การนำเสนอวิธีการสร้างแบบจำลองโดยใช้การเรียนรู้ของเครื่องในรูปแบบผสมเพื่อทำนายประสิทธิภาพและประเมินสถานะของระบบซอฟต์แวร์บนคลาวด์

SOFTWARE SYSTEM PERFORMANCE PREDICTION AND HEALTH ASSESSMENT IN CLOUD ENVIRONMENTS: INTRODUCING CSYSGUARD, AN ENSEMBLE MODELING APPROACH

ณัฏฐ์ ชัยรัตน์

NUTT CHAIRATANA

วิทยานิพนธ์นี้สำหรับการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมหุ่นยนต์ และระบบอัจฉริยะเชิงคำนวณ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2567

KMITL-2024-EN-M-407-292

SOFTWARE SYSTEM PERFORMANCE PREDICTION AND HEALTH ASSESSMENT IN CLOUD ENVIRONMENTS: INTRODUCING CSYSGUARD, AN ENSEMBLE MODELING APPROACH

NUTT CHAIRATANA

A THESIS SUBMITTED IN FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ROBOTICS AND COMPUTATIONAL INTELLIGENCE SYSTEMS

SCHOOL OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2024

KMITL-2024-EN-M-407-292

หัวข้อวิทยานิพนธ์	การนำเสนอวิธีการสร้างแบบจำลองโดยใช้การเรียนรู้ของเครื่องในรูปแบบผสมเพื่อทำนายประสิทธิภาพและประเมินสถานะของระบบซอฟต์แวร์บนคลาวด์
นักศึกษา	นายณัฏฐ์ ชัยรัตน์
รหัสประจำตัว	65016030
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมหุ่นยนต์ และระบบอัจฉริยะเชิงคำนวณ
พ.ศ.	2567
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.รัฐชัย ชาวอุทัย

บทคัดย่อ

บทความนี้นำเสนอระบบซีสิสการ์ด (cSysGuard) เป็นระบบให้ความสามารถในการตรวจสอบภายในเซิร์ฟเวอร์ที่มีรูปแบบไม่เก็บข้อมูลในสภาพแวดล้อมคลาวด์ ระบบนี้สามารถจัดการกับความท้าทายที่สำคัญของภาระงานที่เปลี่ยนแปลงไป ซึ่งบ่อยครั้งทำให้เกิดการขัดจังหวะของบริการเนื่องจากการจัดสรรทรัพยากรที่ไม่เพียงพอ หัวใจหลักของวิธีการของซีสิสการ์ด (cSysGuard) คือแบบจำลองการถดถอยแบบผสมที่ใช้วิธีการซ้อนทับ (Stacking Ensemble Regression Model) เพื่อทำนายตัวชี้วัดประสิทธิภาพของเซิร์ฟเวอร์ ซึ่งมาพร้อมกับแบบจำลองการจำแนกประเภท (Classification Model) ที่ประเมินสถานะของระบบตามการทำนายเหล่านี้เพื่อระบุความล้มเหลวที่อาจเกิดขึ้นล่วงหน้า การวิเคราะห์เปรียบเทียบเผยให้เห็นว่าซีสิสการ์ด (cSysGuard) มีประสิทธิภาพเหนือกว่าแบบจำลองการตรวจสอบแบบดั้งเดิม รวมถึงวิธีการทางสถิติและการเรียนรู้ของเครื่อง ด้วยการปรับปรุงความแม่นยำในการทำนายที่มีตั้งแต่ใกล้เคียงกับค่าเท่ากันไปจนถึงระดับ 2.28 เท่า ขึ้นอยู่กับตัวชี้วัด นอกจากนี้ความสามารถในการประเมินสถานะของระบบที่ได้รับการเสริมด้วย Decision Trees ที่มีการปรับแต่งไฮเปอร์พารามิเตอร์อย่างละเอียดได้คะแนน F1 อยู่ที่ 89.79% ผลการศึกษานี้นำเสนอข้อมูลเชิงลึกที่มีค่าทั้งในด้านทฤษฎีและปฏิบัติของการตรวจสอบเซิร์ฟเวอร์ โดยแสดงถึงประสิทธิภาพของโมเดลที่ดีสำหรับการจัดการความซับซ้อนที่เปลี่ยนแปลงได้ในการบริหารโครงสร้างคลาวด์

Thesis	Software System Performance Prediction and Health Assessment in Cloud Environments: Introducing cSysGuard, an Ensemble Modeling Approach
Student	Mr. Nutt Chairatana
Student ID.	65016030
Degree	Master of Engineering
Program	Robotics and Computational Intelligence Systems
Year	2024
Thesis Advisor	Assoc. Prof. Dr. Rathachai Chawuthai

ABSTRACT

This paper introduces cSysGuard, a cutting-edge framework bolsters monitoring capabilities within stateless cloud computing environments. cSysGuard addresses the significant challenges of varying workloads, often leading to service disruptions due to inadequate resource allocation. The core of cSysGuard's methodology is an ensemble regression model utilizing a stacking approach to predict server key performance indicators alongside a classification model that evaluates system health based on these predictions to identify potential failures preemptively. Comparative analyses reveal that cSysGuard outperforms conventional monitoring models, including statistical and machine learning approaches, with predictive accuracy improvements ranging from close to parity to a remarkable 2.28-fold, depending on the metric. Furthermore, the system's health assessment capabilities, enhanced by Decision Trees with fine-tuned hyperparameters, achieved a macro-averaged F1 score of 89.79%. The findings of this study offer valuable insights into both the theoretical and practical realms of server monitoring, showcasing a robust solution for addressing the dynamic complexities of cloud infrastructure management.

