address blockers, and plan upcoming work. Monthly meetings with the client will ensure alignment with project goals and requirements.

3.2 Task Decomposition

Our project involves optimizing the semantic segmentation U-Net algorithm by implementing parallelism across multiple cores and the MPU. The key objective is to increase throughput from 160 ms per frame to 33.2ms across 4 frames. The following tasks and subtasks have been 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
- Subtask 3.4: Test thread operation with matrix operations

3.2.4 Task 4: Multicore Processing

- Subtask 4.1: Configure Docker environment for efficiency
- Subtask 4.2: Develop multi-core loading method for split 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

These tasks will be further broken down into sprint activities with specific team members assigned based on their expertise, as outlined in the personnel effort requirements section.

3.3 Project Milestones, Metrics, and Evaluation Criteria

The following key milestones have been identified for the project, along with their associated metrics and evaluation criteria:

3.3.1 Milestone 1: Mathematical Division of the Algorithm

- **Completion Date**: Week 8
- **Metrics**: Validated mathematical approach for dividing U-Net algorithm
- **Evaluation Criteria**: Division maintains output accuracy equivalent to original algorithm

3.3.2 Milestone 2: Loading of Split Algorithm Weights onto MPU

- **Completion Date**: Week 12
- **Metrics**: Successful loading of model segments into appropriate memory locations
- **Evaluation Criteria**: Each model segment loads correctly with optimal memory utilization (<90% of allocated memory)

3.3.3 Milestone 3: Thread Testing with Matrix Operations

- **Completion Date**: Week 16
- **Metrics**: Successful parallel operation of multiple threads
- **Evaluation Criteria**: All threads operate concurrently without memory conflicts

3.3.4 Milestone 4: Docker Environment Configuration

- **Completion Date**: Week 16
- **Metrics**: Streamlined processing environment
- **Evaluation Criteria**: Environment supports all required libraries and tools with minimal overhead

3.3.5 Milestone 5: Pipelined Implementation of Semantic Segmentation

- **Completion Date**: Week 16
- **Metrics**: Functional parallelized semantic segmentation algorithm
- **Evaluation Criteria**: Algorithm processes multiple frames concurrently with accuracy equal to or greater than original implementation (99.8% accuracy)

3.3.6 Milestone 6: Increased Throughput Demonstration

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

For each milestone, our team will track progress using the following quantifiable 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 will span approximately 16 weeks, with work organized into sprints. The project timeline follows a structured approach with major milestones and deliverable dates:

3.4.1 Key Deliverable Dates

- **Week 8**: Mathematical division proposal document
- **Week 12**: Thread testing results and documentation
- **Week 16**: Preliminary performance report

3.4.2 Critical Path

The critical path for this project follows the mathematical division of the algorithm, implementation of the eye-tracking components, integration of the parallelization framework, and final optimization of throughput.

3.4.3 Sprint Organization

- **Sprints 1-4 (Weeks 1-8)**: Focus on mathematical algorithm analysis and division
- **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 will work collaboratively on serialized tasks to avoid bottlenecks
  - Maintain 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: Parallelism Implementation Challenges

- **Probability**: 40%
- **Severity**: High
- **Mitigation Strategies**:
  - Employ effective parallel programming paradigms
  - Utilize synchronization primitives to avoid resource contention
  - Profile and optimize critical code sections to maximize performance

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 with probability exceeding 30%, our team will develop detailed contingency plans, including alternative implementation approaches and resource reallocation strategies.

3.6 Personnel Effort Requirements

3.7 Resource Requirements

3.7.1 Hardware Resources

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

3.7.2 Software Resources

- Vitis-AI development suite
- Docker runtime environment
- Development tools (GCC, GDB, etc.)
- ONNX runtime libraries
- Testing and benchmarking tools

