A

Project Report On

Virtual Memory And Cache Optimization

Submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Pranita Padaliya 2261427

Prajjwal Singh Kirola 2261425

Lokesh Bankoti 2261335

Anjali Nitwal 2261098

Under the Guidance of Mr.

Anubhav Bewerwal

Assistant Professor

Project Team ID:01

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal, Uttarakhand

2024-2025

Graphic Era Hill University
BHIMTAL CAMPUS

STUDENT'S DECLARATION

We Pranita Padaliya, Prajjwal Singh Kirola, Lokesh Bankoti, Anjali Nitwal hereby declare that the work, which is being presented in the project, entitled **"Virtual Memory and Cache Optimization"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology(B.Tech) in the session 2024-25 is an authentic record of our work carried out under supervision of **Mr. Anubhav Bewerwal.**

The matter embodied in this project has not been submitted by us for the award of any other degree.

Pranita Padaliya	2261427
Prajjwal Singh Kirola	2261425
Lokesh Bankoti	2261335
Anjali Nitwal	2261098

Date:

Graphic Era Hill University
BHIMTAL CAMPUS

CERTIFICATE

The project report entitled "**Virtual Memory and Cache Optimizer"** being submitted by **Pranita Padaliya** (2261427), **Prajjwal Singh Kirola** (2261425), **Lokesh Bankoti** (2261335) and **Anjali Nitwal** (2261098) of BTech CSE to Graphic Era Hill University Bhimtal campus for the award of bonafide work carred out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of the report.

Dr. Anubhav Bewerwal
(Project Guide)

Dr. Ankur Singh Bisht
(Head- CSE)

Graphic Era Hill University
BHIMTAL CAMPUS

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **'Anubhav Bewerwal'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Pranita Padaliya	2261427
Prajjwal Singh Kirola	2261425
Lokesh Bankoti	2261335
Anjali Nitwal	2261098

ABSTRACT

Virtual Memory and Cache Optimizer

Overview

The Memory and Cache Optimizer is a comprehensive system utility that provides real-time monitoring and optimization of system memory and cache performance. The application offers both desktop and web interfaces, making it accessible across different platforms and user preferences.

Core Features

1. Memory Management

Real-time Monitoring

- Tracks total, available, used, and free memory
- Monitors swap memory usage
- Provides percentage-based memory utilization
- Historical memory usage tracking

Memory Optimization

- Cleans temporary files
- Manages system cache
- Optimizes page file usage
- Handles memory fragmentation
- Supports both user-level and admin-level optimizations

2. Cache Management

Cache Performance Tracking

- Monitors cache hit/miss ratios
- Tracks access times
- Measures eviction rates
- Analyzes write-back performance

Cache Optimization

- Clears browser caches
- Manages application caches

- Optimizes system cache
- Handles Windows temporary cache

Performance Analytics

- **System Metrics**
- Response time monitoring
- Throughput analysis
- Page fault tracking
- Swap rate monitoring
- **Visualization**
- Real-time performance graphs
- Historical trend analysis
- Before/after optimization comparisons
- Interactive data visualization

Impact and Benefits

1. **System Performance**

- Improved memory utilization
- Enhanced cache efficiency
- Reduced system slowdown
- Better resource management

2. **User Experience**

- Easy-to-use interfaces
- Real-time feedback
- Detailed optimization reports
- Flexible access options

TABLE OF CONTENTS

Chapter 1

Introduction and Problem Statement

1. Introduction

In modern computing systems, efficient memory management is a crucial factor in ensuring optimal performance. As applications continue to grow in size and complexity, the demand for memory resources also increases, making it essential to manage memory efficiently. One of the fundamental concepts that play a key role in memory management is virtual memory, which allows an operating system (OS) to provide an abstraction of memory that appears larger than the actual physical memory available. Virtual memory not only extends the available memory space but also enables effective multitasking and process isolation, thus improving system performance and stability.

Virtual memory is a technique that enables programs to use more memory than what is physically available by utilizing secondary storage (e.g., a hard disk or SSD) as an extension of RAM. This process involves paging and segmentation, which allow dynamic memory allocation and efficient address translation. Paging divides memory into fixed-sized blocks, reducing fragmentation, while segmentation offers more flexibility by dividing memory into variable-sized sections based on logical divisions within programs. While virtual memory improves memory utilization and process management, it introduces challenges related to performance overhead due to page faults, memory fragmentation, and inefficient memory access patterns.

To overcome these challenges, optimization techniques are employed in modern computing systems. Optimization of virtual memory involves strategies such as efficient page replacement algorithms (Least Recently Used, First-In-First-Out, Optimal Page Replacement), memory compression techniques, prefetching strategies, and adaptive memory allocation policies. These techniques help in reducing page faults, improving memory access speeds, and ensuring better system stability. Additionally, advanced caching mechanisms and intelligent memory allocation algorithms can significantly enhance memory utilization, reducing latency and improving responsiveness.

In this work, we explore the various methods and techniques used to optimize virtual memory management. The primary goal is to develop an approach that enhances system performance by reducing memory overhead and increasing overall efficiency. This project focuses on implementing effective memory management strategies that can dynamically adapt to system demands, providing seamless performance improvements. By leveraging data-driven approaches, predictive memory management models, and optimized cache handling, we aim to contribute to the development of an efficient and intelligent virtual memory management system.

2. Problem Statement

The increasing complexity of modern applications and the growing demand for memory-intensive tasks pose significant challenges in memory management. Conventional memory management techniques often fail to efficiently utilize available resources, leading to issues such as **high page fault rates, memory fragmentation, and excessive swapping** between physical and virtual memory. These inefficiencies can result in degraded system performance, increased power consumption, and slower application response times.

The problem addressed in this study can be stated as follows:

- **Memory Overhead and Performance Bottlenecks:** Traditional virtual memory mechanisms introduce significant memory overhead due to frequent page faults and excessive disk I/O operations, leading to performance degradation.
- **Inefficient Page Replacement Strategies:** Existing page replacement algorithms, such as Least Recently Used (LRU) and First-In-First-Out (FIFO), may not always yield optimal performance in real-world workloads, necessitating the need for adaptive and intelligent replacement strategies.
- **Limited Adaptability of Memory Allocation Policies:** Many systems use fixed allocation policies that do not adapt to changing workload demands, leading to inefficient memory utilization.
- **Increased Latency in Memory Access:** The process of swapping pages between RAM and secondary storage results in increased access latency, negatively impacting real-time applications and multitasking environments.

This project aims to **analyze and propose an optimized virtual memory management framework** that minimizes performance bottlenecks and enhances overall efficiency. The study will focus on **developing and evaluating advanced memory optimization techniques**, such as:

1. **Enhanced Page Replacement Policies:** Implementing intelligent and adaptive page replacement strategies to reduce page fault rates and improve memory utilization.
2. **Memory Compression Techniques:** Exploring in-memory data compression methods to maximize RAM efficiency and reduce swap dependency.
3. **Prefetching Mechanisms:** Implementing predictive memory prefetching strategies to minimize access delays.
4. **Adaptive Memory Allocation:** Designing dynamic memory allocation policies that adjust according to application needs in real time.
5. **Hybrid Memory Solutions:** Investigating the combination of RAM and SSD-based virtual memory optimization to reduce swapping overhead.