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จได้ด้วยความกรุณาจากอาจารย์ที่ปรึกษา รศ.ดร.รัฐชัย ชาวอุทัย และคณะกรรมการที่ได้เสียสละเวลาในการให้คำปรึกษาและคำชี้แนะช่วยแก้ปัญหาในการศึกษาวิจัยต่าง ๆ ตลอดจนให้ความรู้และประสบการณ์ที่ดีแก่ข้าพเจ้า ข้าพเจ้ารู้สึกซาบซึ้งในความอนุเคราะห์จากท่านอาจารย์เป็นอย่างมากและขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ บิดา มารดา ครูอาจารย์ เพื่อนร่วมชั้นเรียน ตลอดจนเพื่อนร่วมงานที่ได้เสียสละเวลาให้ข้อเสนอแนะ และแสดงความคิดเห็นเพื่อให้งานวิจัยนี้มีความสมบูรณ์ยิ่งขึ้น

ข้าพเจ้าหวังเป็นอย่างยิ่งว่าองค์ความรู้จากงานวิจัยที่อยู่ในวิทยานิพนธ์ฉบับนี้สามารถก่อให้เกิดคุณประโยชน์แก่วงการวิชาการรวมถึงภาคอุตสาหกรรมไม่มากก็น้อย สำหรับคุณงามความดีอันใดที่เกิดจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดา ซึ่งเป็นที่รักและเคารพยิ่ง ตลอดจนครูอาจารย์ที่เคารพทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้และถ่ายทอดประสบการณ์ที่ดีให้แก่ข้าพเจ้า

ณัฏฐ์ ชัยรัตน์

สารบัญ

หน้า

สารบัญตาราง

สารบัญรูป

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคโนโลยีคลาวด์ (Cloud Computing) เป็นที่รู้จักกันดีในเรื่องความสามารถในความยืดหยุ่น ซึ่งได้ปฏิวัติการจัดการทรัพยากรของระบบ ภายในขอบเขตที่กว้างขวางของคลาวด์ เซิร์ฟเวอร์ที่ไม่ถือข้อมูล (Stateless Server) มีบทบาทสำคัญ รูปแบบนี้ช่วยให้ปรับขนาดทรัพยากรของเซิร์ฟเวอร์ได้อย่างยืดหยุ่น เพราะรูปแบบนี้ไม่จำเป็นต้องเก็บข้อมูล Session ของผู้ใช้บริการ ซึ่งเพิ่มความสามารถในการปรับขนาดระบบต่อความต้องการที่เปลี่ยนแปลงอย่างรวดเร็วเป็นอย่างมาก อีกทั้ง เซิร์ฟเวอร์ลักษณะดังกล่าวยังสามารถทำงานร่วมกับสถาปัตยกรรมอื่น ๆ ได้อย่างง่ายดาย โดยเหตุนี้ จึงกลายเป็นที่นิยมในการโฮสต์แอปพลิเคชั่นบนคลาวด์ ที่มีการเปลี่ยนแปลงได้อย่างรวดเร็วและตลอดเวลา แต่อย่างไรก็ตาม ลักษณะที่ไม่สามารถคาดเดาได้ของจำนวนผู้ใช้บริการมักทำให้การจัดสรรทรัพยากรให้มีประสิทธิภาพเป็นเรื่องที่ยาก ซึ่งอาจทำให้ผู้ให้บริการตรวจพบเจอการล้มเหลวของระบบที่เกิดขึ้นได้อย่างไม่คาดคิด ที่จะละเมิดข้อตกลงระดับของการให้บริการ (Service Level Agreements หรือ SLAs) หรือในอีกมุมหนึ่ง จากปัญหาที่เกี่ยวกับการไม่สามารถคาดเดาจำนวนของผู้ใช้บริการได้อย่างมีประสิทธิภาพนั้น อาจส่งผลให้มีการจัดการทรัพยากรได้ไม่เหมาะสม ที่ผู้ดูแลระบบอาจเตรียมทรัพยากรที่มากหรือน้อยเกินไป แม้ว่าระบบคลาวด์จะเสนอกลไกที่สามารถปรับขนาดทรัพยากรให้สอดคล้องกับเกณฑ์ที่ได้ถูกกำหนดไว้อย่างอัตโนมัติ ระบบพวกนั้นมักพึ่งพาเกณฑ์ที่กำหนดไว้อย่างตรงไปตรงมา ไม่มีความยืดหยุ่น เช่น จะมีปรับขนาดของทรัพยากรของเซิร์ฟเวอร์ก็ต่อเมื่อมีการใช้ CPU (Central Processing Unit) ของเซิร์ฟเวอร์มากกว่า 80 เปอร์เซ็นต์เท่านั้น วิธีเหล่านี้อาจมีปัญหาเมื่อตอนผู้ใช้บริการเพิ่มขึ้นและใช้ทรัพยากรที่หลากหลาย ซึ่งทำให้วิธีเหล่านี้มีประสิทธิภาพในการทำงานที่น้อยลง เพราะปฏิสัมพันธ์ระหว่างตัวชี้วัดทรัพยากรและสถานะของเซิร์ฟเวอร์นั้นก็เป็นเรื่องซับซ้อน มันมักถูกกำหนดโดยปัจจัยต่าง ๆ เช่น ลักษณะของแอปพลิเคชั่น หรือการกำหนดค่าระบบ การพึ่งพาเพียงตัวชี้วัดเดียวในการประเมินความเพียงพอของการจัดสรรทรัพยากรอาจเป็นเรื่องที่ท้าทายเช่นกัน มิฉะนั้น ผู้ดูแลระบบคลาวด์จะไม่สามารถจัดสรรและจัดการทรัพยากรของระบบได้อย่างมีประสิทธิภาพเพื่อตอบสนองต่อความต้องการที่ไม่สามารถคาดเดาได้เพิ่มขึ้นในคลาวด์ อีกปัญหาหนึ่ง ผู้ให้บริการบางรายได้เสนอวิธีการที่จะทำนายการใช้ทรัพยากรของเซิร์ฟเวอร์ เพื่อให้ผู้ดูแลระบบนั้นรู้ถึงการใช้ที่จะเกิดขึ้นอันใกล้ แต่อย่างไรก็ตาม เราค้นพบว่าระบบนี้จะถูกฝึก