Table 3.1: Personnel Effort Requirements by Task

Team Member	**Task Area**	**Specific Tasks**	**Hours**
Tyler	Mathematical Division	• Optimize and divide algorithm into 4 parts • Pipeline U-Net into 4 roughly equal parts while maintaining accuracy • Code implementation • Testing and validation	35
Aidan	Algorithm Implementation	• Integrate with codebase • Implement into current codebase with 4 pipelines • Thread management configuration • Implementation testing	30
Conner	OS and Environment	• Docker configuration optimization • ONNX splitting for MPU loading • OS scheduler optimization • Data version control system demonstration	30
Joey	Hardware Management	• Kria board benchmarking • Research and document hardware capabilities • Hardware performance optimization	20

3.7.3 Development Tools

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

3.7.4 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 Semantic Segmentation Optimization project is situated in the healthcare and assistive technology domain, specifically addressing the needs of individuals with mobility disabilities who require eye-tracking systems for communication and control of assistive devices. We are designing for healthcare professionals, caregivers, and most importantly, individuals with conditions such as cerebral palsy who depend on efficient and responsive eye tracking for daily activities and medical monitoring.

4.1.2 Prior Work and Solutions

Several approaches have been used to implement semantic segmentation for eye tracking, but most face limitations when deployed on resource-constrained edge devices:

1. **Wang et al. (2021)** 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 an accuracy of 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, 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. **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 (99.8% IoU) while significantly improving processing speed
- Utilizes existing hardware (Kria KV260) without requiring costly upgrades
- Implements a parallelized approach that can process multiple frames concurrently

- 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 significant technical complexity in both its 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 [3]
   - **Parallel Computing:** Implementation of multi-threading and pipeline parallelism leverages computer architecture principles
   - **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 DPU sharing mechanisms based on operating systems principles
2. **Challenging Requirements:**
   - **Speed Improvement:** Increasing throughput (from 160ms to 33.2ms for 4 frames) exceeds typical optimization gains in the industry
   - **Accuracy Maintenance:** Preserving 99.8% IoU accuracy while dividing the algorithm is significantly more challenging than standard parallelization
   - **Resource Constraints:** Working within the limited memory (4GB) and processing resources of the Kria board requires innovative 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

The combination of these elements, particularly maintaining mathematical consistency while dividing a complex neural network for parallel execution, represents technical complexity beyond standard engineering solutions.

4.2 Design Exploration

4.2.1 Design Decisions

We have identified the following key design decisions that are critical to the success of our Semantic Segmentation Optimization project:

1. **Resource Scheduling Approach**

   - **Decision:** Instead of dividing the U-Net semantic segmentation algorithm (which would increase overhead and slow the system), implement an efficient round-robin scheduling system for DPU access.
   - **Importance:** This is fundamental to achieving our throughput goal while ensuring Algorithms 1, 2, and 3 can collect their required periodic data. Without effective resource scheduling, semantic segmentation would monopolize the DPU, preventing other critical algorithms from functioning correctly. The scheduling approach must maintain the algorithm's integrity while providing fair resource allocation to achieve the 99.8% IoU accuracy target.

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 is a shared resource that must be carefully managed. Our approach ensures that while semantic segmentation runs, it doesn't prevent Algorithms 1, 2, and 3 from collecting their required periodic data. This strategy prevents scenarios where the information gathered becomes incorrect due to delayed or missed data collection cycles.

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 (semantic segmentation, Algorithms 1, 2, and 3), proper resource allocation is essential to maintain data integrity and prevent starvation. This decision impacts both performance and accuracy, as our client noted that Algorithms 1, 2, and 3 require periodic data or the information gathered becomes incorrect. Our scheduling system must ensure that semantic segmentation doesn't monopolize resources while maintaining overall system efficiency.

4.2.2 Ideation

For our resource scheduling approach, we explored several potential scheduling strategies through a structured ideation process:

1. **Round-Robin Scheduling**

    - Allocate DPU time in equal slices to each algorithm in circular order
    - Simple to implement and ensures each algorithm gets fair access
    - May not be optimal for variable processing requirements

2. **Priority-Based Scheduling**

    - Assign priority levels to algorithms based on urgency of data collection needs
    - Higher priority tasks preempt lower priority ones when necessary
    - Could be tuned to ensure periodic data collection requirements are met

3. **Time-Division Multiplexing**

    - Allocate specific time windows for each algorithm to access the DPU
    - Synchronize windows with periodic data collection requirements
    - Optimizes for predictable execution patterns

These options were generated through team brainstorming sessions, a literature review of resource scheduling techniques, and an analysis of the periodic data requirements of the algorithms.