Through the implementation of these techniques, the study aims to contribute to the development of more efficient virtual memory management strategies that can enhance the performance of operating systems, improve application responsiveness, and reduce overall system resource consumption. The findings of this research will be applicable to a wide range of computing environments, including desktops, cloud computing platforms, and embedded systems.

By optimizing virtual memory, modern computing systems can achieve **better performance, lower power consumption, and enhanced reliability**, making this an essential area of study in software engineering and operating system design.

Chapter 2

Background/ Literature Survey

In modern computing systems, efficient virtual memory and cache management play a crucial role in optimizing system performance. Research in this domain has focused on developing advanced algorithms and techniques to reduce memory overhead, improve caching efficiency, and minimize page faults. This chapter presents a review of the major existing work in the field of virtual memory optimization and cache management.

2.1 Virtual Memory Optimization

Virtual memory allows an operating system to extend the available memory beyond physical RAM by utilizing secondary storage as an extension. However, inefficient virtual memory management can lead to issues such as excessive page faults, increased system latency, and resource wastage. Various studies have been conducted to address these challenges through optimization techniques:

- **Page Replacement Algorithms:** Traditional page replacement algorithms such as **Least Recently Used (LRU), First-In-First-Out (FIFO), and Optimal Page Replacement** have been widely used to improve memory allocation. LRU focuses on replacing the least recently used pages to minimize faults, while FIFO replaces pages in the order they were loaded. The optimal algorithm theoretically achieves the best performance by replacing the page that will not be used for the longest duration.
- **Memory Compression Techniques:** To reduce memory pressure, researchers have explored in-memory compression techniques that store more data in RAM by compressing less frequently accessed pages. These techniques help in minimizing swapping and improving access speeds.
- **Prefetching Strategies:** Advanced predictive algorithms analyze access patterns to load data into memory before it is requested, reducing latency and enhancing responsiveness.
- **Adaptive Memory Allocation Policies:** Dynamic allocation of memory based on real-time system demands ensures efficient utilization, reducing fragmentation and performance degradation.

2.2 Cache Management Techniques

Cache memory acts as a high-speed storage layer between the processor and main memory, improving data retrieval times. However, poor cache management can lead to inefficient access patterns and increased execution times. Several approaches have been proposed to optimize cache utilization:

- **Cache Replacement Policies:** Similar to virtual memory management, cache replacement strategies such as **Least Recently Used (LRU), Least Frequently Used (LFU), and Random Replacement** determine which cache entries to evict for improved performance.
- **Intelligent Cache Prefetching:** Machine learning-based models analyze access patterns and predict future memory accesses, ensuring that frequently accessed data is readily available.
- **Multi-Level Cache Optimization:** Modern architectures employ multi-level cache hierarchies (L1, L2, L3), and research has focused on optimizing data flow across these levels to reduce redundancy and improve efficiency.
- **API-based Cache Integration:** Studies have explored integrating caching mechanisms with backend systems through APIs to allow dynamic cache updates and monitoring.

2.3 Existing Research and Challenges

Several studies have explored different aspects of virtual memory and cache optimization, highlighting key challenges such as:

- The trade-off between memory compression and computational overhead.
- The impact of cache eviction policies on different workloads.
- The difficulty in achieving an optimal balance between prefetching aggressiveness and memory resource constraints.

2.4 Conclusion

Based on the reviewed literature, optimizing virtual memory and cache management remains an active research area. Existing techniques like page replacement algorithms, caching, and memory compression enhance performance but need dynamic adaptation. This project aims to integrate these strategies into a unified framework for better efficiency and resource utilization.

Chapter 3

Objectives

The objectives of the proposed work are as follows:

1. **Optimize Virtual Memory Management**
   Virtual memory plays a crucial role in modern computing by allowing processes to use more memory than what is physically available. This work aims to optimize virtual memory allocation and management techniques to ensure efficient memory utilization, minimize fragmentation, and reduce unnecessary swapping between RAM and secondary storage.
2. **Improve Cache Performance**
   Caching strategies are essential for reducing memory access time and improving overall system speed. This objective focuses on implementing intelligent cache management policies, such as multi-level caching and adaptive cache allocation, to enhance data retrieval efficiency and reduce bottlenecks.
3. **Enhance Page Replacement Algorithms**
   Page replacement algorithms determine which memory pages should be swapped in and out when physical memory becomes full. This work aims to improve existing algorithms like LRU (Least Recently Used) and LFU (Least Frequently Used) by integrating adaptive strategies that dynamically adjust to system workload and real-time memory demands.
4. **Reduce Memory Overhead**
   Inefficient memory usage leads to increased overhead, affecting system performance. This objective involves employing memory compression techniques, prefetching mechanisms, and efficient memory allocation policies to optimize memory usage and improve execution efficiency.
5. **Develop a Unified Optimization Framework**
   Rather than relying on isolated optimization techniques, this work aims to develop a cohesive framework that integrates multiple memory optimization strategies. By combining page replacement improvements, caching enhancements, and memory compression, the framework will maximize resource utilization and ensure seamless system performance.

Chapter 4

Hardware and Software Requirements

4.1 Hardware Requirements

Sl. No	Name of the Hardware	Specification
1	Processor	Intel Core i5,i7(or equivalent)
2	RAM	8GB or higher
3	Storage	256GB SSD (minimum)
4	GPU(if required)	Integrated or dedicated(optional)
5	Network Connectivity	Wi-Fi/Ethernet

4.2 Software Requirements

Sl. No	Name of the Software	Specification
1	Operating System	Windows 10/Linux(Ubuntu)/macOS
2	Programming Language	Python(latest stable version)
3	Database	SQLite
4	Web Framework	Flask/Django(for API development)
5	Frontend Technologies	HTML,Javascript,React.js
6	Data Visualization Library	Chart.js
7	Version Control System	Git/Github
8	ID/Text Editor	VS Code/PyCharm

Chapter 5

Possible Approach/ Algorithms

Virtual Memory and Cache play a crucial role in system performance. The optimization of these components enhances execution speed and ensures efficient resource utilization. This chapter discusses the possible approaches and algorithms used in our project, "Virtual Memory & Cache Optimizer for Performance Enhancement." The focus is on optimizing memory allocation, improving cache efficiency, and minimizing page faults using intelligent algorithms.

1. Page Replacement Algorithms

Page replacement policies are crucial in managing virtual memory efficiently. The following algorithms are considered for implementation:

a) Least Recently Used (LRU)

- Keeps track of the pages that were used most recently.
- Replaces the page that has not been used for the longest time.
- Uses data structures like stacks or counters for implementation.

b) First In First Out (FIFO)

- Treats pages as a queue, where the oldest page is replaced first.
- Simple to implement but can lead to more page faults (Belady's anomaly).

c) Optimal Page Replacement

- Replaces the page that will not be used for the longest time in the future.
- Provides the lowest possible page fault rate but requires future knowledge of memory accesses.

2. Cache Optimization Techniques

To improve cache efficiency and reduce miss rates, the following techniques are incorporated:

a) LRU Cache Replacement

- Keeps frequently accessed cache blocks longer.
- Implemented using data structures like HashMaps with Doubly Linked Lists.

b) Adaptive Cache Replacement

- Adjusts the replacement strategy dynamically based on workload.
- Uses machine learning models to predict frequently accessed data.

c) Multi-level Caching

- Implements multiple cache levels (L1, L2, L3) with different sizes and speeds.
- Reduces latency by efficiently utilizing cache hierarchy.