โดยใช้วิธีการที่จะอิงตามรูปแบบที่เคยมีมาก่อนหน้านี้ ซึ่งมักจะมีปัญหากับรูปแบบที่ไม่สม่ำเสมอที่เข้ามาในระบบ โดยท้ายที่สุด จะนำไปสู่การจัดการทรัพยากรของเซิร์ฟเวอร์อย่างไม่มีประสิทธิภาพ เช่นกัน

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

เพื่อจัดการกับความท้าทายเหล่านี้ งานวิจัยนี้ต้องการเครื่องมือที่ซับซ้อนเพื่อทำนายตัวชี้วัดสำคัญได้อย่างแม่นยำ และเข้าใจความสัมพันธ์ที่ซับซ้อนระหว่างตัวชี้วัดและสถานะของเซิร์ฟเวอร์ในสภาพแวดล้อมที่มีการเปลี่ยนแปลงอยู่ตลอดเวลา มากไปกว่านั้น งานวิจัยนี้ต้องสามารถให้ผลลัพธ์ที่แม่นยำกว่ารูปแบบการใช้งานการเรียนรู้ด้วยเครื่องในที่ผ่านมา และให้ความยืดหยุ่นในการใช้งานอีกด้วย งานวิจัยนี้มีเป้าหมายที่จะค้นคว้าและพัฒนาระบบที่สามารถตอบสนองต่อความต้องการนี้ได้ ซึ่งท้ายที่สุด งานวิจัยได้พัฒนาระบบที่มีชื่อว่าซีสิสการ์ด (cSysGuard) หรือที่มีชื่อเต็มว่า Cloud System Guard โดยระบบนี้เป็นเครื่องมือที่ถูกออกแบบมาเพื่อทำนายการใช้งานทรัพยากรต่าง ๆ ที่ไม่สามารถคาดเดาได้ของเซิร์ฟเวอร์บนคลาวด์ได้อย่างมีประสิทธิภาพ โดยซีสิสการ์ด (cSysGuard) ประกอบด้วยสองส่วนหลักนั้นก็คือตัวทำนายตัวชี้วัดประสิทธิภาพของระบบที่อาจเกิดขึ้นในอนาคต และการประเมินสถานะของเซิร์ฟเวอร์ โดยในส่วนที่ประเมินสถานะของระบบนั้น ระบบจะอิงจากผลของการทำนายตัวชี้วัดประสิทธิภาพก่อนหน้าอีกที ในงานวิจัยนี้ เราทำการทดสอบซีสิสการ์ด (cSysGuard) กับสภาพแวดคลาวด์ที่ไม่สามารถคาดเดาได้ เพื่อจะทำการวัดผลว่าระบบที่เราออกแบบและพัฒนามานั้นจะสามารถให้ผลที่ดีกว่าระบบที่ใช้อยู่ในปัจจุบันได้หรือไม่

1.3 สมมุติฐานของการศึกษา

อุปสรรคอย่างหนึ่งสำหรับการออกแบบระบบซีสิสการ์ด (cSysGuard) คือการพัฒนาระบบที่สามารถทำนายโหลดของคลาวด์ที่อยู่ในสภาพแวดล้อมที่มีความซับซ้อนเปลี่ยนแปลงได้ตลอดเวลา และอีกทั้งยังต้องให้ผลการทำนายที่ดีกว่าการเรียนรู้ด้วยเครื่องแบบดั้งเดิม ซึ่งรวมถึงรูปแบบที่ใช้โมเดลประสาทเทียม หรือ Neural Network ที่มีจำนวนชั้นหลาย ๆ ชั้น หรือนั้นก็คือเทคนิคการเรียนรู้เชิงลึกหรือ Deep Learning ซึ่งเป็นวิธีการที่สามารถให้ผลของการทำนายที่มีประสิทธิภาพที่ดี แต่ยังทรัพยากรในการทำนายที่เยอะ เพราะฉะนั้น ขณะที่ซีสิสการ์ด (cSysGuard) ยังคงต้องเป็นระบบที่ให้ประสิทธิภาพในการทำนายที่ดี ในเวลาเดียวกัน ก็ต้องเป็นระบบที่มีความยืดหยุ่นในการปรับใช้ทรัพยากรให้เหมาะสมกับสภาพแวดล้อมนั้น ๆ อีกด้วย เพื่อให้เกิดความสมดุลระหว่างความแม่นยำในทำนายและการใช้ทรัพยากรในการคำนวณอย่างเหมาะสม ด้วยเหตุนี้ งานวิจัยจึงใช้การเรียนรู้ด้วย

เครื่องในรูปแบบผสม (Ensemble Model) เพื่อผู้ใช้สามารถดึงข้อดีเฉพาะของแต่ละโมเดล หรือปรับจำนวนโมเดลที่ใช้ให้เหมาะสมกับสภาพแวดล้อมนั้น ๆ ซึ่งจะให้ความยืดหยุ่นในการใช้งานมากที่สุด

