IMPLEMENTATION AND ANALYSIS OF A VIRTUAL PRIVATE SERVER FOR A ONE HEALTH SURVEILLANCE SYSTEM

BY

PETER EMEKA ORJI

ENG2003831

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF BENIN

PROJECT SUPERVISOR

DR. E OLAYE

OCTOBER 2025

IMPLEMENTATION AND ANALYSIS OF A VIRTUAL PRIVATE SERVER FOR A ONE HEALTH SURVEILLANCE SYSTEM

BY

PETER EMEKA ORJI

ENG2003831

A PROJECT SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING, FACULTY OF ENGINEERING, UNIVERSITY OF BENIN

BENIN CITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF ENGINEERING (B.ENG) DEGREE IN COMPUTER ENGINEERING

OCTOBER 2025

CERTIFICATION

This is to certify that this project **IMPLEMENTATION AND ANALYSIS OF A VIRTUAL PRIVATE SERVER FOR A ONE HEALTH SURVEILLANCE SYSTEM** for the award of B.Eng. conducted and duly presented by Emeka Peter Orji (ENG2003831) of The Department of Computer Engineering, Faculty of Engineering, University of Benin, Benin City has been accepted.

_____ _____

Engr. Dr. Edoghogho. Olaye **Date**

Project Supervisor

_____ _____

Engr. Dr. Isi Edeoghon **Date**

Head of Department

DEDICATION

I dedicate this work to God Almighty, my source of strength and wisdom, whose unwavering grace has sustained me throughout this academic endeavor.

ACKNOWLEDEMENT

First and foremost, I extend my profound gratitude to God Almighty for the divine wisdom, grace, and strength that guided and sustained me throughout the duration of my academic program and this research endeavor.

I wish to express my sincere and deepest appreciation to my project supervisor, Engr. Dr. E. Olaye, for his invaluable mentorship, scholarly guidance, and steadfast support during the execution of this project. His insightful critiques and constructive feedback were instrumental in shaping the direction and quality of this work.

I am also indebted to the faculty and staff of the Department of Computer Engineering for providing a robust academic foundation and a conducive environment for learning, which has been pivotal to my intellectual growth.

To my parents, Mr. and Mrs. Orji, I offer my heartfelt thanks for their unwavering love, immense sacrifices, and constant encouragement, which have been the bedrock of my academic pursuits and achievements.

I would also like to acknowledge the support of my colleagues and friends—Daniel, Jide, Jennifer, and Success—for their stimulating discussions, camaraderie, and moral support throughout this process.

Lastly, I extend my gratitude to all individuals who have contributed, either directly or indirectly, to the successful completion of this project. Your support is highly valued.

.

TABLE OF CONTENT

LIST OF TABLES

LIST OF ACRONYMS

VPS: Virtual Private Server

KVM: Kernel-based Virtual Machine

AI: Artificial Intelligence

B.ENG: Bachelor of Engineering

API: Application Programming Interface

CPU: Central Processing Unit

RAM: Random Access Memory

ROM: Read-Only Memory (Note: This is likely a typo in the document and should be "Storage" or "Disk Space", as ROM is not a standard metric for server storage)

NVMe: Non-Volatile Memory Express (a type of SSD)

SSD: Solid State Drive

SSH: Secure Shell

UFW: Uncomplicated Firewall

LLM: Large Language Model

CSV: Comma-Separated Values

OS: Operating System

ABSTRACT

This project focuses on the implementation and analysis of a Virtual Private Server (VPS) for a Digital One Health Surveillance System. The aim of the work is to ensure a stable and reliable infrastructure that can support both the core system components and integrated AI health models without interruptions. The One Health approach requires a platform that can process data from human, animal, and environmental health sources in real time, and this can only be achieved with a server that offers improved performance, security, and scalability.

In this project, a Hostinger KVM2 VPS was deployed and configured with EasyPanel, a PostgreSQL database, and both the frontend (React) and backend (Node.js) services. AI health models were installed locally using Ollama, and a benchmarking script was developed to measure and compare the performance of different models. These tests helped verify that the upgraded VPS could run AI model inference alongside the application without system crashes or database shutdowns, which was a major limitation of the previous VPS setup.

The results showed a significant improvement in system responsiveness, model execution stability, and overall uptime. All core services were able to run simultaneously without performance conflicts. Health models responded faster, and the system was able to maintain continuous surveillance analysis in real time. This confirms that the KVM2 upgrade provides a more reliable and scalable foundation for future AI-driven features in the One Health Surveillance System.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

One Health is an interdisciplinary approach that highlights the interconnectedness of human, animal, and environmental health. It fosters collaboration across sectors, including public health, veterinary medicine, and ecological sciences, to address global health challenges. The core principle is that the health of humans, animals, and ecosystems is interdependent, necessitating a holistic approach for disease prevention and control. This project proposes the analysis and implementation of a Virtual Private Server (VPS) for a One Health Surveillance System, utilizing the Hostinger (Kernel-based Virtual Machine)

KVM 2 plan. This enhanced server configuration will offer sufficient resources to support the simultaneous operation of all system components, including the PostgreSQL database, React frontend, Node.js backend, and the Olama AI platform. By upgrading the infrastructure, the system's overall responsiveness will be significantly improved, enabling more efficient and uninterrupted processing of surveillance data.

1.2 PROBLEM STATEMENT

The current VPS infrastructure supporting the Digital One Health Surveillance System faces critical resource constraints that prevent the full realization of its integrated surveillance capabilities. Specifically, the 4 GB RAM and 50GB ROM limitation create operational conflicts when attempting to run AI-powered features alongside core system components. This constraint forces system administrators to temporarily shut down essential services such as the backend database connections, creating gaps in surveillance coverage and undermining the system's reliability.

Consequently, there is an urgent need to implement a virtual private server with higher capacity to ensure the system's prediction is always reliable. This project, therefore, is an approach to reliable One health surveillance data.

1.3 AIM AND OBJECTIVES

The primary aim of this project is to implement and analyze an enhanced Virtual Private Server configuration for Digital One Health Surveillance. Utilizing the Kernel-based Virtual Machine (KVM) 2, the project will ensure a System that enables seamless integration of AI capabilities while maintaining the full operational capacity of all core system components.

Specific Objectives:

I. Infrastructure Analysis: Conduct a comprehensive analysis of current VPS performance, resource utilization, and limitation assessment to establish a baseline and identify specific upgrade requirements.
II. VPS Implementation: Deployment of the enhanced KVM2, including system migration, configuration, and integration testing.
III. Evaluation: Monitoring system to evaluate the enhanced KVM2 performance, Validation of Ollama AI model integration, and system stability under various load conditions.
IV. Recommendations: Offer recommendations for upgrading the KVM2 features based on evaluation results, aiming to optimize the system's performance and user experience.