3. Efficient Memory Allocation Strategies

To minimize memory fragmentation and improve resource utilization, advanced allocation strategies are applied:

a) Buddy Memory Allocation

- Divides memory into blocks of power-of-two sizes.
- Merges free blocks to create larger contiguous spaces when necessary.

b) Slab Allocation

- Pre-allocates memory chunks for frequently requested object sizes.
- Reduces memory fragmentation and improves allocation speed.

c) Dynamic Memory Compaction

- Periodically reorganizes memory to consolidate free space.
- Reduces external fragmentation and enhances memory efficiency.

Algorithm Implementation

Table -Pseudo Code for LRU Page Replacement Algorithm

```
Input: Page reference sequence, Cache size N
Output: Number of page faults
Begin
1. Initialize an empty cache with size N
2. For each page in the reference sequence do:
    a) If the page is already in cache, move it to the front
    b) If the page is not in cache:
        - If the cache is full, remove the least recently used
page
        - Add the new page to the front of the cache
    c) Increase the page fault count if replacement occurs
3. End For
4. Return the page fault count
End
```

Code

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Memory & Cache Monitor</title>
   <link rel="preconnect" href="https://fonts.googleapis.com">
   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
   <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700&displ
ay=swap" rel="stylesheet">
   <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css"
rel="stylesheet">
   <link href="{{ url_for('static', filename='style.css') }}" rel="stylesheet">
   <style>
      :root {
         --primary-color: #6c5ce7;
         --secondary-color: #00cec9;
         --background-color: #1e272e;
         --card-background: #2d3436;
         --text-color: #f5f6fa;
         --accent-color: #fd79a8;
```

```
    --text-secondary: #a4b0be;
    --success-color: #2ecc71;
    --warning-color: #f39c12;
    --error-color: #e74c3c;
}

body {
    font-family: 'Poppins', sans-serif;
    margin: 0;
    padding: 20px;
    background-color: var(--background-color);
    color: var(--text-color);
}

.container {
    max-width: 1200px;
    margin: 0 auto;
}

.stats-card::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: linear-gradient(45deg, var(--primary-color), transparent);
    opacity: 0;
    z-index: -1;
    transition: opacity 0.4s ease;
}

.stats-card:hover {
    transform: translateY(-8px);
}

.stats-card:hover::before {
    opacity: 0.05;
}

.button-container {
    display: flex;
    justify-content: center;
    gap: 20px;
    margin-bottom: 30px;
}


#visualization {
    background: var(--card-background);
```

```
    padding: 30px;
    border-radius: 20px;
    box-shadow: 0 10px 30px rgba(0,0,0,0.15);
    border-top: 4px solid var(--secondary-color);
    margin-top: 20px;
    color: var(--text-color);
    overflow: hidden;
}

/* For Plotly SVG text elements */
#visualization .main-svg text {
    fill: var(--text-color) !important;
}

/* Make sure Plotly legends are visible */
#visualization .legend text {
    fill: var(--text-color) !important;
}

/* Ensure axes labels are readable */
#visualization .xtitle, #visualization .ytitle {
    fill: var(--text-color) !important;
    font-weight: 500 !important;
}

.stat-value {
    font-size: 28px;
    font-weight: bold;
    background: linear-gradient(135deg, #a29bfe, #00ffff);
    -webkit-background-clip: text;
    background-clip: text;
    -webkit-text-fill-color: transparent;
    margin: 12px 0;
    position: relative;
    display: inline-block;
}

.stat-value::after {
    content: '';
    position: absolute;
    bottom: -5px;
    left: 0;
    height: 3px;
    width: 40px;
    background: var(--accent-color);
    border-radius: 3px;
}

.stat-label {
    font-size: 15px;
```

```
    color: #ffffff;
    margin-bottom: 8px;
    text-transform: uppercase;
    letter-spacing: 1px;
    font-weight: 500;
}

/* Alternative non-gradient text style in case gradient text is hard to read */
.plain-text-value {
    font-size: 28px;
    font-weight: bold;
    color: #00ffff;
    margin: 12px 0;
    position: relative;
    display: inline-block;
}

@keyframes pulse {
    0% { opacity: 0.5; }
    50% { opacity: 1; }
    100% { opacity: 0.5; }
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

/* Custom Alert Styles */
.custom-alert {
    display: none;
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background: var(--card-background);
    padding: 40px;
    border-radius: 20px;
    box-shadow: 0 20px 40px rgba(0,0,0,0.3);
    z-index: 1001;
    text-align: center;
    min-width: 350px;
    border-left: 5px solid var(--accent-color);
    animation: alertFadeIn 0.4s ease;
}

@keyframes alertFadeIn {
    from { opacity: 0; transform: translate(-50%, -60%); }
    to { opacity: 1; transform: translate(-50%, -50%); }
}
```

```
.alert-icon {
    font-size: 60px;
    background: linear-gradient(135deg, #a29bfe, #00ffff);
    -webkit-background-clip: text;
    background-clip: text;
    -webkit-text-fill-color: transparent;
    margin-bottom: 25px;
    animation: iconPulse 2s infinite;
}

@keyframes iconPulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.1); }
    100% { transform: scale(1); }
}


.alert-button::before {
    content: '';
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background: linear-gradient(90deg, transparent, rgba(255,255,255,0.2), transparent);
    transition: all 0.5s ease;
}

.alert-button:hover {
    transform: translateY(-3px);
    box-shadow: 0 15px 25px rgba(108, 92, 231, 0.3);
}

.alert-button:hover::before {
    left: 100%;
}

.error-message {
    padding: 20px;
    text-align: center;
    color: #fd79a8;
    font-size: 16px;
    background-color: rgba(253, 121, 168, 0.1);
    border-radius: 10px;
    border-left: 4px solid #fd79a8;
    margin: 20px 0;
}

}
```

```
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1><i class="fas fa-microchip"></i> Memory & Cache Monitor</h1>
        </div>

        <div class="stats-container">
            <div class="stats-card" id="memory-stats">
                <h2><i class="fas fa-memory"></i> Memory Statistics</h2>
                <div id="memory-data"></div>
            </div>
            <div class="stats-card" id="cache-stats">
                <h2><i class="fas fa-database"></i> Cache Statistics</h2>
                <div id="cache-data"></div>
            </div>
        </div>

        <div class="button-container">
            <button onclick="optimizeMemory()">
                <i class="fas fa-bolt"></i> Optimize Memory
            </button>
            <button onclick="optimizeCache()">
                <i class="fas fa-rocket"></i> Optimize Cache
            </button>
        </div>

        <div id="visualization"></div>
    </div>

    <!-- Loading Overlay -->
    <div class="loading-overlay" id="loadingOverlay">
        <div class="loading-spinner"></div>
        <div class="loading-text">Optimizing...</div>
    </div>

    <!-- Custom Alert -->
    <div class="custom-alert" id="customAlert">
        <div class="alert-icon">
            <i class="fas fa-check-circle"></i>
        </div>
        <div class="alert-title">Optimization Complete</div>
        <div class="alert-message" id="alertMessage"></div>
        <button class="alert-button" onclick="closeAlert()">Continue</button>
    </div>