อุปสรรคอย่างหนึ่งของการออกแบบระบบนี้นั้นก็คือการประเมินสถานะของเซิร์ฟเวอร์ด้วย ข้อมูลที่จะนำมาป้อนใส่โมเดลนั้นจะมาจากผลของการทำนายตัวชี้วัดประสิทธิภาพของระบบที่อาจเกิดขึ้นในอนาคต ซึ่งไม่ได้เป็นการอิงจากค่าจริงที่เกิดขึ้น ณ ตอนนั้น แต่เป็นค่าที่อาจเกิดขึ้นในอนาคต ซึ่งมีโอกาสที่จะเกิดความคาดเคลื่อนในการทำนายสถานะของเซิร์ฟเวอร์ได้ ด้วยเหตุนี้ได้ใช้วิธีการประเมินสถานะของเซิร์ฟเวอร์ด้วยเทคนิค Sliding Window โดยเป็นเทคนิคที่จะนำช่วงหนึ่งของชุดข้อมูลก่อนหน้านี้มาช่วยประเมินสถานะของเซิร์ฟเวอร์เพื่อช่วยเพิ่มประสิทธิภาพในการทำนายผลสุดท้ายให้ออกมาแม่นยำมากที่สุด

1.4 แนวความคิดที่ใช้ในการวิจัย

งานวิจัยนี้มุ่งหวังที่จะให้ซีสิสการ์ด (cSysGuard) ทำนายตัวชี้วัดประสิทธิภาพและใช้ประโยชน์จากการทำนายเพื่อประเมินสถานะของระบบ วิธีการนี้ช่วยให้สามารถระบุและแก้ไขความล้มเหลวที่อาจเกิดขึ้นได้ล่วงหน้า ทำให้สามารถจัดการทรัพยากรได้อย่างรอบคอบในสภาพแวดล้อมคลาวด์ที่เปลี่ยนแปลงได้ การที่มีสภาพแวดล้อมคลาวด์ที่แตกต่างไม่เหมือนกัน เราจำเป็นต้องมีระบบที่ยืดหยุ่นสามารถปรับให้เหมาะกับสภาพแวดล้อมนั้น ๆ โดยภาพรวม เราจึงใช้เทคนิค Stacking Ensemble Regression Model ซึ่งเป็นการทำนายแบบจำลองการถดถอยแบบผสมที่ใช้วิธีการซ้อนทับ เพื่อเพิ่มประสิทธิภาพของการทำนายเซิร์ฟเวอร์ที่รูปแบบซับซ้อนได้ดียิ่งขึ้น ในอีกส่วนที่เราจะประเมินสถานะของเซิร์ฟเวอร์ งานวิจัยนี้จะใช้ Classification Model ที่เหมาะสมที่สุด เป็นส่วนท้ายที่สุด ซึ่งการทำแบบนี้จะสามารถทำให้เราวิเคราะห์และรู้การล้มเหลวของเซิร์ฟเวอร์ในอนาคต ซึ่งรายละเอียดของระบบซีสิสการ์ด (cSysGuard) จะมีการอธิบายในบทถัดไป

1.5 ขอบเขตการวิจัย

งานวิยังได้จำลองสภาพแวดล้อมแอปพลิเคชั่นบนคลาวด์ เพื่อนำมาใช้เป็นกรณีศึกษาในครั้งนี้ เราได้ใช้ประโยชน์จากผู้ให้บริการคลาวด์ที่เป็นของ Digital Ocean โดยติดตั้งเครื่องเสมือน (Virtual Machine) ด้วยระบบปฏิบัติการ Linux ด้วยกันสามเครื่อง โดยแต่ละเครื่องจะมี 1vCPU, 1GB ของหน่วยความจำ และดิสก์ 10GB แต่ละเครื่องมีความรับผิดชอบที่แตกต่างกัน ซึ่งจะมีอธิบายเพิ่มเติมในบท 3.1.2 งานวิจัยจะทำการศึกษาตรวจสอบ

การทำงานระบบเป็นเวลา 48 ชั่วโมง ด้วยการจำลองผู้ใช้งานที่เราได้เตรียมเอาไว้ โดยจะมีการเปลี่ยนแปลงคำขอและปริมาณอยู่ตลอดเวลา เพื่อให้เกิดความสมจริงและซับซ้อนมากที่สุด

1.6 ขั้นตอนของการศึกษา

งานวิจัยนี้จะนำเสนอรายละเอียดการทดลองและการทำงานของระบบซีสิสการ์ด (cSysGuard) ซึ่งได้ทดลองกับแอปพลิเคชั่นที่มีรูปแบบไม่ถือข้อมูล โดยการนำเสนอจะเริ่มต้นด้วยการทบทวนงานวิจัยที่เกี่ยวข้องเพื่อวางรากฐานสำคัญในส่วนต่อ ๆ ไป ต่อจากนั้น งานวิจัยจะอธิบายวิธีการของการทำวิจัยที่เกี่ยวกับการทำนายตัวชี้วัดประสิทธิภาพและการประเมินสถานะของระบบเซิร์ฟเวอร์ ต่อมา งานวิจัยจะนำเสนอผลลัพธ์ของการทำนายแต่ละส่วนของซีสิสการ์ด (cSysGuard) และนำเสนอมุมมองใหม่ที่ถูกค้นพบระหว่างการทำทดลอง และท้ายที่สุดท้าย งานวิจัยจะสรุปผลของการค้นพบและนำเสนอทิศทางที่สามารถต่อยอดได้ในอนาคตต่อไป

บทที่ 2

การทบทวนงานวิจัย

2.1 การใช้การเรียนรู้ด้วยเครื่องในสภาพแวดล้อมคลาวด์

แม้ว่าผู้ให้บริการคลาวด์จะนำเสนอนวัตกรรมกลยุทธ์ในการจัดการทรัพยากรแบบใหม่ ๆ แต่ก็ยังมีความท้าทายในการตรวจสอบการใช้ทรัพยากรในสภาพแวดล้อมคลาวด์ที่มีการเปลี่ยนแปลงอยู่ตลอดเวลา ซึ่งมักทำให้การตรวจสอบที่มีอยู่นั้นมีประสิทธิภาพที่น้อยลงไป โดยท้ายที่สุด อาจส่งผลให้มีการปรับใช้ทรัพยากรล่วงหน้าในปริมาณที่ไม่เหมาะสมกับความต้องการ ณ เวลานั้น จากปัญหาที่เกิดขึ้น ผู้ใช้บริการคลาวด์ต้องการวิธีการตรวจสอบที่ยืดหยุ่นมากขึ้นเพื่อประเมินและปรับตัวตามความต้องการที่เปลี่ยนแปลงได้อย่างต่อเนื่องของสภาพแวดล้อมที่ซับซ้อน