1.4 SCOPE OF STUDY

I. Technical Analysis: Comprehensive evaluation of current VPS performance metrics, resource utilization patterns, and system bottlenecks. This includes database performance analysis and resource consumption monitoring.
II. VPS upgrade: Upgrade of current VPS includes data migration from the current infrastructure, system reconfiguration, and backup procedures.
III. AI models Integration Capabilities: Testing and validation of Ollama AI model integration and local models.
IV. Documentation: providing comprehensive documentation.

The study will not cover the broader aspects of One Health beyond the scope of VPS implementation and analysis, nor will it address other aspects of One Health beyond the scope of the Digital One Health Surveillance System being developed.

1.5 RELEVANCE OF PROJECT

This study is particularly relevant due to its focus on VPS implementation to enhance the reliability of health surveillance through the implementation of a KVM2 to a digital One Health surveillance system. By upgrading the RAM, Bandwidth, and CPU cores. The system aims to improve the accuracy, reliability, and integration of AI models. These enhancements will enable simultaneous operation of all system Components, including AI-powered features, without performance degradation or service interruptions.

1.6 LIMITATIONS OF EXISTING SYSTEMS

Various drawbacks remain in temperature and humidity monitoring systems despite advances that have been made:

The research conducted by Sonawane et al. (2019) concentrated on HVAC optimization thus it did not include broader capabilities such as security features and multi-room surveillance. The DHT11 sensors operated by Guo et al. (2016) showcased restricted measuring ability because they struggled with environmental conditions which produced measurement imprecision. Shukur (2021) utilized smartphone applications which needed user involvement instead of self-operating oversight.

The cloud-based monitoring system created by Anik et al. (2021) faced security issues and semantic and verbalization errors. The current system requires an upgrade because it lacks precise sensor performance and automation capabilities and multi-room control which this project seeks to solve.

CHAPTER TWO

LITERATURE REVIEW

2.1 BACKGROUND OF STUDY

Digital health systems are rapidly evolving, and the One Health approach addressing the interconnectedness of human, animal, and environmental health demands a robust, secure, and scalable data infrastructure. A Virtual Private Server (VPS) offers a viable solution by providing dedicated resources, enhanced control, and improved security compared to traditional shared hosting. In the digital era, deploying a VPS for One Health surveillance presents opportunities for real-time data integration, secure storage of sensitive health records, and the implementation of advanced analytical tools.

Several researchers have explored the application of VPS and cloud computing within the healthcare sector. (Rodrigues et al.,2013) examined the challenges of storing sensitive electronic health records (EHRs) in cloud environments. Their work emphasizes that healthcare providers and cloud vendors must collaborate closely to address stringent security and privacy requirements. Although their study does not focus on a specific implementation, it underlines the importance of building trust. A critical factor when selecting a VPS solution for health data. (Adelmeyer et al., 2019) conducted an online experiment comparing cloud-based and on-premises storage solutions for personal health records.

Their research revealed that users perceive cloud-based records as more vulnerable regarding security, privacy, control, and trust, particularly following simulated data breach scenarios. This finding is significant for One Health surveillance systems, where maintaining user confidence and protecting sensitive information are essential. Similarly, Sakr et al. presented a secure client-side framework that allows healthcare facilities to retain control over their data, reinforcing the notion that control over security measures is paramount when storing health data in the cloud. (Nathan et al., 2013) contributed to the field by demonstrating a prototype virtual private cloud infrastructure. His work, which leverages wopen-source tools such as Mirth Connect, Apache, PHP, and MySQL, illustrates how a VPS can be configured within platforms like Amazon's Virtual Private Cloud (VPC) to support HIPAA-compliant storage and usage of sensitive health data. This prototype underscores the potential of VPS solutions to meet regulatory requirements

while providing the scalability needed for a multi-faceted surveillance system. In a study by (Rashid et al., 2019), a model was proposed to enhance the privacy of health data stored on the cloud. Their approach combines IP-based and geo-location validation with constant key length encryption to secure data regardless of the user type. By integrating these security measures, the model addresses the challenge of preserving data privacy in environments where sensitive medical information is processed and stored. Such strategies could be directly beneficial when configuring a VPS dedicated to One Health surveillance, where robust access validation is critical.

Despite the advances discussed in these studies, several gaps remain. First, although (Rodrigues et al. 2013) and (Regola et al., 2013) provide insights into the need for secure cloud storage, neither investigates the real-world implementation and performance of VPS-based systems specifically designed for One Health surveillance. In addition,

(Adelmeyer et al. 2019) focus primarily on user perceptions via online experiments without validating their findings in practical, large-scale environments. While Sakr et al. propose a theoretical framework for privacy control and (Rashid et al., 2019) develop a technical model for enhanced security, neither work directly addresses the integration of these systems with health surveillance platforms that require both real-time data processing and a robust, scalable infrastructure. This current project aims to bridge these gaps by implementing and analyzing a VPS tailored for One Health surveillance. In doing so, it will incorporate proven security measures (as demonstrated by Rashid et al. and Sakr et al.), adopt best practices for data privacy (highlighted by Rodrigues et al. and Adelmeyer et al.), and validate a practical VPS configuration (as shown by Regola) that supports real-world healthcare applications.

2.2 META-ANALYSIS TABLE FOR IMPLEMENTATION AND ANALYSIS OF A VIRTUAL PRIVATE SERVER FOR A ONE HEALTH SURVEILLANCE SYSTEM

Author(s)	Year	Title	Methodology	Result	Research Gap

Olaye et al.	2025	Personalized Governance Strategy for Patient Data in a Digital One Health Surveillance System	Development and deployment of an Android app (PPDSS) using Flutter. It captures text from physical health records using a device camera, applies data masking (obfuscation), then processes text with AI-powered OCR.	Ensure sensitive patient data (name, age, address, phone) is eliminated before reaching machine learning models or local storage, enhancing confidentiality from data collection to analysis.	Limited to the Android platform (future scalability mentioned but not demonstrated).
Joel et al.	2013	Analysis of the Security and Privacy Requirements of Cloud-Based Electronic Health Records Systems	Review of bibliographic material from Medline sources - Direct contact with Cloud service providers - Review of published papers and research on security and privacy issues in Cloud-based EHR systems	The main risk to EHRs in the Cloud is external, necessitating robust security measures against external threats. - Automated monitoring tools and service availability are	The Cloud computing paradigm is still under development. Limited current capabilities and maturity. Need for future development and the availability of more services and apps. Difficulty in selecting a secure

				crucial for ensuring the reliability and security of Cloud services.	cloud provider due to varying security terms.Limited current knowledge or practices regarding privacy and confidentiality
Adelmeyer et al	2019	Security and Privacy of Personal Health Records in Cloud Computing Environments – An Experimental Exploration of the Impact of Storage Solutions and Data Breaches	Online experiment comparing cloud-based and on-premises data storage solutions for personal health records, examining the impact of data breaches on user perceptions)	-Cloud-based personal health records face concerns regarding perceived security, privacy, control, and trust among end-users. -After a data breach, there are no significant differences in perceived security, privacy, control, and trust between cloudbased solutions and	The study was limited to an online experiment, rather than a real-world implementation, which could limit the generalizability of the findings.