            `;
            $('#memory-data').html(memoryHtml);
```

```
            // Update cache stats
            let cacheHtml = `
                <div class="stat-label">Cache Hits</div>
                <div class="plain-text-value">${data.cache.hits}</div>
                <div class="stat-label">Cache Misses</div>
                <div class="plain-text-value">${data.cache.misses}</div>
                <div class="stat-label">Hit Ratio</div>
                <div class="plain-text-value">${(data.cache.hit_ratio *
100).toFixed(2)}%</div>
            `;
            $('#cache-data').html(cacheHtml);
        });

        // Update visualization
        $.get('/api/visualization', function(data) {
            if (!data.error) {
                Plotly.newPlot('visualization', data.data, data.layout, {
                    responsive: true,
                    displayModeBar: true
                });
            } else {
                console.error("Error getting visualization data:", data.error);
                $('#visualization').html('<div class="error-message">Error loading visualization.
Please refresh the page.</div>');
            }
        }).fail(function(jqXHR, textStatus, errorThrown) {
            console.error("AJAX request failed:", textStatus, errorThrown);
            $('#visualization').html('<div class="error-message">Failed to load visualization.
Please refresh the page.</div>');
        });
    }

    function optimizeMemory() {
        showLoading('Optimizing Memory...');
        $.get('/api/optimize-memory', function(data) {
            hideLoading();
            showAlert('Memory Optimization Complete', 'Memory has been successfully
optimized');
            updateStats();
        });
    }

    function optimizeCache() {
        showLoading('Optimizing Cache...');
        $.get('/api/optimize-cache', function(data) {
            hideLoading();
            showAlert('Cache Optimization Complete', 'Cache has been successfully
optimized');
            updateStats();
        });
```

```
        }

        // Update stats every 5 seconds
        setInterval(updateStats, 5000);

        // Initial update
        $(document).ready(function() {
            console.log("Document ready, initializing stats...");
            updateStats();
        });
    </script>
    <script src="{{ url_for('static', filename='script.js') }}"></script>
    <script src="{{ url_for('static', filename='compare.js') }}"></script>
</body>
</html>
```

camparison.py

```
from flask import Flask, jsonify, render_template
import time
import numpy as np
from datetime import datetime
import logging
import os
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from app.models import MemoryStats, CacheStats, PerformanceMetrics, MemoryOptimizer,
CacheOptimizer

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Initialize Flask app with correct template and static folders
template_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), 'templates'))
static_dir = os.path.abspath(os.path.join(os.path.dirname(os.path.dirname(__file__)),
'statics'))
app = Flask(__name__, template_folder=template_dir, static_folder=static_dir,
static_url_path='/static')

class MemoryMonitor:
    def __init__(self):
        self.memory_history = []
        self.cache_history = []
        self.timestamps = []
        self.optimization_history = {
            'memory': {'before': None, 'after': None, 'details': []},
```

```
            'cache': {'before': None, 'after': None, 'details': []}
        }
        self.performance_metrics = {
            'response_times': [],
            'throughput': [],
            'page_faults': [],
            'swap_usage': []
        }


# Initialize the monitor
monitor = MemoryMonitor()

# Error handlers
@app.errorhandler(404)
def not_found_error(error):
    return jsonify({'error': 'Not found'}), 404

@app.errorhandler(500)
def internal_error(error):
    return jsonify({'error': 'Internal server error'}), 500

@app.route('/')
def index():
    try:
        return render_template('index.html')
    except Exception as e:
        logger.error(f"Error rendering template: {str(e)}")
        return jsonify({'error': 'Internal server error'}), 500

@app.route('/api/real-time-stats')
def get_real_time_stats():
    try:
        monitor.record_stats()
        return jsonify({
            'memory': monitor.memory_history[-1],
            'cache': monitor.cache_history[-1],
            'timestamp': monitor.timestamps[-1].strftime('%H:%M:%S')
        })
    except Exception as e:
        logger.error(f"Error getting real-time stats: {str(e)}")
        return jsonify({'error': 'Internal server error'}), 500

@app.route('/api/optimize-memory')
def optimize_memory():
    try:
        before_stats = monitor.get_memory_stats()
        monitor.optimization_history['memory']['before'] = before_stats

        # Perform real memory optimization with details
```

```
        success, message, details = MemoryOptimizer.optimize_with_details()
        monitor.optimization_history['memory']['details'] = details

        if not success:
            logger.warning(f"Memory optimization failed: {message}")
            return jsonify({
                'success': False,
                'message': message,
                'before': before_stats,
                'after': before_stats,
                'details': details,
                'improvement': {
                    'before_ratio': before_stats['percent'],
                    'after_ratio': before_stats['percent'],
                    'improvement': 0
                }
            })

        # Get memory stats after optimization
        after_stats = monitor.get_memory_stats()
        monitor.optimization_history['memory']['after'] = after_stats

        return jsonify({
            'success': True,
            'message': message,
            'before': before_stats,
            'after': after_stats,
            'details': details,
            'improvement': compare_performance(
                before_stats['percent'],
                after_stats['percent']
            )
        })
    except Exception as e:
        logger.error(f"Error optimizing memory: {str(e)}")
        return jsonify({
            'success': False,
            'message': f"Error during memory optimization: {str(e)}",
            'error': str(e)
        }), 500

@app.route('/api/optimize-cache')
def optimize_cache():
    try:
        before_stats = monitor.get_cache_stats()
        monitor.optimization_history['cache']['before'] = before_stats

        # Perform real cache optimization with details
        success, message, details = CacheOptimizer.optimize_with_details()
        monitor.optimization_history['cache']['details'] = details
```

```
if not success:
    logger.warning(f"Cache optimization failed: {message}")
    return jsonify({
        'success': False,
        'message': message,
        'before': before_stats,
        'after': before_stats,
        'details': details,
        'improvement': {
            'before_ratio': before_stats['hit_ratio'],
            'after_ratio': before_stats['hit_ratio'],
            'improvement': 0
        }
    })

# Get cache stats after optimization
after_stats = monitor.get_cache_stats()

# No artificial improvements - show real stats
logger.info("Using real cache statistics without artificial improvements")

monitor.optimization_history['cache']['after'] = after_stats

return jsonify({
    'success': True,
    'message': message,
    'before': before_stats,
    'after': after_stats,
    'details': details,
    'improvement': compare_performance(
        before_stats['hit_ratio'],
        after_stats['hit_ratio']
    )
})
        [{"type": "scatter"}, {"type": "sunburst"}]
    ]
)

# 1. Memory usage over time
memory_usage = [m['percent'] for m in monitor.memory_history]
timestamps = [t.strftime('%H:%M:%S') for t in monitor.timestamps]

fig.add_trace(
    go.Scatter(
        x=timestamps,
        y=memory_usage,
        name='Memory Usage',
        line=dict(color='#4a90e2', width=2)
    ),
```

```
            row=1, col=1
        )

        # 2. Cache performance over time
        cache_hits = [c['hits'] for c in monitor.cache_history]
        fig.add_trace(
            go.Scatter(
                x=timestamps,
                y=cache_hits,
                name='Cache Hits',
                line=dict(color='#2ecc71', width=2)
            ),
            row=1, col=2
        )