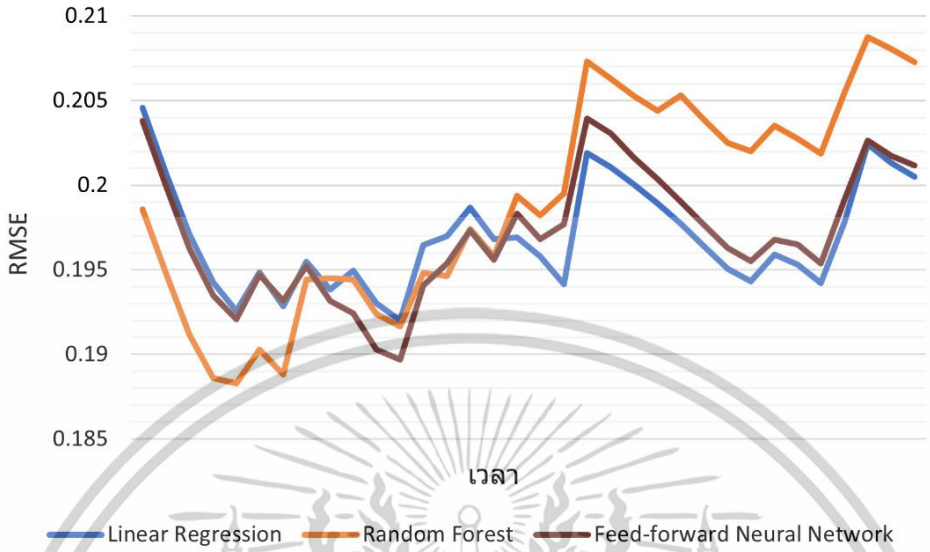
ดังนั้น จึงมีความสนใจเพิ่มขึ้นในการใช้เทคนิคการเรียนรู้ของเครื่องเพื่อรับมือกับความท้าทายเหล่านี้ การวิเคราะห์การทำนายที่ขับเคลื่อนด้วยการเรียนรู้ของเครื่องสามารถวิเคราะห์รูปแบบข้อมูลที่ซับซ้อนเพื่อทำนายปัญหาที่อาจเกิดขึ้น วิธีการรับมือล่วงหน้านี้เพิ่มความน่าเชื่อถือของระบบและการจัดสรรทรัพยากร ซึ่งต้องรับรองได้ว่าการบริการยังคงสามารถปรับขนาดและตอบสนองต่อความต้องการที่แตกต่างกันได้ การผสานการเรียนรู้ของเครื่องเข้ากับคลาวด์แสดงถึงความก้าวหน้าที่สำคัญในการปรับปรุงการจัดการทรัพยากรของระบบเซิร์ฟเวอร์

2.2 งานวิจัยที่เกี่ยวข้อง

หลายงานวิจัยมักจะหาโมเดลตัวที่ให้ผลการทำนายที่ดีที่สุดมาเป็นตัวเลือกและผลสรุปของงานวิจัย โดยจะไม่คำนึงถึงความเฉพาะเจาะจงของระบบ ซึ่งเป็นวิธีการที่มักเรียกว่า one-size-fits-all อย่างไรก็ตาม ตามที่งานวิจัยของ Kim et al. ได้ทำการพิสูจน์ว่าการพึ่งพาโมเดลทำนายเพียงหนึ่งโมเดลนั้นไม่เพียงพอในการจัดการกับลักษณะการทำงานของคลาวด์ที่เปลี่ยนแปลงได้ตลอดเวลาหรือที่มีรูปแบบความผันผวนระยะสั้น ฉะนั้น การเลือกโมเดลตัวใดตัวนึงเพื่อมาทำนายในสภาพแวดดังกล่าวอาจเป็นที่ไม่เหมาะสม ต่อมา งานวิจัยของ Singh et al. เสนอวิธีการแก้ไขอื่นโดยใช้ Support Vector Machine เพื่อจำแนกภาระงานเข้าสู่หมวดหมู่ เช่น "ต่ำมาก" "ต่ำ" "ปานกลาง" และ "สูง" ถึงแม้ว่าวิธีการนี้จะเสนอภาพรวมที่เรียบง่ายของสถานะ การพึ่งพาตัวแปรหมวดหมู่นั้นไม่

เพียงพอสำหรับการประเมินความต้องการทรัพยากรของแอปพลิเคชั่นที่แม่นยำ โดยจำเป็นต้องมีค่าหรือผลลัพธ์การทำนายที่ละเอียดอ่อนมากกว่านี้ เพื่อที่จะได้ปรับทรัพยากรให้เหมาะสมกับสถานการณ์นั้น ๆ ได้ดีกว่า งานวิจัยของ Gao et al. และ Caron et al. เสนอวิธีการทำนายที่อิงตามข้อมูลตามรูปแบบที่เกิดขึ้นก่อนหน้า แต่อย่างไรก็ตาม เนื่องจากเป็นรูปแบบในการพึ่งพาข้อมูลที่เคยเกิด วิธีการเหล่านี้มีข้อจำกัดในสภาพแวดล้อมที่เปลี่ยนแปลงอย่างรวดเร็ว โดยเฉพาะเมื่อพบกับรูปแบบใหม่ที่ปรากฏขึ้นและไม่เคยมีมาก่อนหน้านี้ ซึ่งอาจทำให้การปรับใช้ทรัพยากรมีประสิทธิภาพที่ลดลงในช่วงเวลานั้น ในทางตรงกันข้ามซีสิสการ์ด (cSysGuard) ใช้วิธีการเรียนรู้ของเครื่องแบบผสมที่ใช้ประโยชน์จากจุดแข็งเฉพาะของโมเดลต่าง ๆ เพื่อบรรลุความสามารถในการทำนายที่มีประสิทธิภาพกับสภาพแวดล้อมคลาวด์ที่เปลี่ยนแปลงอย่างรวดเร็ว