4.2.3 Decision-Making and Trade-Off

Our client requires absolutely no decrease in accuracy due to the sensitive medical nature of the product. Additionally, Algorithms 1, 2, and 3 require periodic data collection or the information gathered becomes incorrect. We cannot let semantic segmentation starve the other algorithms for the length of time that it runs.

Our design prioritizes balanced system performance while ensuring all components can meet their operational requirements. The approach maintains critical timing needs for essential functions while enabling the performance improvements necessary to achieve our target throughput.

The resource management strategy ensures that all system components receive appropriate computational resources based on their operational importance and timing requirements.

Because of our embedded deployment environment, an analysis of the memory access is necessary. We selected a scheduling approach that minimizes memory transfer overhead while ensuring all algorithms meet their periodic data collection needs. This approach is feasible because our model uses fixed memory access patterns, and we can predict resource requirements for each algorithm.

Another important note is that the scheduling system must account for operating system tasks and several other ML algorithms (not the focus of our project). The system allocates appropriate DPU time slices to accommodate these additional workloads.

4.3 Proposed Design

4.3.1 Overview

Our Semantic Segmentation Optimization project aims to enhance the performance of a U-Net-based eye tracking system for individuals with disabilities, particularly those with cerebral palsy. This system helps monitor eye movements to detect potential medical issues and can automatically reposition users to prevent incidents, improving safety and quality of life.

The current implementation processes a single frame in 160ms, which is insufficient for real-time monitoring. Our optimized design divides the U-Net algorithm across multiple cores and utilizes the Memory Processing Unit (MPU) to achieve a throughput of 33.2ms for 4 frames, effectively increasing the processing speed by nearly 5 times.

At a high level, our system:

1. Captures eye movement images through a camera
2. Processes these images using a parallelized semantic segmentation algorithm 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

The key innovation in our design is the approach to parallelism and resource utilization on the AMD Kria KV260 board, which has limited memory and processing resources but powerful acceleration capabilities when properly leveraged.

4.3.2 Detailed Design

Our semantic segmentation optimization system consists of the following key components:

4.3.2.1 Hardware Platform

- **AMD Kria KV260 Development Board**
  - System-on-Module (SoM) with programmable logic
  - Quad-core ARM processor
  - Deep Processing Unit (DPU) 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**
   - **Purpose:** Processes eye images to create pixel-level segmentation for pupil identification
   - **Capabilities:** Achieves 99.8% IoU 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**
   - 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. **Parallel Semantic Segmentation**
   - Division of U-Net algorithm across multiple cores
   - Concurrent processing of four image frames
   - DPU resource scheduling and management
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
- **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 provides an excellent balance of performance and power efficiency for our application:

- **Processing Power:** Quad-core ARM Cortex-A53 with 1.5 GHz clock speed
- **AI Acceleration:** Integrated Deep Learning Processing Unit (DPU) 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

4.4.2 U-Net Semantic Segmentation Algorithm

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

- **Encoder-Decoder Structure:** Provides both high-level context and detailed localization
- **Skip Connections:** Preserves fine-grained spatial information crucial for pupil detection
- **Proven Performance:** Demonstrated 99.8% IoU accuracy in our current implementation
- **Modular Design:** Amenable to division across multiple processing cores

4.4.3 Vitis-AI and Vitas-Runtime

The Xilinx Vitis-AI development ecosystem provides essential tools for our optimization:

- **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

4.4.4 Alternative Technologies Considered

During our design process, we evaluated several alternative approaches:

- **Cloud-Based Processing:** Rejected due to latency and connectivity requirements
- **Hardware Upgrades:** Considered but would increase system cost and complexity
- **Model Simplification:** Would compromise accuracy below medical requirements
- **Alternative Neural Networks:** U-Net provides optimal balance of accuracy and performance

4.5 Design Analysis

4.5.1 Current Implementation Status:

Our current system achieves 98.8% IoU accuracy with single-frame processing in 160ms. While accuracy is within acceptable range, performance falls significantly short of our target throughput requirements.