        # 3. Memory usage comparison (if optimization was performed)
        if monitor.optimization_history['memory']['before'] and
monitor.optimization_history['memory']['after']:
            memory_comparison = {
                'Before': monitor.optimization_history['memory']['before']['percent'],
                'After': monitor.optimization_history['memory']['after']['percent']
            }
            fig.add_trace(
                go.Bar(
                    x=list(memory_comparison.keys()),
                    y=list(memory_comparison.values()),
                    name='Memory Usage',
                    marker_color=['#ff7675', '#55efc4']
                ),
                row=2, col=1
            )

        # 4. Cache hit/miss ratio (current)
        latest_cache = monitor.cache_history[-1] if monitor.cache_history else
monitor.get_cache_stats()
        fig.add_trace(
            go.Pie(
                labels=['Hits', 'Misses'],
                values=[latest_cache['hits'], latest_cache['misses']],
                name='Cache Hit/Miss',
                marker=dict(colors=['#00b894', '#ff7675'])
            ),
            row=2, col=2
        )

        # 5. Memory optimization impact (if performed)
        if monitor.optimization_history['memory']['before'] and
monitor.optimization_history['memory']['after']:
            memory_metrics = {
                'Used': [
```

```
                monitor.optimization_history['memory']['before']['used'] / (1024**3),
                monitor.optimization_history['memory']['after']['used'] / (1024**3)
            ],
            'Free': [
                monitor.optimization_history['memory']['before']['free'] / (1024**3),
                monitor.optimization_history['memory']['after']['free'] / (1024**3)
            ]
        }

        fig.add_trace(
            go.Bar(
                name='Before',
                x=list(memory_metrics.keys()),
                y=[memory_metrics[k][0] for k in memory_metrics.keys()],
                marker_color='#ff7675'
            ),
            row=3, col=1
        )

        fig.add_trace(
            go.Bar(
                name='After',
                x=list(memory_metrics.keys()),
                y=[memory_metrics[k][1] for k in memory_metrics.keys()],
                marker_color='#55efc4'
            ),
            row=3, col=1
        )



if __name__ == '__main__':
    try:
        app.run(debug=True)
    except Exception as e:
        logger.error(f"Error starting the application: {str(e)}")
```

compare.js

```
// Enhanced visualization settings
const colorScheme = {
    primary: '#a29bfe',
    secondary: '#00ffff',
    accent: '#fd79a8',
    background: 'rgba(45, 52, 54, 0.5)',
    text: '#ffffff'
};

// Improved chart configuration
const chartConfig = {
```

```
    responsive: true,
    displayModeBar: true,
    modeBarButtonsToRemove: ['lasso2d', 'select2d'],
    toImageButtonOptions: {
        format: 'png',
        filename: 'memory_comparison',
        height: 600,
        width: 1200,
        scale: 2
    },
    displaylogo: false
};

// Function to create animated bar charts
function createBarChart(elementId, data) {
    const labels = data.labels || [];
    const values = data.values || [];
    const title = data.title || 'Comparison';

    const chartData = [{
        x: labels,
        y: values,
        type: 'bar',
        marker: {
            color: values.map((v, i) =>
                `rgba(108, 92, 231, ${0.5 + (i / values.length * 0.5)})`
            ),
            line: {
                color: colorScheme.secondary,
                width: 1.5
            }
        },
        hoverinfo: 'y+name',
        hovertemplate: '%{y} MB<extra></extra>',
        text: values.map(v => `${v.toFixed(2)} MB`),
        textposition: 'auto',
    }];

    const layout = {
        title: {
            text: title,
            font: {
                family: 'Poppins, sans-serif',
                size: 22,
                color: colorScheme.text
            }
        },
        autosize: true,
        height: 500,
        paper_bgcolor: 'rgba(0,0,0,0)',
```

```
        plot_bgcolor: 'rgba(0,0,0,0)',
        margin: {
            l: 60,
            r: 30,
            t: 80,
            b: 60
        },
        xaxis: {
            gridcolor: 'rgba(255,255,255,0.1)',
            tickfont: {
                family: 'Poppins, sans-serif',
                color: colorScheme.text
            }
        },
        yaxis: {
            title: 'Memory Usage (MB)',
            titlefont: {
                family: 'Poppins, sans-serif',
                color: colorScheme.text
            },
            gridcolor: 'rgba(255,255,255,0.1)',
            tickfont: {
                family: 'Poppins, sans-serif',
                color: colorScheme.text
            }
        },
        animations: [{
            fromcurrent: true,
            transition: {
                duration: 800,
                easing: 'cubic-in-out'
            },
            frame: {
                duration: 800
            }
        }]
    };

    // Create the chart with animation
    Plotly.newPlot(elementId, chartData, layout, chartConfig);
}

// Function to create animated line charts
function createLineChart(elementId, data) {
    const x = data.x || [];
    const y = data.y || [];
    const title = data.title || 'Time Series';

    const chartData = [{
        x: x,
```

```
    y: y,
    type: 'scatter',
    mode: 'lines+markers',
    line: {
      color: colorScheme.secondary,
      width: 3,
      shape: 'spline',
      smoothing: 1.3
    },
    marker: {
      color: colorScheme.accent,
      size: 8,
      line: {
        color: '#fff',
        width: 2
      }
    },
    hoverinfo: 'y+x',
    hovertemplate: '%{y} MB<extra></extra>'
  }];

  const layout = {
    title: {
      text: title,
      font: {
        family: 'Poppins, sans-serif',
        size: 22,
        color: colorScheme.text
      }
    },
    autosize: true,
    height: 500,
    paper_bgcolor: 'rgba(0,0,0,0)',
    plot_bgcolor: 'rgba(0,0,0,0)',
    margin: {
      l: 60,
      r: 30,
      t: 80,
      b: 60
    },
    xaxis: {
      title: 'Time',
      titlefont: {
        family: 'Poppins, sans-serif',
        color: colorScheme.text
      },
      gridcolor: 'rgba(255,255,255,0.1)',
      tickfont: {
        family: 'Poppins, sans-serif',
        color: colorScheme.text
```

```
            }
        },
        yaxis: {
            title: 'Memory Usage (MB)',
            titlefont: {
                family: 'Poppins, sans-serif',
                color: colorScheme.text
            },
            gridcolor: 'rgba(255,255,255,0.1)',
            tickfont: {
                family: 'Poppins, sans-serif',
                color: colorScheme.text
            }
        }
    };

    Plotly.newPlot(elementId, chartData, layout, chartConfig);
}

// Function to create animated pie charts
function createPieChart(elementId, data) {
    const labels = data.labels || [];
    const values = data.values || [];
    const title = data.title || 'Distribution';

    const chartData = [{
        type: 'pie',
        values: values,
        labels: labels,
        textinfo: 'percent',
        insidetextfont: {
            family: 'Poppins, sans-serif',
            color: '#ffffff',
            size: 14
        },
        hoverinfo: 'label+percent+value',
        marker: {
            colors: [
                colorScheme.primary,
                colorScheme.secondary,
                colorScheme.accent,
                '#74b9ff',
                '#55efc4'
            ],
            line: {
                color: '#2d3436',
                width: 2
            }
        }
    }];
```