หลายงานวิจัยได้เสนอวิธีการใช้โมเดลในรูปแบบผสมเพื่อที่รวบรวมตัวทำนายหลาย ๆ ตัวไว้ด้วยกัน ตัวอย่างเช่น งานวิจัยของ Kim et al. ได้บรรลุการทำนายโดยใช้โมเดลหลาย ๆ ตัว โดยใช้กลยุทธ์การหาค่าเฉลี่ยแบบถ่วงน้ำหนัก (Weighted Average) วิธีการนี้จะทำนายผลสุดท้ายที่มาจากการอิงผลของแต่ละโมเดลที่ไม่เท่ากัน โดยจะเชื่อถือกับโมเดลที่มีความแม่นยำที่สูงกว่าเป็นหลัก ซึ่งวิธีนี้จะให้ผลลัพธ์ที่แม่นยำกว่าการหาค่าเฉลี่ยทั่วไป และสามารถปรับทำนายให้เหมาะกับสภาพแวดล้อมนั้น ๆ เพื่อเพิ่มความยืดหยุ่นให้มากขึ้น แต่อย่างไรก็ตามตามที่หมวด 6.2.3 ที่ได้ระบุไว้ การพึ่งพาด้วยการหาค่าเฉลี่ยแบบถ่วงน้ำหนักอย่างเดียวนั้นได้ให้มีความแม่นยำที่น้อยกว่าเมื่อเทียบกับซีสิสการ์ด (cSysGuard) อีกทั้ง งานวิจัยของ Mehmood et al. เสนอการผสมโมเดลโดยใช้รูปแบบ Stacking ซึ่งใช้ Decision Tree เป็นเมต้าโมเดลเพื่อสรุปการทำนายจากโมเดลตัวอื่น ๆ ถึงแม้การใช้โมเดลอีกชั้นหนึ่งเพื่อสรุปผลลัพธ์จะมีความแม่นยำที่มากกว่าเมื่อเทียบกับการใช้โมเดลแบบเดี่ยว ผลการค้นพบของงานวิจัยเราบ่งชี้ว่าการใช้เมต้าโมเดลอันเดียวอาจไม่สามารถให้ประสิทธิภาพที่ดีได้ตลอดเวลา โดยเฉพาะอย่างยิ่งในสภาพแวดล้อมที่ไม่สามารถคาดเดาได้ รูปที่ 2.1 แสดงถึงประสิทธิภาพในการทำนายที่ผันผวนอยู่ตลอดเวลาของเมต้าโมเดลในการทำนายการใช้ CPU ของระบบตามช่วงเวลาต่าง ๆ ข้อมูลบ่งชี้ว่าเมต้าโมเดลที่มีประสิทธิภาพที่สุดมักจะเปลี่ยนไปเป็นระยะ ๆ ซึ่งนำเสนอความท้าทายในการเลือกเมต้าโมเดลที่ดีที่สุดสำหรับการใช้งานในสภาพแวดล้อมคลาวด์ที่เปลี่ยนแปลงอยู่ตลอดเวลา ในงานวิจัยนี้ ซีสิสการ์ด (cSysGuard) ได้ใช้วิธีการผสมโมเดลในรูปแบบ Stacking หลายชั้นพร้อมวิธีการหาค่าเฉลี่ยตามน้ำหนักเพื่อรวมผลการทำนายของทุกโมเดลในการให้ผลลัพธ์ที่มีประสิทธิภาพมากกว่าวิธีใดวิธีหนึ่งที่กล่าวมาข้างต้น



รูปที่ 2.1 RMSE ของเมต้าโมเดลต่าง ๆ ที่ได้ทำนายการใช้ CPU ตามช่วงเวลาในคลาวด์

เทคนิคการเรียนรู้เชิงลึกใช้กันอย่างกว้างขวางในหลายการศึกษา ถึงแม้เทคนิคการเรียนรู้เชิงลึกจะเป็นเครื่องมือที่มีศักยภาพสำหรับการวิเคราะห์การทำนาย แต่ทว่าข้อมูลที่มีรูปแบบเรียบง่าย เทคนิคเหล่านี้ก็ยังจำเป็นต้องใช้ทรัพยากรในการคำนวณเป็นอย่างมากเพื่อให้ได้ผลลัพธ์ที่แม่นยำ ในทางตรงกันข้ามซีสิสการ์ด (cSysGuard) ได้นำเสนอกลยุทธ์ที่ให้ผู้ใช้สามารถปรับสมดุลระหว่างการใช้ทรัพยากรในการคำนวณและความแม่นยำในการทำนาย โดยสามารถเลือกโมเดลที่เหมาะสมตามลักษณะของข้อมูลนั้น ๆ และมากไปกว่านั้น ถึงแม้การใช้ทรัพยกรในการคำนวณจะไม่ถูกพิจารณา ซีสิสการ์ด (cSysGuard) ก็ยังให้ผลทำนายที่แม่นยำกว่าเทคนิคการเรียนรู้เชิงลึกอีกด้วย เนื่องจากเป็นเทคนิคที่เกิดจากการรวบรวมของโมเดลที่หลากหลายเพื่อมาใช้ในการทำนายที่มีประสิทธิภาพ