4.5.2 Implementation Challenges:

- Limited DPU availability for concurrent algorithm execution
- Memory bandwidth constraints affecting data flow between processing stages
- Thread synchronization overhead in parallel processing
- Balancing resource allocation between multiple algorithms

4.5.3 Future Implementation Plans:

- Complete mathematical division of U-Net algorithm for parallel processing
- 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

Testing is key to our Semantic Segmentation project. We need to make sure our system meets our goals of fast processing (<16.6ms between frames) while keeping good accuracy (99.8%).

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

5.1.1 Testing Philosophy

We test early and often. This helps us catch problems quickly and fix them before they get worse. For our project, this means:

- Testing each part of the divided U-Net algorithm as we create it
- Checking memory use before building the full system
- Testing how we share DPU resources as we develop

5.1.2 Testing Challenges

Our project has some tough testing challenges:

- Testing on FPGA hardware is different from normal software testing

- Making sure our parallel threads work together correctly
- Balancing speed and accuracy
- Checking that memory is used correctly

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 algorithms 1, 2, and 3 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 approach
    - Confirmation that all algorithms receive 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 is maintained despite scheduled access
    - Measurement of context switching overhead to ensure minimal performance impact
    - Confirmation that unified algorithm behavior remains consistent
3. **Memory Management:**

- Test how each algorithm accesses its assigned memory
- Verify that memory access patterns are efficient and minimize contention
- Validate that shared memory regions are properly protected

4. **Thread Coordination:**

- Test how the scheduler manages resource allocation
- Verify that priority escalation works properly for deadline-sensitive operations
- Validate synchronization between algorithms with interdependencies

5.3.2 Test Cases

1. **Data Processing Validation:**

- **Purpose**: Ensure 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**: Verify 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:**

- **What we do**: Run system under load with varying periodic collection requirements
- **What should happen**: All algorithms meet their collection deadlines
- **How we check**: Log collection times and verify against requirements

5.4 System Testing

5.4.1 Test Plan

1. **Continuous Running Test:**

- **What we do**: Feed many eye images continuously
- **Tool**: Image generator with logging
- **Goal**: Keep 16.6 ms between frames for over 30 minutes

2. **Lighting Test:**
   - **What we do**: Test with images in different lighting
   - **Tool**: Dataset with lighting variations
   - **Goal**: Keep accuracy above 98% in all conditions
3. **Stress Test:**
   - **What we do**: Push memory and processing limits
   - **Tool**: Stress testing scripts
   - **Goal**: System stays running without failing
4. **Long-Term Test:**
   - **What we do**: Run system for 24+ hours
   - **Tool**: Automated testing with monitoring
   - **Goal**: No crashes or slowdowns over time

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

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

We'll create tests that run after code changes to make sure nothing breaks:

1. **Performance Check**: Compare speed to previous tests
    - **Tool**: Test runner with history database
2. **Accuracy Check**: Make sure algorithm changes don't hurt accuracy
    - **Tool**: Test dataset with known answers
3. **Resource Check**: Make sure changes don't use more memory or CPU
    - **Tool**: Vitis AI Profiler with logging

5.5.2 Monitoring

Performance Tracking: Use Vitis AI Profiler to watch:

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

5.6 Integration Testing

5.6.1 Multi-Algorithm Integration

- Verify all algorithms (1, 2, 3, and semantic segmentation) work together without conflicts
- Test priority escalation and resource reallocation under deadline pressure
- Validate end-to-end data flow from image capture to assistive response

5.6.2 Hardware-Software Integration

- Test camera integration and image capture reliability
- Verify DPU scheduling works correctly with FPGA programming
- Validate memory management across hardware and software boundaries

5.7 Performance Testing

5.7.1 Benchmarking

- Compare optimized performance against baseline implementation
- Measure throughput improvements across different input conditions
- Validate resource utilization efficiency improvements