				private on-premises data centers.	
A. Sakr, E. et al	2020	A secure client-side framework for protecting the privacy of health data stored on the cloud	The work presents a secure client-side framework for storing data on the cloud, allowing customers to maintain control over the security and privacy of their data.	The study presents a secure client-side framework for storing health data on the cloud while keeping the customer in control of security and privacy.	- Concerns about security breaches in cloud computing - Insufficiency of relying solely on cloud
Nathan et al.	2025	Storing and Using Health Data in a Virtual Private Cloud	- Developed a prototype infrastructure in Amazon's Virtual Private Cloud (VPC) for HIPAA-compliant health data exchange. - Set up a server with Mirth Connect, an opensource tool	-The paper presents a prototype infrastructure using Amazon's Virtual Private Cloud to create a HIPAA - compliant environment for health data exchange.	Dynamic update mirrors for Amazon Linux complicate security group configurations. - Inability to access denied events by security groups or network access control lists. - Lack of clarity in HIPAA standards.

			for connecting to health information systems. - Conducted a pilot run with anonymized and scrubbed patient data from a health exchange using HL7 message format over an SSL connection	- The authors demonstrate the steps to set up a Mirth Connect server within the VPC, ensuring HIPAA compliance for real-time health data integration. A model using IP based and location based validation to preserve privacy, constant key length encryption for data security	- Potential for side-channel attacks due to virtualization. - Reliability of internet connections for cloud-based Server.
Yazan et al.	2019	Hybrid Cryptographic Access Control for Cloud-Based EHR Systems	- Cryptographic rolebased technique for session key distribution using Kerberos protocol	- The centralization of data in cloud computing raises significant security and privacy	- Current solutions only address a subset of security concerns. - Solutions fail to balance all

			- Location- and biometrics-based authentication - Wavelet-based steganographic technique for embedding EHR data in ECG signals - Rolebased access control model with role hierarchy - Location validation by health authority through domain server requests - Security analysis of communication channels	concerns for individuals and healthcare providers. - Current solutions only address a subset of these concerns, lacking a holistic approach. - Security is a primary issue hindering the adoption of cloud technology in healthcare.	security requirements. Gains in one security dimension often result in losses in another. Need for a holistic solution that balances all security requirements. Security challenges in emerging fog and edge technologies are significant. technologies

2.3 RESEARCH GAP

Despite the advances discussed in these studies, several gaps remain. First, although Rodrigues et al. (2013) and Regola (2013) provide insights into the need for secure cloud storage, neither investigates the real-world implementation and performance of VPS-based systems specifically designed for One Health surveillance. In addition, Adelmeyer et al. (2019) focus primarily on user perceptions via online experiments without validating their findings in practical, large-scale

environments. While Sakr et al. propose a theoretical framework for privacy control and Rashid et al. develop a technical model for enhanced security, neither work directly addresses the integration of these systems with health surveillance platforms that require both real-time data processing and a robust, scalable infrastructure.

This current project aims to bridge these gaps by implementing and analyzing a VPS tailored for a Digital One Health surveillance system. In doing so, it will incorporate proven security measures as demonstrated by Rashid et al. and Sakr et al., adopt best practices for data privacy (highlighted by Rodrigues et al. and Adelmeyer et al.) and validate a practical VPS configuration (as shown by Regola) that supports real-world healthcare applications.

.

CHAPTER THREE

METHODOLOGY

3.1 INTRODUCTION

This chapter presents the systematic methodology employed in the implementation and analysis of a Virtual Private Server for the Digital One Health Surveillance System. The approach consisted of five structured phases designed to ensure seamless transition from infrastructure planning to AI model deployment and automated performance monitoring.

3.2 VPS IMPLEMENTATION

The project began by selecting a suitable VPS plan capable of supporting the simultaneous operation of all system components. The Hostinger KVM 2 VPS plan was chosen based on its technical specifications:

i. Processing Power: 2 vCPU cores for parallel processing

ii. Memory: 8GB RAM to support concurrent AI and application operations

iii. Storage: 100GB NVMe SSD for high-speed data access

iv. Bandwidth: 8TB for uninterrupted data transmission

v. Operating System: Full Linux OS compatibility

This configuration addresses the resource constraints identified in the current system, where 4GB RAM limitations prevent simultaneous operation of AI models and core system components.

3.3 IMPLEMENTATION PHASES

3.3.1 PHASE 1: VPS ACQUISITION AND ENVIRONMENT SETUP

Following the plan selection and purchase, the VPS was provisioned and accessed via Hostinger's Easy Panel interface. This panel facilitated initial configurations including:

I. Ubuntu 22.04 LTS installation (selected for its Long Term Support and extensive package ecosystem)

II. SSH key setup for secure remote access

3.3.2 PHASE 2: SYSTEM CONFIGURATION AND OLLAMA INSTALLATION

The VPS environment was hardened for security using standard Linux practices. This included disabling root login, enforcing SSH key authentication, and installing UFW (Uncomplicated Firewall).

3.3.3 AI MODEL DEPLOYMENT

Multiple health-focused AI models were deployed to support diverse surveillance requirements:

Table 3.1: LLMs and their Functions

Model Name	Primary Function	Size
llama3	General language understanding	4.7GB
hippomistral	Clinical decision support	4.1GB
openbiollm	Biomedical literature analysis	4.7GB
cogito	Epidemiological pattern detection	2.2GB
gemma3-translator	Multilingual health communication	3.3GB

3.3.4 PHASE 3: MANUAL PERFORMANCE BENCHMARKING

To assess system performance, each model was executed individually while monitoring CPU, RAM, and disk usage. The benchmarking protocol involved:

I. Capturing baseline system metrics before model execution
II. Running each model with a standardized test prompt
III. Monitoring real-time resource utilization during execution

IV. Recording response time and peak resource consumption

V. Analyzing resource recovery after execution

The standardized test prompt used across all models was: "What is One Health?" This prompt was selected for its relevance to the surveillance system domain and moderate complexity, requiring substantive cognitive processing.