ในงานวิจัยที่เกี่ยวกับการทำนายสถานะของระบบโดยใช้ข้อมูลจาก Google Trace นั้นมีอย่างมากมาย ตัวอย่างเช่น งานวิจัยของ Asmawi, T. et al. และ Jassas, M.S. et al. ได้ใช้โมเดลเพื่อทำนายโอกาสล้มเหลวของเซิร์ฟเวอร์โดยใช้ตัวชี้วัดสากลในการป้อนข้อมูล เช่น การใช้งาน CPU หน่วยความจำ (Memory) และการเก็บข้อมูลดิสก์ (Disk Storage) ผลการศึกษาของพวกเขาเปิดเผยความสัมพันธ์ระหว่างตัวชี้วัดเหล่านี้และการล้มเหลวของเซิร์ฟเวอร์ ในทำนองเดียวกัน งานวิจัยนี้ได้นำการใช้งาน CPU และหน่วยความจำเพื่อทำนายโอกาสที่จะล้มเหลวของระบบ แต่อย่างไรก็ตาม ในขณะที่หลายการศึกษาเน้นการใช้งานดิสก์เป็นตัวชี้วัดสำคัญในการทำนาย

ซึ่งบ่อยครั้งมักถูกอ้างว่าเป็นปัจจัยสำคัญที่สุดในงานวิจัยต่าง ๆ แต่ทว่าตัวชี้วัดนี้ไม่สอดคล้องกับลักษณะของเซิร์ฟเวอร์ที่ไม่ถือข้อมูลอย่างที่งานวิจัยเราได้นำมาเป็นขอบเขตของการศึกษา ซึ่งจะมีการใช้งานดิสก์ที่น้อยมากหรือไม่มีเลย เพราะฉะนั้น งานวิจัยนี้จะไม่นำมาเป็นข้อมูลในการป้อนโมเดล เพื่อหลีกเลี่ยงประสิทธิภาพในการทำนายที่ลดลง แต่อย่างไรก็ตาม เพื่อที่จะคงรักษาประสิทธิภาพในการทำนายที่ดี งานวิจัยนี้ได้เพิ่มตัวชี้วัดมากขึ้นเพื่อมาทดแทนข้อมูลที่หายไป เช่น เน็ตเวิร์คแบนด์วิดท์ (Network Bandwidth) จำนวนการทำธุรกรรมต่อวินาที (Transactions Per Second หรือ TPS) และเวลาตอบสนองของคำขอโดยเฉลี่ย (Average Response Time)

นอกจากนี้ งานวิจัยของ Liu X. et al. เสนอวิธีการทำนายโอกาสล้มเหลวของระบบเซิร์ฟเวอร์บนคลาวด์โดยใช้ข้อมูลจากตัวชี้วัดจากฮาร์ดแวร์ ตัวอย่างเช่น สถานะเคอร์เนล (Kernel Status) และไฟล์บันทึกในระบบ (System Logs) อย่างไรก็ตาม การใช้ข้อมูลที่มีโครงสร้างไม่ชัดเจน (Unstructured Data) จะก่อให้เกิดความซับซ้อนในการทำนายเพราะอาจจะพบข้อมูลที่ไม่เกี่ยวข้องอยู่บ่อยครั้ง อีกทั้ง แต่ละระบบปฏิบัติการ (Operating System หรือ OS) ก็มักจะมีรูปแบบของข้อมูลเหล่านี้ที่ไม่เหมือนกัน ในทางตรงกันข้าม งานวิจัยของเราได้ใช้ตัวชี้วัดที่สามารถเข้าถึงได้อย่างง่ายดายในทุกสถาปัตยกรรม เช่น CPU หน่วยความจำ เน็ตเวิร์คแบนด์วิดท์ จำนวนการทำธุรกรรมต่อวินาที และเวลาตอบสนองของคำขอโดยเฉลี่ย ตัวชี้วัดเหล่านี้มักจะหาได้ง่ายและเป็นข้อมูลที่เหมาะสม เพราะมักจะมีรูปแบบลักษณะที่มั่นคง ซึ่งเป็นที่เหมาะสมอย่างมากในการใช้มาเป็นข้อมูลให้กับโมเดลต่าง ๆ

งานวิจัยของ Hioual O. et al. นำเสนอวิธีการใหม่ในการทำนายโอกาสล้มเหลวของเซิร์ฟเวอร์โดยพิจารณาจากองค์ประกอบของฮาร์ดแวร์ ตัวอย่างเช่น อุณหภูมิของเครื่องเซิร์ฟเวอร์ เพื่อใช้ในการตัดสินใจการแบ่งภาระงานคำขอของผู้ใช้บริการไปยังเซิร์ฟเวอร์ต่าง ๆ เพื่อให้มีความยืดหยุ่นที่มากขึ้น อย่างไรก็ตาม ลักษณะที่เน้นการตรวจสอบสภาพของฮาร์ดแวร์อาจจำกัดความสามารถในการตรวจสอบของผู้ใช้คลาวด์เป็นหลัก เพราะเซิร์ฟเวอร์ส่วนใหญ่บนคลาวด์มักจะทำงานผ่านเครื่องเสมือน (Virtual Machine) ที่ถูกสร้างผ่านตัวเครื่องฮาร์ดแวร์อีกที ซึ่งเป็นไปได้ยากสำหรับผู้ใช้คลาวด์เป็นหลักที่จะสามารถตรวจสอบสถานะของเซิร์ฟเวอร์ผ่านเครื่องฮาร์ดแวร์โดยตรงได้ ในทางตรงกันข้าม งานวิจัยนี้จะวิเคราะห์ข้อมูลที่จากข้างในระบบโดยตรง ซึ่งข้อมูลเหล่านี้เป็นข้อมูลที่เข้าถึงได้ง่ายกับผู้ใช้ต่าง ๆ ไม่ว่าจะเป็นผู้ใช้บนคลาวด์หรือเครื่องฮาร์ดแวร์โดยตรง