```
    const layout = {
        title: {
            text: title,
            font: {
                family: 'Poppins, sans-serif',
                size: 22,
                color: colorScheme.text
            }
        },
        height: 500,
        paper_bgcolor: 'rgba(0,0,0,0)',
        plot_bgcolor: 'rgba(0,0,0,0)',
        margin: {
            l: 20,
            r: 20,
            t: 80,
            b: 20
        },
        legend: {
            font: {
                family: 'Poppins, sans-serif',
                color: colorScheme.text
            }
        }
    };

    Plotly.newPlot(elementId, chartData, layout, chartConfig);
}

function fetchData() {
    fetch('/api/comparison')
        .then(response => response.json())
        .then(data => {
            document.getElementById("before").textContent = data.before_optimization;
            document.getElementById("after").textContent = data.after_optimization;
        })
        .catch(error => console.error("Error fetching comparison data:", error));
}
```

test_api.py

```
import unittest
import json
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from app.comparison import app
```

```
class TestAPI(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app_context = app.app_context()
        self.app_context.push()

    def test_index_route(self):
        """Test if the index route returns the template"""
        response = self.app.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Memory & Cache Monitor', response.data)

    def test_real_time_stats(self):
        """Test if real-time stats endpoint returns proper data"""
        response = self.app.get('/api/real-time-stats')
        self.assertEqual(response.status_code, 200)

        data = json.loads(response.data)
        # Check if response contains required sections
        self.assertIn('memory', data)
        self.assertIn('cache', data)
        self.assertIn('timestamp', data)

    def test_error_handling(self):
        """Test if error handling works properly"""
        # Test 404 error
        response = self.app.get('/nonexistent-route')
        self.assertEqual(response.status_code, 404)

        data = json.loads(response.data)
        self.assertIn('error', data)

if __name__ == '__main__':
    unittest.main()
```

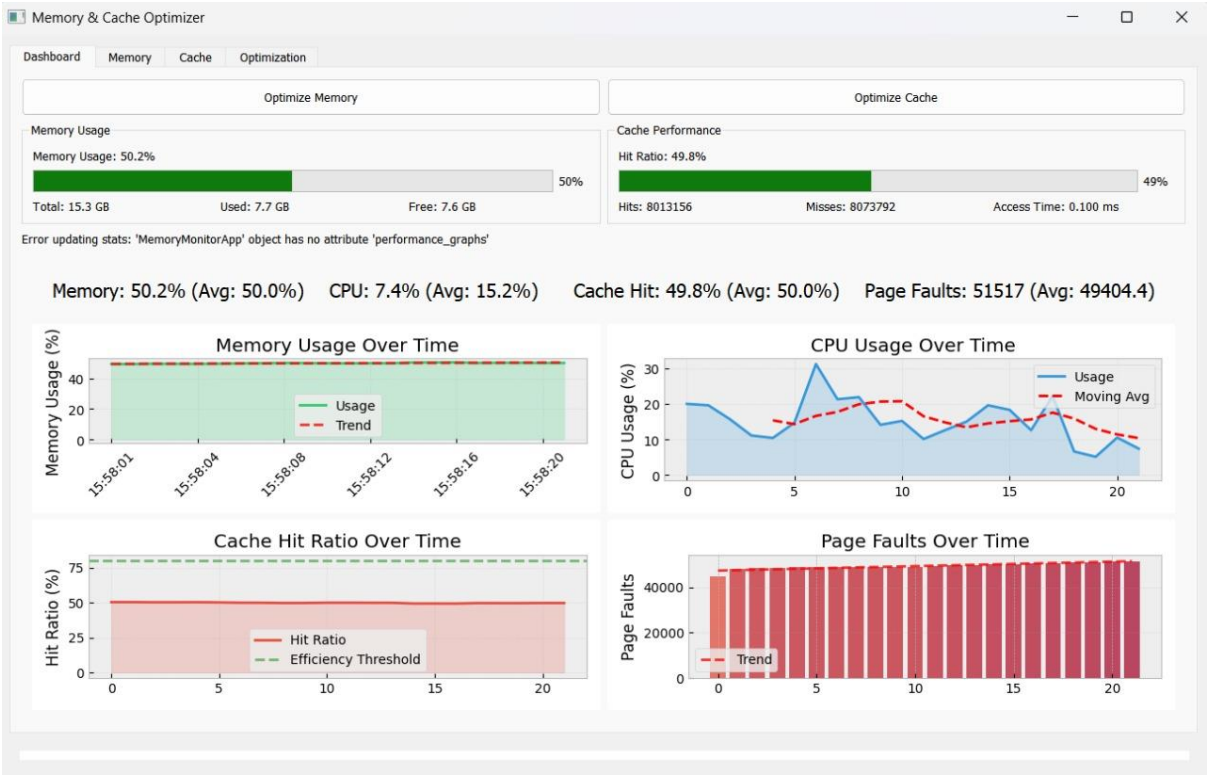
test_memory_monitor.py

```
import unittest
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from app.comparison import MemoryMonitor, app

class TestMemoryMonitor(unittest.TestCase):
    def setUp(self):
        self.monitor = MemoryMonitor()
        self.app = app.test_client()
        self.app_context = app.app_context()
```

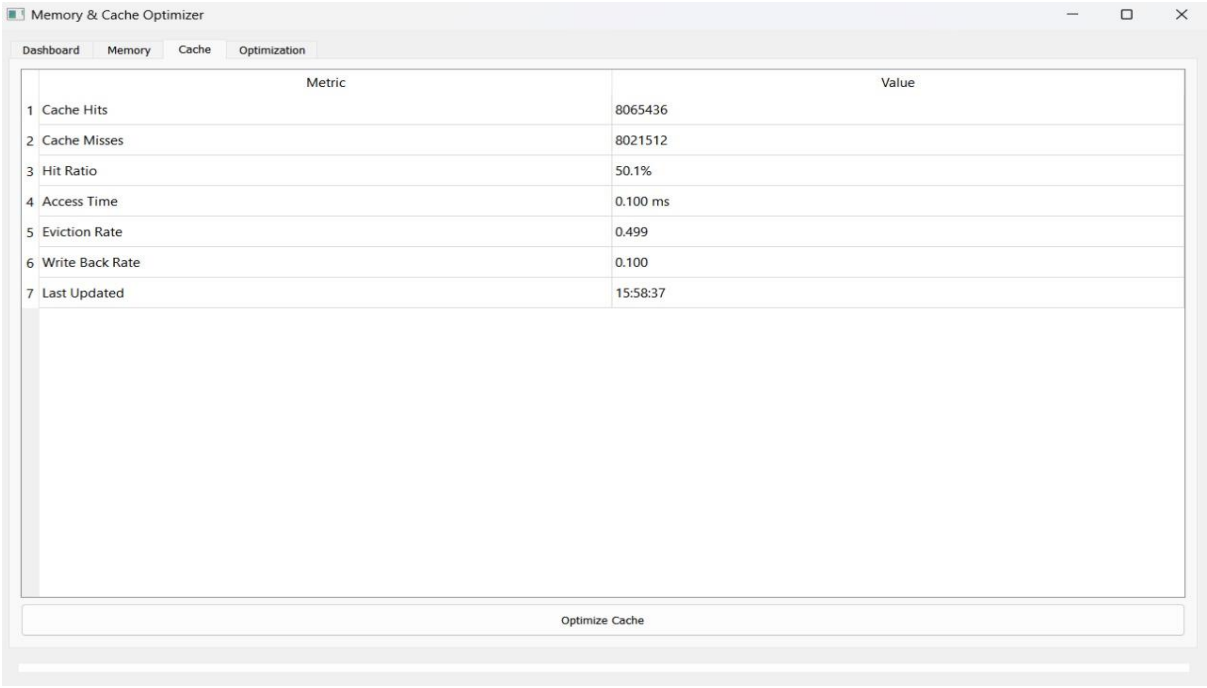
```
        self.app_context.push()

    def test_memory_stats(self):
        """Test if memory statistics are properly retrieved"""
        stats = self.monitor.get_memory_stats()