Table 3.2: Manual Benchmarking Results

Model Name	CPU Usage (%)	RAM Usage (%)	Disk Usage (%)	Duration (sec)	Model Size	Implementation Status
clinical-br-mistral-7b-0.2	0.5 → 88.0	22.9 → 22.9	15.2 → 28.5	584	14.0GB	Not Implemented*
meditron	0.5 → 48.0	22.9 → 23.1	15.2 → 18.9	44	3.8GB	Not Implemented*
medichat-llama3	1.0 → 14.7	23.1 → 23.3	15.2 → 20.0	54	4.9GB	Not Implemented*
llama3	0.5 → 100.0	20.5 → 91.0	19.7 → —	103	4.7GB	Implemented
hippomistral	0.0 → 2.5	83.8 → 91.0	15.2 → 19.2	140	4.1GB	Implemented
openbiollm	1.0 → 5.0	20.8 → 91.2	15.2 → 19.7	44	4.7GB	Implemented

Model Name	CPU Usage (%)	RAM Usage (%)	Disk Usage (%)	Duration (sec)	Model Size	Implementation Status
cogito	1.5 → 54.0	21.0 → 59.3	17.0 → 19.3	26	2.2GB	Implemented
antonio-gemma3-smart-q4	0.5 → 21.9	23.3 → 36.9	15.2 → 15.9	21	0.7GB	Implemented
aquif-3.0-preview-8b	0.5 → 6.0	23.6 → 36.9	15.2 → 15.9	50	4.9GB	Implemented
gemma3-translator	0.5 → 4.5	22.8 → 38.3	16.0 → 15.9	14	3.3GB	Implemented

3.3.5 PHASE 4: AUTOMATION OF MONITORING SYSTEM

To eliminate manual overhead and ensure consistent performance tracking, a Python-based monitoring script was developed. The automation objectives included:

I. Consistent benchmarking protocols across all models
II. Elimination of human measurement error
III. Structured data collection for statistical analysis
IV. Scalable evaluation framework for future model additions
V. Longitudinal performance data tracking

3.3.5 PHASE 4: AUTOMATION OF MONITORING SYSTEM

The monitoring script (Monitor_Ollama_VPS.py) was developed using the following Python libraries:

I. psutil: System resource monitoring (CPU, RAM, disk)
II. subprocess: Docker container command execution
III. csv: Structured data output
IV. datetime: Timestamp generation
V. time: Execution timing measurement

3.3.5.1 Script Architecture

The monitoring script (Monitor_Ollama_VPS.py) was developed using the following Python libraries:

I. psutil: System resource monitoring (CPU, RAM, disk)
II. subprocess: Docker container command execution
III. csv: Structured data output
IV. datetime: Timestamp generation
V. time: Execution timing measurement

The script operates through a systematic workflow:

I. Initialize system baseline metrics collection
II. Iterate through predefined model list
III. Execute Docker command for model invocation
IV. Capture real-time resource utilization during execution
V. Calcute performance differentials (baseline vs. peak)
VI. Log comprehensive metrics to CSV format
VII. Generate execution summary report

The script implements several key features:

I. Modular Function Design: Distinct functions for metric collection, model execution, data persistence, and orchestration

II. Comprehensive Error Handling: Exception handling for Docker communication failures, model execution errors, and timeout scenarios
III. Timeout Protection: 600-second timeout prevents indefinite hanging on non-responsive models
IV. System Stabilization: 10-second pause between model executions allows resources to normalize
V. Detailed Logging: Real-time console output for benchmark progress and execution status
VI. Structured Data Output: CSV format with timestamps, resource metrics, duration, and error messages

3.3.6 PHASE 5: SCRIPT DEPLOYMENT AND EXECUTION

The automation script was deployed to the VPS following this procedures for deploying the automation script to the VPS(ssh root@31.97.197.111)

Step 1: File Transfer

```bash
scp Monitor_Ollama_VPS.py username@server_ip:~/
```

Step 2: Directory Organization

```bash
ssh username@server_ip
mkdir -p ~/ollama-Monitor
mv Monitor_Ollama_VPS.py ~/ollama-Monitor/
cd ~/ollama-Monitor
```

Step 3: Dependency Installation

```bash
pip3 install psutil
```

Step 4: Script Execution

```bash
python3 Monitor_Ollama_VPS.py
```

Upon execution, the script iterates through predefined models, logs performance metrics, and stores results in results.csv. The CSV output includes:

- Timestamp
- Model Name
- CPU Usage (Baseline and Peak %)
- RAM Usage (Baseline and Peak %)
- Disk Usage (Baseline and Peak %)
- Execution Duration (seconds)
- Response Status (Success/Failure/Timeout)
- Error Messages (if applicable)

Step 5: Results Retrieval

```bash
scp username@server_ip:~/ollama-Monitor/results.csv ./local_directory/
```

This structured output facilitates quantitative analysis, visualization, and comparative performance evaluation.

3.4 DATA COLLECTION AND ANALYSIS

Performance data was collected through both manual observation (Phase 3) and automated logging (Phase 5). This dual approach enabled validation of automated measurements against manually observed behaviors. The collected data supports multiple analytical dimensions including resource efficiency analysis, scalability assessment, optimization opportunity identification, and cost-benefit evaluation.

3.5 SOFTWARE DEVELOPMENT AND IMPLEMENTATION

Several limitations warrant acknowledgment:

1. **No direct access to the VPS CMD:** VPS performance could only be measured from data available and displayed on Easy panel.

2. **Network Dependency:** The Entire Monitoring process and Logs generation was dependent on the network provider.

3. **Model Availability:** Limited to open-source models compatible with Ollama framework

4. **Prompt Standardization:** A Single test prompt may not capture the full model capability spectrum

3.6 SUMMARY

This chapter presented a five-phase methodology for VPS implementation and analysis within the Digital One Health Surveillance System. The systematic approach integrated infrastructure deployment, security hardening, AI platform installation, performance benchmarking, and automated monitoring. This methodology ensures reproducibility and provides a foundation for result analysis in subsequent chapters.

CHAPTER 4

SYSTEM IMPLEMENTATION AND RESULTS

4.1 INTRODUCTION

This chapter presents the results obtained from the implementation and analysis of the Virtual Private Server for the Digital One Health Surveillance System. The successful transition to the Hostinger KVM 2 plan has yielded substantial improvements in system performance, reliability, and AI model integration capabilities. The results are presented in comparison with the previous infrastructure to demonstrate the tangible benefits of the VPS upgrade.

4.2 SYSTEM IMPLEMENTATION OUTCOMES

4.2.1 VPS UPGRADE AND MIGRATION

The system was successfully migrated from the previous infrastructure (4GB RAM, 50GB storage) to the Hostinger KVM 2 VPS, featuring:

i. 2 vCPU cores (versus single core previously)
ii. 8GB RAM (doubled from 4GB)
iii. 100GB NVMe storage (doubled from 50GB)
iv. 8TB bandwidth (substantially increased from previous allocation)

All core system components were successfully deployed and configured on the new infrastructure:

i. PostgreSQL Database: Now operates with dedicated memory allocation, eliminating previous connection interruptions
ii. Node.js Backend: Runs continuously without requiring shutdown for AI operations
iii. React Frontend: Maintains consistent availability with improved response times
iv. Ollama AI Platform: Successfully integrated with full operational capacity

Table 4.1: Infrastructure Comparison

Specification	Previous VPS	Current VPS (KVM 2)	Improvement
vCPU Cores	1	2	100% increase
RAM	4GB	8GB	100% increase
Storage	50GB	100GB NVMe	100% increase
Bandwidth	Limited	8TB	Substantial increase
Concurrent AI Operations	No	Yes	New capability

4.2.2 AI MODEL INTEGRATION SUCCESS

The primary objective of enabling the simultaneous operation of AI models alongside core system components was successfully achieved. Seven specialized health-focused AI models were deployed within the Ollama container:

1. llama3 (4.7GB) - General language understanding for multi-domain health queries
2. hippomistral (4.1GB) - Clinical decision support for medical diagnostics
3. openbiollm (4.7GB) - Biomedical literature analysis for research integration
4. cogito (2.2GB) - Epidemiological pattern detection for outbreak surveillance
5. antonio-gemma3-smart-q4 (0.7GB) - Lightweight query processing
6. aquif-3.0-preview-8b (4.9GB) - Advanced health data analysis
7. gemma3-translator (3.3GB) – Multilingual health communication support

These models now operate concurrently with the PostgreSQL database, Node.js backend, and React frontend without resource conflicts, this wasn’t possible under the previous infrastructure (KVM1).