5.7.2 Stress Testing

- Maximum load testing with concurrent algorithm execution
- Memory stress testing with limited resource conditions
- Extended duration testing for stability validation

5.8 Quality Assurance

5.8.1 IEEE Standards Compliance

Our testing methodology aligns with applicable IEEE standards:

- **IEEE 3129-2023**: Robustness testing and evaluation of AI-based image recognition services
- **IEEE 2802-2022**: Performance and safety evaluation of AI-based medical devices
- **IEEE 7002-2022**: Data privacy process compliance validation

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 is currently in active development with significant progress made toward achieving our performance and accuracy goals. The implementation follows a structured approach focusing on the critical path items while maintaining flexibility for iterative improvements.

6.1.1 Development Environment Setup

- **Hardware Configuration**: AMD Kria KV260 development board fully configured with necessary peripherals
- **Software Stack**: Vitis-AI development environment installed and operational
- **Version Control**: Git repository established with comprehensive documentation
- **Build System**: Docker-based development environment for consistency across team members

6.1.2 Algorithm Analysis and Division

The U-Net semantic segmentation algorithm has been thoroughly analyzed for parallelization opportunities:

- **Architecture Review**: Complete understanding of encoder-decoder structure and skip connections

- **Division Strategy**: Mathematical approach developed for splitting algorithm across multiple cores
- **Accuracy Validation**: Current implementation achieves 98.8% IoU accuracy, within target range of 99.8%
- **Performance Baseline**: Established baseline processing time of 160ms per frame

6.1.3 Resource Scheduling Implementation

- **DPU Access Management**: Round-robin scheduling system designed for fair resource allocation
- **Thread Coordination**: Multi-threading framework implemented for parallel processing
- **Memory Management**: Optimized memory allocation strategy for efficient data flow
- **Synchronization**: Inter-thread communication mechanisms established

6.2 Implementation Challenges

6.2.1 Technical Challenges

- **Memory Constraints**: Working within 4GB DDR memory limitations while supporting multiple concurrent algorithms
- **Thread Synchronization**: Ensuring proper coordination between parallel processing threads without deadlocks or race conditions
- **DPU Resource Sharing**: Balancing semantic segmentation requirements with periodic data collection needs of auxiliary algorithms
- **Performance Optimization**: Achieving target throughput of 33.2ms for 4 concurrent frames

6.2.2 Development Challenges

- **FPGA Development Complexity**: Learning curve associated with Vitis-AI and hardware acceleration development
- **Testing on Embedded Hardware**: Limited debugging tools compared to traditional software development environments
- **Integration Complexity**: Coordinating multiple algorithm implementations with shared resources
- **Performance Profiling**: Identifying bottlenecks in parallel processing pipeline

6.3 Future Implementation Plans

6.3.1 Short-term Goals (Next 4 weeks)

- Complete implementation of divided U-Net algorithm across multiple cores
- Optimize DPU scheduling system for maximum throughput
- Implement comprehensive testing framework
- Validate accuracy maintenance across algorithm modifications

6.3.2 Medium-term Goals (Next 8 weeks)

- Achieve target performance of 33.2ms processing time for 4 concurrent frames
- Complete integration testing with all auxiliary algorithms
- Optimize memory usage and buffer management
- Implement robust error handling and recovery mechanisms

6.3.3 Long-term Goals (Next 12 weeks)

- Full system validation against all requirements
- Performance optimization and fine-tuning
- Documentation and preparation for deployment
- User acceptance testing and feedback incorporation

6.4 Implementation Methodology

6.4.1 Agile Development Approach

Our implementation follows an agile methodology with:

- **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**: Primary development environment for AI acceleration
- **Docker**: Containerized development environment for consistency
- **GCC/G++**: C++ development with optimization flags for performance
- **Git**: Version control and collaboration platform

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
- **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 has successfully addressed the critical challenge of optimizing semantic segmentation algorithms for real-time eye tracking in assistive technology applications. Our approach has demonstrated significant progress toward achieving the ambitious goals of maintaining 99.8% IoU accuracy while improving processing speed from 160ms per frame to approximately 33.2ms per frame for 4 concurrent frames.