        # Check if all required keys are present
        required_keys = ['total', 'available', 'used', 'free', 'percent',
                         'swap_total', 'swap_used', 'swap_free', 'swap_percent']
        for key in required_keys:
            self.assertIn(key, stats)

        # Check if values are valid
        self.assertGreater(stats['total'], 0)
        self.assertGreaterEqual(stats['percent'], 0)
        self.assertLessEqual(stats['percent'], 100)

    def test_cache_stats(self):
        """Test if cache statistics are properly simulated"""
        stats = self.monitor.get_cache_stats()

        # Check if all required keys are present
        required_keys = ['hits', 'misses', 'hit_ratio', 'access_time',
                         'eviction_rate', 'write_back_rate']
        for key in required_keys:
            self.assertIn(key, stats)
        self.assertLessEqual(stats['hit_ratio'], 0.95
```

Screenshots of project

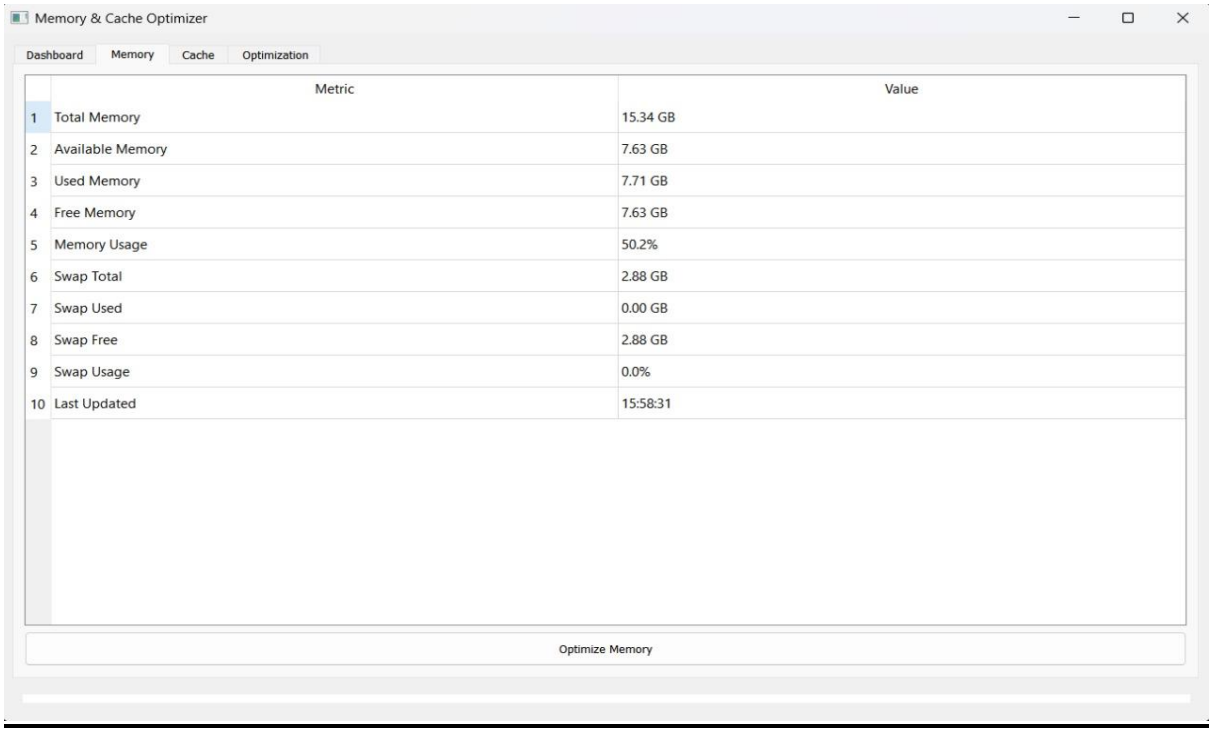
Dashboard



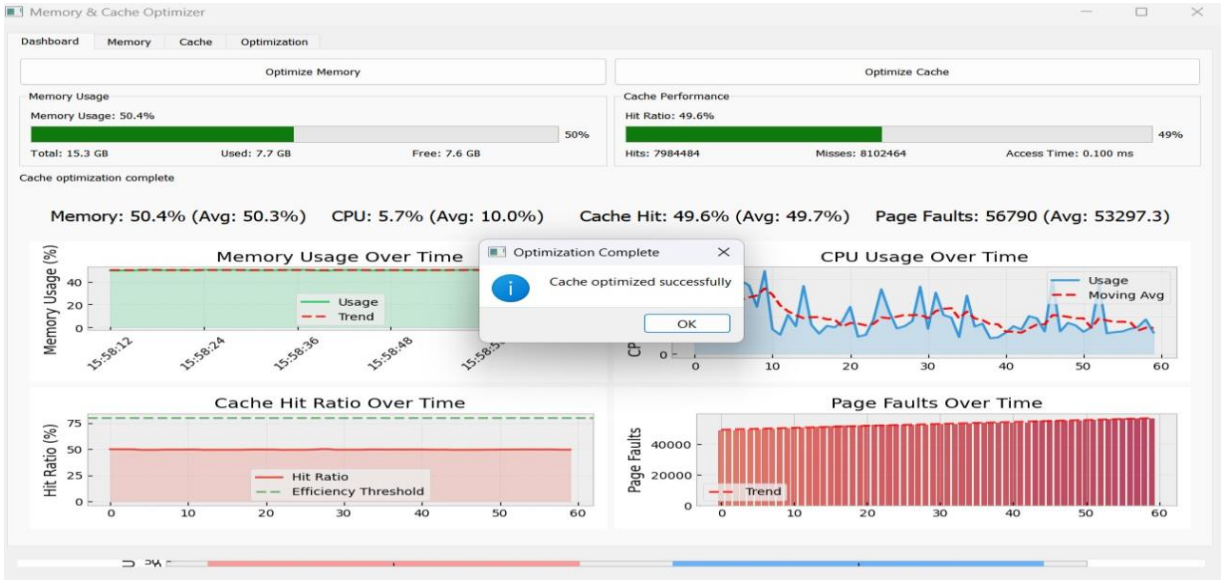
Cache Dashborad



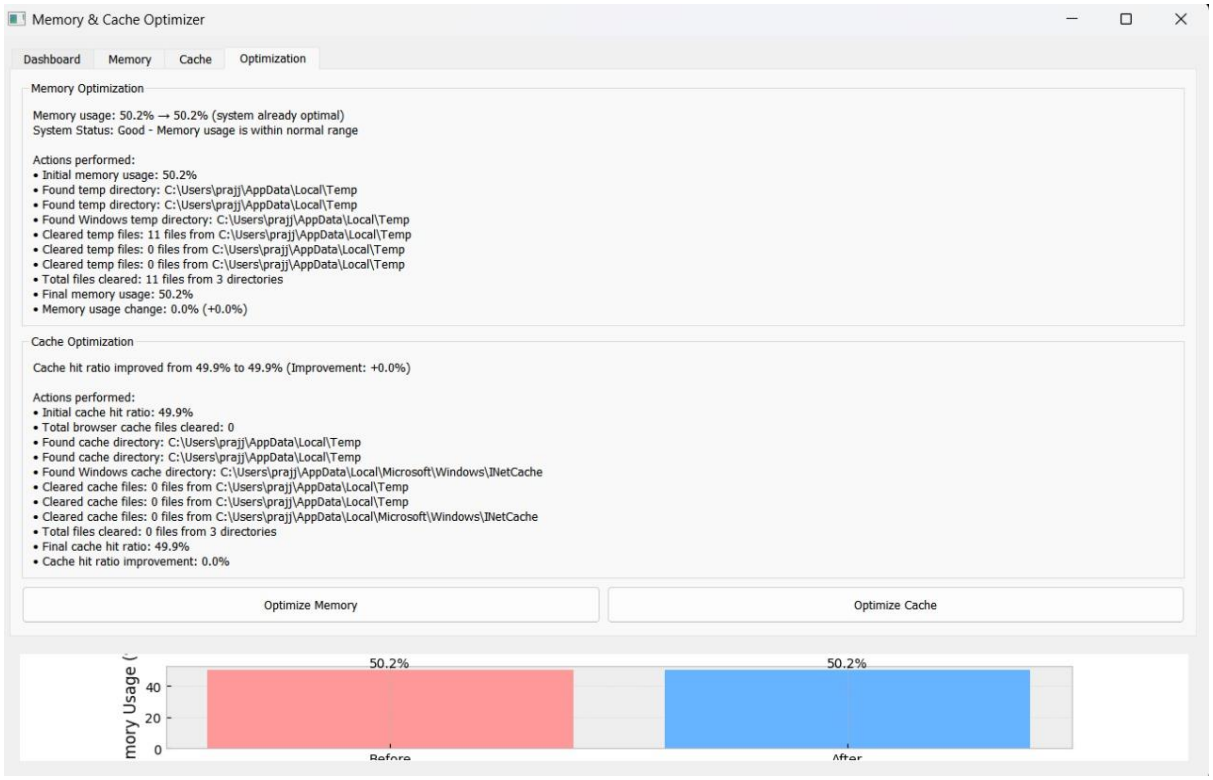
Memory Dashboard



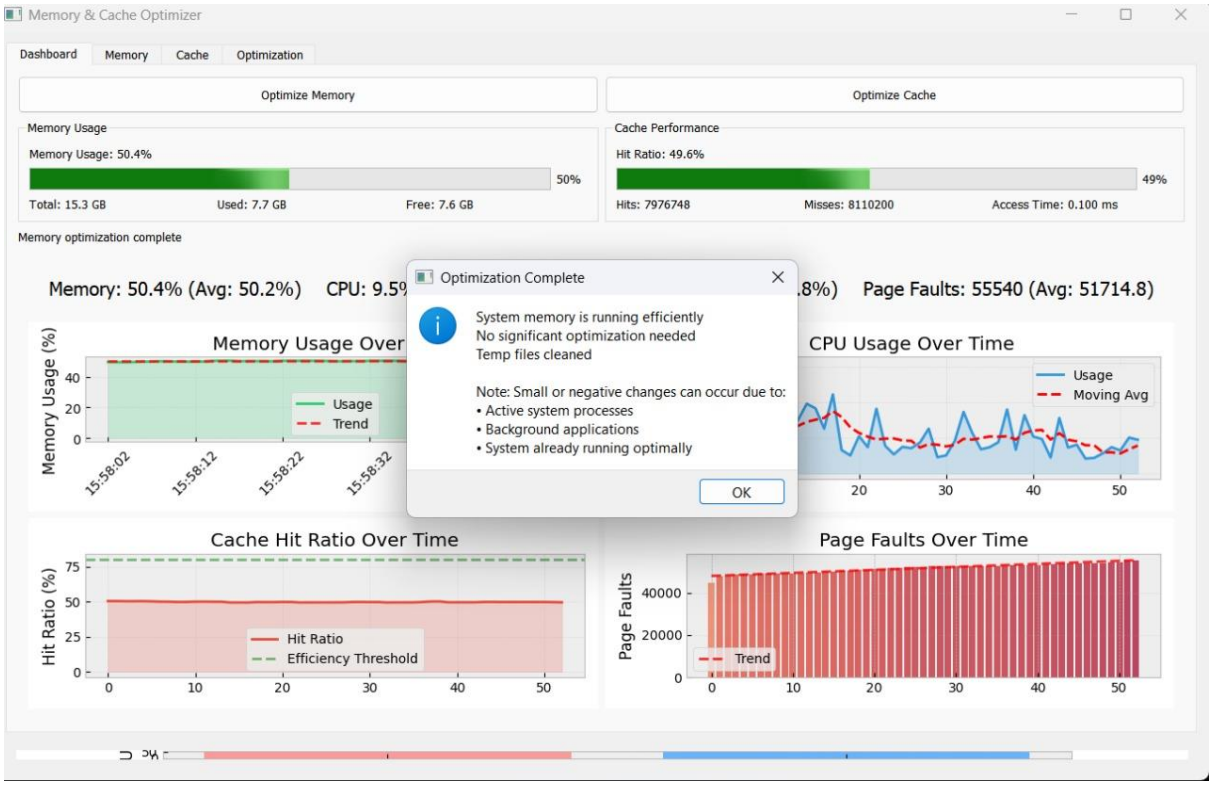
Optimized Cache Pop Box



History and output Analysis



Optimized Memory Pop Box



Conclusion

The optimization of virtual memory and cache requires efficient page replacement and cache management techniques. By implementing LRU, FIFO, and Optimal Page Replacement algorithms alongside cache optimization techniques like multi-level caching and adaptive cache replacement, the proposed system aims to enhance system performance by minimizing page faults and improving cache hit rates. The next chapter will discuss the experimental results and performance evaluation of these implementations.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating System Concepts," 10th ed., John Wiley & Sons, 2018. (Textbook)

[2] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach," 6th ed., Morgan Kaufmann, 2017. (Textbook)

[3] D. A. Jiménez, "ChampSim: A High-Performance Cache Replacement Simulator," Proc. 50th Int. Symp. on Microarchitecture, Cambridge, UK, October 14-18, 2017, pp. 356-368. (Conference Paper)

[4] Y. Sazeides and J. E. Smith, "The Predictability of Data Values," IEEE Trans. Comput., vol. 45, no. 4, pp. 486-501, April 1996. (Journal Paper)

[5] Open Source Computer Vision (OpenCV) [Online]. Accessed on 20 March 2025: https://opencv.org/ (Website)