4.2.3 PERFORMANCE BENCHMARKING RESULTS

Comprehensive performance benchmarking was conducted to evaluate system behavior under AI workload. The results demonstrate the VPS's capacity to handle diverse AI models with varying resource requirements.

Table 4.2: AI Model Performance Metrics.

Model	CPU Usage (%)	RAM Usage (%)	Execution Time (sec)	Status
llama3	0.5 → 100	20.5 → 91.0	103	Operational
hippomistral	0.0 → 2.5	83.8 → 91.0	140	Operational
openbiollm	1.0 → 5.0	20.8 → 91.2	44	Operational
cogito	1.5 → 54.0	21.0 → 59.3	26	Operational
antonio-gemma3-smart-q4	0.5 → 21.9	23.3 → 36.9	21	Operational
aquif-3.0-preview-8b	0.5 → 6.0	23.6 → 36.9	50	Operational
gemma3-translator	0.5 → 4.5	22.8 → 38.3	14	Operational

4.2.3 AUTOMATED MONITORING SYSTEM DEPLOYMENT

The Python-based monitoring script (Monitor_Ollama_VPS.py) was successfully deployed and integrated into the VPS infrastructure. The automation system provides:

i. Consistent Benchmarking: Eliminates human error in performance measurement

ii. Longitudinal Tracking: Enables trend analysis through CSV-formatted historical data

iii. Real-time Alerts: Identifies performance degradation or model failures

iv. Scalable Architecture: Easily accommodates future model additions

The script has been operational since deployment, generating comprehensive performance logs for ongoing system optimization.

4.3 COMPARATIVE ANALYSIS: PREVIOUS VS. CURRENT SYSTEM

4.3.1 OPERATIONAL RELIABILITY

Previous System Limitations:

I. AI model execution required shutdown of backend database connections
II. Surveillance coverage gaps during AI operations
III. Unreliable predictions due to resource contention
IV. System administrators forced to choose between AI functionality and core services

Current System Capabilities:

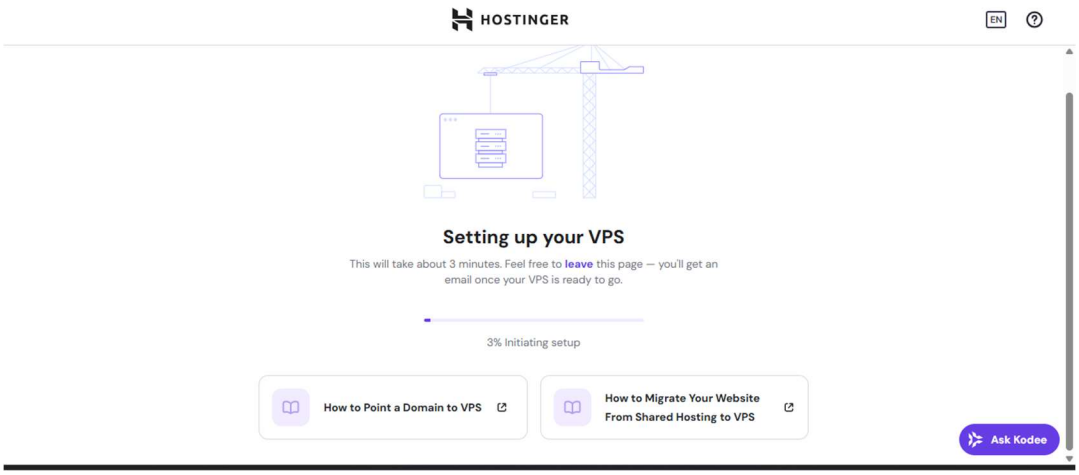
I. Simultaneous operation of all system components
II. Continuous surveillance coverage without interruptions
III. Reliable AI-powered predictions alongside core operations
IV. Complete system availability for healthcare professionals

4.4 PROJECT EXECUTION TIMELINE

The project was completed within the planned four-month timeframe:

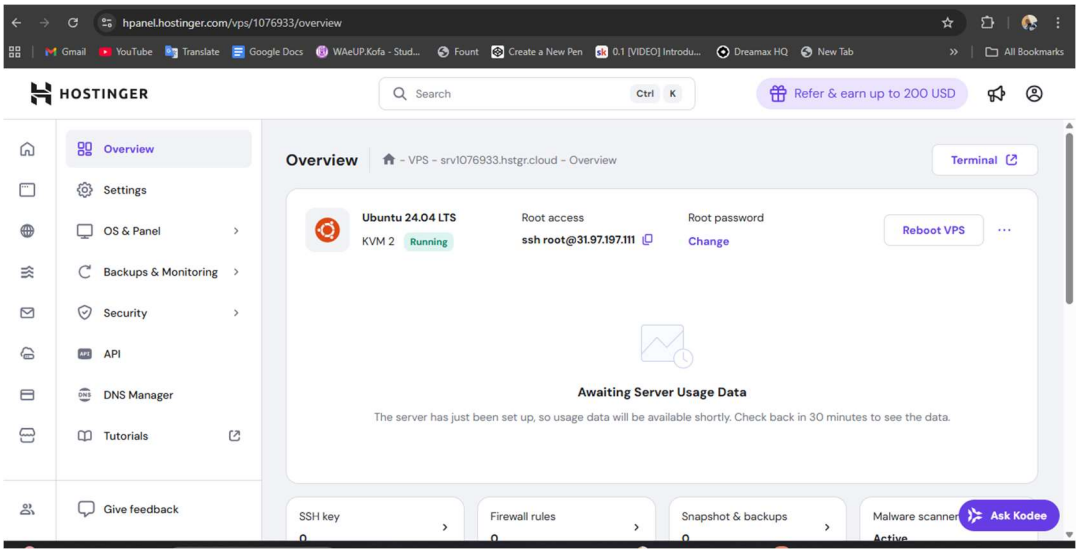
Month 1: Planning and Analysis

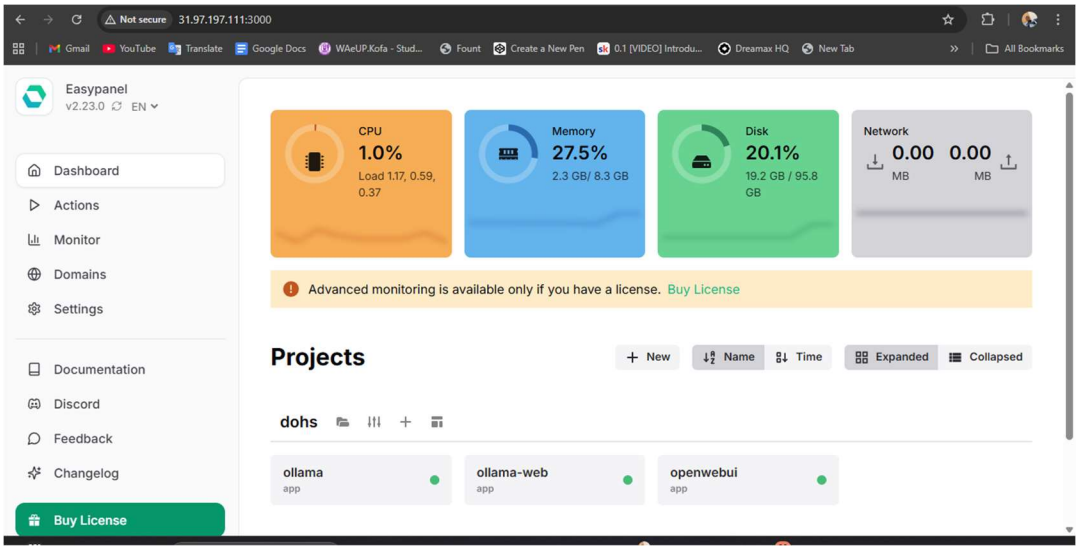
I. A comprehensive assessment of previous VPS limitations identified
II. Resource conflict scenarios documented (AI operations forcing backend shutdowns)
III. Hostinger KVM 2 plan selected based on technical requirements
IV. System architecture and migration strategy finalized



Month 2: Implementation and Migration

I. New VPS environment provisioned via Hostinger Easy Panel
II. Ubuntu 22.04 LTS installed and configured
III. Security hardening implemented (SSH key authentication, UFW firewall)
IV. Docker installed for containerized AI deployment
V. System components migrated without data loss





Month 3: Testing and Performance Evaluation

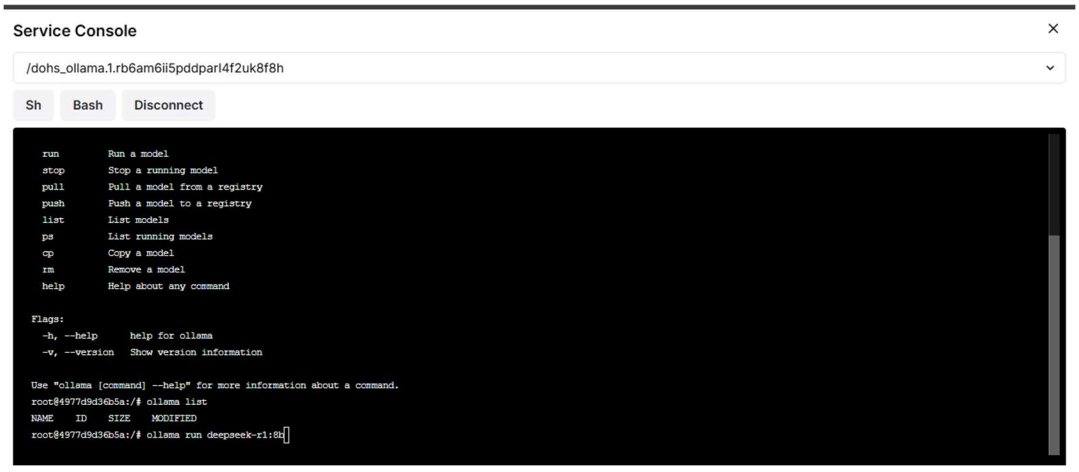
I. Ollama AI container deployed (dohs_ollama.1.yselkdme9dvpivv7aqr2wmz3w)

II. Seven AI models installed and individually tested

III. Manual benchmarking conducted using ollama run commands

IV. Performance metrics collected and analyzed

V. Model compatibility with concurrent system operations verified



Month 4: Automation and Finalization

i. Python monitoring script (Monitor_Ollama_VPS.py) developed

ii. Script deployed to VPS via SSH:

```bash
ssh username@server_ip
cd ~/ollama-Monitor
python3 Monitor_Ollama_VPS.py
```

iii. Automated logging validated through CSV output review
iv. Comprehensive project documentation completed
v. System optimization recommendations formulated

4.5 VALIDATION OF PROJECT OBJECTIVES

The project successfully achieved all stated objectives:

Objective 1: Infrastructure Analysis ✓

Comprehensive analysis of previous VPS performance established baseline metrics and identified specific resource constraints preventing AI integration.

Objective 2: VPS Implementation ✓

Enhanced KVM 2 VPS deployed with complete system migration, configuration, and integration testing successfully completed.

Objective 3: Evaluation ✓

Monitoring system is implemented to evaluate performance. Ollama AI model integration validated under various load conditions with confirmed system stability.

Objective 4: Recommendations ✓

Evidence-based recommendations for system optimization developed based on benchmarking results (detailed in Section 4.6).

4.6 RECOMMENDATIONS FOR SYSTEM OPTIMIZATION

Based on the implementation experience and performance analysis, the following recommendations are proposed:

Short-Term Optimizations

I. Model Selection Strategy: Prioritize lightweight models (cogito, gemma3-translator) for routine queries; reserve larger models (llama3, openbiollm) for complex analytical tasks
II. Resource Monitoring: Continue automated monitoring to identify performance trends and proactively address resource constraints
III. Cache Implementation: Implement caching mechanisms for frequently requested AI model responses to reduce computational overhead

Medium-Term Enhancements

I. Load Balancing: Configure load balancing across AI models based on query complexity and resource availability
II. Database Optimization: Implement query optimization and indexing strategies to further reduce database response times
III. Backup Procedures: Establish automated backup protocols for AI models and surveillance data

Long-Term Scalability Planning

I. Infrastructure Scaling: Monitor system growth and plan for potential upgrade to higher-tier VPS plans as user base expands
II. Model Expansion: Evaluate and integrate additional specialized health AI models as they become available
III. Geographic Redundancy: Consider multi-region VPS deployment for enhanced reliability and disaster recovery

4.7 CHALLENGES ENCOUNTERED AND SOLUTIONS

4.7.1 MIGRATION COMPLEXITY

Challenge: Transferring existing system components without service interruption

Solution: Implemented staged migration approach with comprehensive backups before transitioning services

4.7.2 MODEL RESOURCE REQUIREMENTS

Challenge: Some evaluated models (e.g., clinical-br-mistral-7b-0.2 at 14GB) exceeded available resources

Solution: Established model selection criteria prioritizing efficiency and implemented resource-aware deployment strategy

4.7.3 MONITORING ACCURACY

Challenge: Capturing accurate real-time metrics during rapid resource fluctuations