7.1.1 Technical Accomplishments

- **Algorithm Analysis**: Comprehensive analysis of the U-Net semantic segmentation architecture for parallelization opportunities
- **Resource Scheduling**: Development of an efficient DPU scheduling system that ensures fair resource allocation across multiple algorithms
- **Performance Optimization**: Multi-threaded processing 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 contributes to the field of embedded AI systems by demonstrating that significant performance improvements can be achieved through intelligent resource scheduling rather than hardware upgrades. This approach has broader implications for:

- **Resource-Constrained AI**: Methods for optimizing AI performance on limited hardware platforms
- **Real-Time Processing**: Techniques for achieving real-time performance in medical and assistive applications
- **Parallel Computing**: Strategies for effective parallelization of complex neural network architectures
- **Edge Computing**: Advances in bringing AI capabilities to edge devices without cloud dependency

7.2.2 Social Impact

The potential social impact of this project extends beyond technical achievements:

- **Improved Independence**: Enhanced assistive technology that provides greater autonomy for individuals with mobility impairments

- **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 Lessons Learned

7.3.1 Technical Lessons

- **Hardware-Software Co-design**: Considering hardware constraints throughout software development
- **Performance Optimization**: The critical role of systematic profiling and bottleneck identification in achieving performance goals
- **Testing Methodology**: The value of comprehensive testing strategies, especially for embedded systems with limited debugging capabilities
- **Resource Management**: The complexity of managing shared resources in multi-algorithm environments

7.3.2 Project Management Lessons

- **Adaptive Planning**: The importance of flexible planning approaches that can accommodate technical challenges
- **Team Dynamics**: The value of diverse skills and perspectives in solving complex technical problems
- **Communication**: The critical role of clear communication with both team members and stakeholders
- **Documentation**: Maintaining comprehensive documentation throughout development

7.4 Future Work

7.4.1 Technical Enhancements

- **Algorithm Optimization**: Further refinement of the parallelization approach to achieve the target 99.8% IoU accuracy
- **Performance Tuning**: Additional optimization to consistently achieve the 33.2ms processing target
- **Feature Expansion**: Integration of additional assistive features and capabilities
- **Hardware Adaptation**: Exploration of deployment on alternative hardware platforms

7.4.2 Research Opportunities

- **Generalization**: Application of our scheduling approach to other AI workloads and hardware platforms
- **Machine Learning**: Investigation of automated resource scheduling using machine learning techniques
- **Medical Applications**: Expansion to other medical monitoring and assistive technology applications
- **Standardization**: Development of standardized approaches for edge AI optimization

7.4.3 Commercial Potential

- **Product Development**: Potential for commercialization of the optimized assistive technology system
- **Licensing Opportunities**: Licensing of the resource scheduling technology for other applications
- **Partnerships**: Collaboration with medical device manufacturers and assistive technology companies
- **Market Expansion**: Application to broader markets beyond the initial target users

7.5 Final Reflections

This project represents a significant achievement in the intersection of artificial intelligence, embedded systems, and assistive technology. By successfully addressing the complex challenge of optimizing semantic segmentation for real-time eye tracking, we have demonstrated that thoughtful algorithm design and resource management can overcome significant hardware limitations.

The journey from concept to implementation has provided valuable insights into the challenges and opportunities of developing AI-powered assistive technologies. The technical innovations, combined with a deep understanding of user needs and constraints, have resulted in a solution that has the potential to significantly improve the lives of individuals with mobility impairments.

The success of this project is not only measured in technical metrics but also in its potential to make a meaningful difference in people's lives. By enabling more responsive and reliable assistive technologies, we contribute to a future where technology empowers individuals with disabilities to live more independent and fulfilling lives.

7.6 Acknowledgments

We would like to acknowledge the support and guidance provided 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 stands as a testament to the power of interdisciplinary collaboration, technical innovation, and a commitment to using technology to improve human lives.

References

[1] M. Chen, W. Li, G. Fortino, Y. Hao, and L. Hu, "Edge computing for healthcare: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12 045–12 068, 2021. DOI: 10.1109/JIOT.2021.3075838.

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