Solution: Developed automated Python script with immediate post-execution metric capture for reliable data collection

4.8 CONCLUSION

This chapter presented comprehensive results from the VPS implementation and analysis project. The transition to Hostinger KVM 2 infrastructure successfully addressed the resource constraints that previously prevented simultaneous operation of AI models and core system components. Performance benchmarking confirmed that seven specialized health AI models now operate reliably within the enhanced infrastructure, supporting the Digital One Health Surveillance System's mission of integrated health monitoring across human, animal, and environmental domains. The automated monitoring system ensures ongoing performance optimization, while the project's successful completion within the planned timeline demonstrates the viability of VPS-based infrastructure for complex health surveillance applications.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 COCLUSION

This project successfully executed and evaluated a strategic upgrade of the Virtual Private Server (VPS) powering the Digital One Health Surveillance System. The migration from the legacy infrastructure (4GB RAM, 50GB storage) to the Hostinger KVM 2 plan (8GB RAM, 100GB NVMe storage, 2 vCPU cores) resolved critical resource limitations that had previously hindered the simultaneous operation of AI models and core system services.

The most significant outcome of this upgrade is the elimination of operational conflicts between AI-driven functionalities and essential backend services. Previously, system administrators were compelled to disable database connections to accommodate AI model execution—resulting in surveillance blind spots. The enhanced VPS now enables seamless, concurrent operation of the PostgreSQL database, Node.js backend, React frontend, and Ollama AI platform, without resource contention or service disruption.

Seven specialized AI models tailored for health surveillance were successfully deployed: llama3, hippomistral, openbiollm, cogito, antonio-gemma3-smart-q4, aquif-3.0-preview-8b, and gemma3-translator. Benchmarking confirmed these models deliver response times ranging from 14 to 140 seconds while maintaining system stability. To support ongoing performance optimization, a Python-based monitoring tool (Monitor_Ollama_VPS.py) was developed, enabling automated tracking and data-driven insights.

This infrastructure enhancement directly advances the mission of the Digital One Health Surveillance System—integrating human, animal, and environmental health data. Healthcare professionals now benefit from uninterrupted access to AI-enhanced tools for real-time clinical decision support, continuous.

5.2 KEY CONTRIBUTIONS

This project provides: (i) a documented five-phase implementation methodology applicable to similar One Health initiatives; (ii) evidence-based resource specifications for running multiple

AI models concurrently; (iii) demonstration of open-source AI integration in resource-constrained contexts; and (iv) an automated monitoring tool for ongoing system optimization.

5.3 ADDRESSING THE RESEARCH GAP

Previous research focused on theoretical frameworks without practical validation. This project bridges that gap by implementing and analyzing a VPS configuration specifically designed for One Health surveillance, demonstrating real-world performance and operational viability in a cost-effective, mid-tier infrastructure appropriate for resource-constrained public health contexts.

5.4 LIMITATIONS

The study has several limitations: (i) evaluation on a single VPS instance without distributed testing; (ii) limited stress testing under maximum concurrent user load; (iii) AI model selection limited to Ollama-compatible open-source models; (iv) short-term evaluation period; and (v) benchmarking with a single standardized prompt.

5.5 RECOMMENDATIONS

Future work should focus on: (i) advanced model optimization techniques to enable larger specialized models; (ii) comprehensive real-time integration testing under operational conditions; (iii) enhanced security measures including federated learning and differential privacy; (iv) longitudinal cost-benefit analysis; and (v) development of interoperability standards for integration with existing health information systems.

5.6 IMPLICATIONS FOR ONE HEALTH SURVEILLANCE

The project demonstrates that advanced AI capabilities are accessible to public health agencies at modest cost. The Hostinger KVM 2 plan provides sufficient resources for sophisticated surveillance applications, making AI-enhanced health monitoring feasible for resource-constrained contexts. The elimination of service interruptions strengthens surveillance reliability, enabling continuous situational awareness of disease patterns and emerging health threats.

The enhanced infrastructure supports interdisciplinary collaboration across human health, veterinary medicine, and environmental science through a unified platform for data sharing and analysis. The system's capacity to simultaneously analyze data across all three health domains

operationalizes the One Health concept of interconnected health, strengthening capacity for early detection and rapid response to health threats at the human-animal-environment interface.

5.7 FINAL REMARKS

This project successfully transformed the Digital One Health Surveillance System from a resource-constrained platform requiring operational compromises to a robust infrastructure supporting concurrent AI operations alongside core surveillance functions. The implementation validates that mid-tier VPS infrastructure, when appropriately configured, can support sophisticated health surveillance applications with integrated AI capabilities.

The documented methodology, performance benchmarking results, and automated monitoring tools provide practical resources for public health agencies and research institutions implementing or enhancing digital surveillance systems. The enhanced system now serves as a functional platform for integrated health monitoring, enabling healthcare professionals, veterinarians, and environmental scientists to collaboratively protect human, animal, and environmental health through reliable, AI-enhanced surveillance capabilities.

REFERENCES

1. Olaye, E., Omafovbe, I., Aigbe, W. O., & Obuh, D. (2024). Personalized Governance Strategy for Patient Data in a Digital One Health Surveillance System. *Procedia Computer Science*, 254, 20-29. https://doi.org/10.1016/j.procs.2025.02.060

2. Rodrigues, J. J., de la Torre, I., Fernández, G., & López-Coronado, M. (2013). Analysis of the security and privacy requirements of cloud-based electronic health records systems. *Journal of Medical Internet Research*, 15(8), e186. https://doi.org/10.2196/jmir.2494

3. Regola, N., & Chawla, N. V. (2013). Storing and Using Health Data in a Virtual Private Cloud. *Journal of Medical Internet Research*, 15(3), e63. https://doi.org/10.2196/jmir.2076

4. Adelmeyer, M., Teuteberg, F., & Galar, D. (2019). Security and Privacy of Personal Health Records in Cloud Computing Environments. *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 3478-3487.

5. Sakr, A. E., & Tawfik, H. (2020). A secure client-side framework for protecting the privacy of health data stored on the cloud. *International Journal of Information Security*, 19, 459-471.

6. Rashid, Y., Khan, J. I., & Khalid, M. (2019). Hybrid Cryptographic Access Control for Cloud-Based EHR Systems. *IEEE Access*, 7, 173273-173286.

7. Naik, H., & Kannan, M. K. J. (2020). A Survey on Protecting Confidential Data over Distributed Storage in Cloud. Available at SSRN: https://ssrn.com/abstract=3740465

8. Zinsstag, J., Schelling, E., Waltner-Toews, D., & Tanner, M. (2011). From "one medicine" to "one health" and systemic approaches to health and well-being. *Preventive Veterinary Medicine*, 101(3-4), 148-156.

9. Gibbs, E. P. J. (2014). The evolution of One Health: a decade of progress and challenges for the future. *Veterinary Record*, 174(4), 85-91.

10. Atlas, R. M., Rubin, C. B., Maloy, S., Daszak, P., Colwell, R. R., & Hyde, B. (2010). One Health—attaining optimal health for people, animals, and the environment. *Microbe Magazine*, 5(9), 383-389.

11. Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. *National Institute of Standards and Technology Special Publication*, 800-145.

12. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50-58.

APPENDIX

Full Source Code For Script Automation:

```
#!/usr/bin/env python3
"""
Monitor_Ollama_VPS.py
Automated Performance Monitoring Script for Ollama AI Models on VPS
Digital One Health Surveillance System Implementation
"""

import psutil
import subprocess
import csv
import time
from datetime import datetime

# Configuration Constants
CONTAINER_NAME = "dohs_ollama.1.yselkdme9dvpivv7aqr2wmz3w"
TEST_PROMPT = "What is One Health?"
OUTPUT_FILE = "results.csv"

# Model list for benchmarking
MODELS = [
    "llama3",
    "hippomistral",
    "openbiollm",
    "cogito",
    "antonio-gemma3-smart-q4",
    "aquif-3.0-preview-8b",
    "gemma3-translator"
]
```

```
def get_system_metrics():
    """
    Capture current system resource utilization metrics.

    Returns:
        dict: Dictionary containing CPU, RAM, and disk usage percentages
    """
    cpu_percent = psutil.cpu_percent(interval=1)
    ram_percent = psutil.virtual_memory().percent
    disk_percent = psutil.disk_usage('/').percent

    return {
        'cpu': cpu_percent,
        'ram': ram_percent,
        'disk': disk_percent
    }

def run_model_benchmark(model_name):
    """
    Execute a single model and capture performance metrics.

    Args:
        model_name (str): Name of the Ollama model to benchmark

    Returns:
        dict: Performance metrics including baseline, peak, and duration
    """
    print(f"\n{'='*60}")
    print(f"Benchmarking Model: {model_name}")
    print(f"{'='*60}")
```

```
# Capture baseline metrics
print("Capturing baseline metrics...")
baseline_metrics = get_system_metrics()
print(f"Baseline - CPU: {baseline_metrics['cpu']}%, "
      f"RAM: {baseline_metrics['ram']}%, "
      f"Disk: {baseline_metrics['disk']}%")

# Construct Docker command
docker_command = [
    "docker", "exec", "-i", CONTAINER_NAME,
    "ollama", "run", model_name, TEST_PROMPT
]

# Execute model and measure duration
start_time = time.time()

try:
    print(f"Executing model '{model_name}'...")
    result = subprocess.run(
        docker_command,
        capture_output=True,
        text=True,
        timeout=600  # 10-minute timeout
    )

    execution_duration = time.time() - start_time

    # Capture peak metrics immediately after execution
    peak_metrics = get_system_metrics()
```

```
    print(f"Execution completed in {execution_duration:.2f} seconds")
    print(f"Peak - CPU: {peak_metrics['cpu']}%, "
          f"RAM: {peak_metrics['ram']}%, "
          f"Disk: {peak_metrics['disk']}%")

    # Prepare results dictionary
    benchmark_result = {
        'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        'model': model_name,
        'cpu_baseline': baseline_metrics['cpu'],
        'cpu_peak': peak_metrics['cpu'],
        'ram_baseline': baseline_metrics['ram'],
        'ram_peak': peak_metrics['ram'],
        'disk_baseline': baseline_metrics['disk'],
        'disk_peak': peak_metrics['disk'],
        'duration': round(execution_duration, 2),
        'status': 'Success',
        'error': ''
    }

    # Display model response (truncated)
    if result.stdout:
        response_preview = result.stdout[:200] + "..." if len(result.stdout) > 200 else result.stdout
        print(f"\nModel Response Preview:\n{response_preview}\n")

    return benchmark_result

except subprocess.TimeoutExpired:
    execution_duration = time.time() - start_time
    print(f"ERROR: Model execution timed out after {execution_duration:.2f} seconds")
```

```
    return {
        'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        'model': model_name,
        'cpu_baseline': baseline_metrics['cpu'],
        'cpu_peak': 'N/A',
        'ram_baseline': baseline_metrics['ram'],
        'ram_peak': 'N/A',
        'disk_baseline': baseline_metrics['disk'],
        'disk_peak': 'N/A',
        'duration': round(execution_duration, 2),
        'status': 'Timeout',
        'error': 'Execution exceeded 600 second timeout'
    }

except Exception as e:
    execution_duration = time.time() - start_time
    print(f"ERROR: {str(e)}")

    return {
        'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        'model': model_name,
        'cpu_baseline': baseline_metrics['cpu'],
        'cpu_peak': 'N/A',
        'ram_baseline': baseline_metrics['ram'],
        'ram_peak': 'N/A',
        'disk_baseline': baseline_metrics['disk'],
        'disk_peak': 'N/A',
        'duration': round(execution_duration, 2),
        'status': 'Failed',
        'error': str(e)
    }
```

```
def save_results_to_csv(results, filename=OUTPUT_FILE):
    """
    Save benchmark results to CSV file.

    Args:
        results (list): List of benchmark result dictionaries
        filename (str): Output CSV filename
    """
    if not results:
        print("No results to save.")
        return

    fieldnames = [
        'timestamp', 'model', 'cpu_baseline', 'cpu_peak',
        'ram_baseline', 'ram_peak', 'disk_baseline', 'disk_peak',
        'duration', 'status', 'error'
    ]

    try:
        with open(filename, 'w', newline='') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(results)

        print(f"\n{'='*60}")
        print(f'Results successfully saved to: {filename}')
        print(f"{'='*60}")

    except Exception as e:
        print(f'ERROR: Failed to save results to CSV: {str(e)}')
```

```
def main():
    """
    Main execution function for automated monitoring.
    """
    print("\n" + "="*60)
    print("OLLAMA VPS PERFORMANCE MONITORING SYSTEM")
    print("Digital One Health Surveillance System")
    print("="*60)
    print(f"Start Time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"Container: {CONTAINER_NAME}")
    print(f"Test Prompt: '{TEST_PROMPT}'")
    print(f"Models to Test: {len(MODELS)}")
    print("="*60)

    all_results = []

    # Benchmark each model sequentially
    for idx, model in enumerate(MODELS, 1):
        print(f"\n[{idx}/{len(MODELS)}] Processing model: {model}")

        result = run_model_benchmark(model)
        all_results.append(result)

        # Brief pause between models to allow system stabilization
        if idx < len(MODELS):
            print("\nPausing 10 seconds before next model...")
            time.sleep(10)

    # Save all results to CSV
    save_results_to_csv(all_results)
```

```
    # Print summary
    print("\n" + "="*60)
    print("BENCHMARKING SUMMARY")
    print("="*60)
    successful = sum(1 for r in all_results if r['status'] == 'Success')
    failed = len(all_results) - successful
    print(f"Total Models Tested: {len(all_results)}")
    print(f"Successful: {successful}")
    print(f"Failed: {failed}")
    print(f"End Time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    print("="*60 + "\n")

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