2.3 ทฤษฎีที่เกี่ยวข้อง

ในส่วนนี้จะอธิบายทฤษฎีที่เกี่ยวข้องกับงานวิจัยอย่างครอบคลุม เพื่อให้เข้าภาพรวมของงานวิจัยที่เราได้นำทฤษฎีต่าง ๆ ที่ใช้ในการสร้างและประเมิน รวมถึงการจัดการข้อมูลของซีสิสการ์ด (cSysGuard) ให้ออกมาอย่างมีประสิทธิภาพมากที่สุด

2.3.1 Low-pass Butterworth Filter

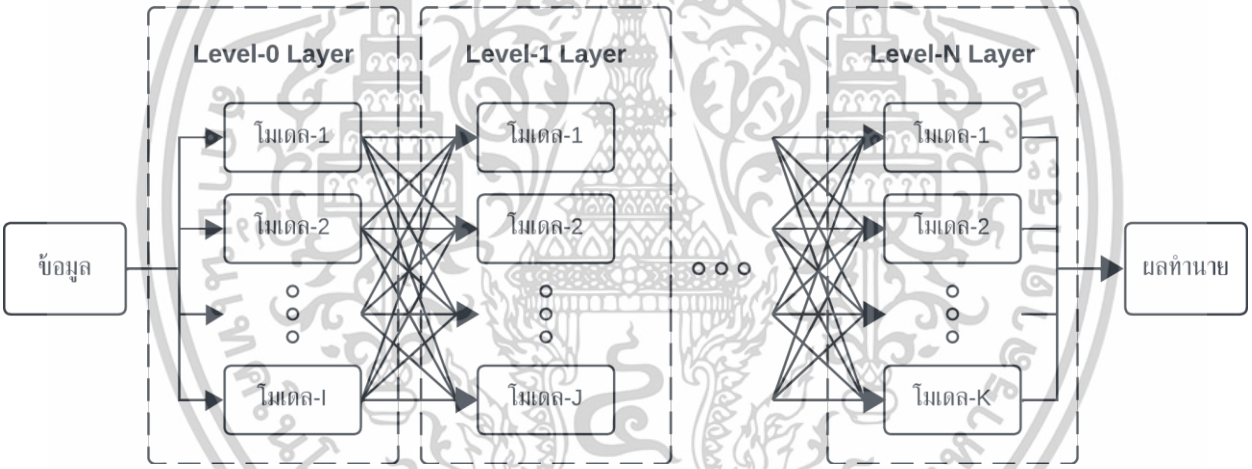
ตัวกรองความถี่ต่ำ Butterworth เป็นความคิดหลักการในการประมวลผลสัญญาณที่ช่วยลดเสียงรบกวนความถี่สูงแต่ยังรักษาความสมบูรณ์ของสัญญาณไว้ โดยที่มีลักษณะเด่นด้วยการตอบสนองขนาดที่เรียบสูงสุด (Maximally Flat Magnitude Response) และแถบผ่านที่ไม่มีการเปลี่ยนแปลงของสัญญาณ (Ripple-free Passband) ตัวกรองนี้จะตอบสนองความถี่ที่สม่ำเสมอไปจนถึงความถี่ตัด (Cut-off Frequency) ทำให้เหมาะสมกับแอปพลิเคชั่นที่ต้องการผลลัพธ์ที่ไม่บิดเบือน ฟังก์ชันการถ่ายโอนซึ่งถูกกำหนดโดยระดับของตัวกรอง (Filter Order) มีอิทธิพลต่อการลดลงของความชันที่จุดตัดความถี่ ตัวกรองที่มีระดับสูงจะทำให้การเปลี่ยนจากแถบผ่านไปยังแถบหยุด (Passband-to-stopband) ได้เร็วขึ้น แต่ก็เพิ่มความซับซ้อนและความเสี่ยงของการบิดเบือนของสัญญาณ ในการวิเคราะห์ข้อมูล ตัวกรองทำหน้าที่ลดเสียงรบกวนและความแปรปรวนของข้อมูล ซึ่งส่งเสริมความน่าเชื่อถือของผลการวิเคราะห์ให้ดียิ่งขึ้น

2.3.2 Synthetic Minority Oversampling Technique (SMOTE)

เทคนิคการสร้างตัวอย่างเทียมสำหรับกลุ่มข้อมูลน้อย หรือ SMOTE จะจัดการกับข้อมูลที่ไม่สมดุล (Imbalanced Dataset) โดยจะสร้างตัวอย่างเทียมให้กับกลุ่มข้อมูลที่มีปริมาณน้อยกว่ากลุ่มอื่น ๆ ให้มีจำนวนที่ใกล้เคียงกัน ซึ่งทำให้การกระจายข้อมูลดีขึ้นต่อการใช้งานโมเดลต่าง ๆ เทคนิคนี้จะสร้างตัวอย่างใหม่ผ่านการแทรกระหว่างตัวอย่างของกลุ่มข้อมูลน้อยและเพื่อนบ้านของมัน เพื่อเพิ่มความหลากหลายและลดความเสี่ยงของ Overfitting ได้อย่างมีประสิทธิภาพ อย่างไรก็ตาม ประสิทธิภาพของการทำนายก็ยังขึ้นอยู่กับการปรับแต่งพารามิเตอร์อย่างระมัดระวัง รวมถึงจำนวนเพื่อนบ้านและปริมาณข้อมูลเทียมที่สร้างขึ้น ความหลากหลายของ SMOTE ทำให้สามารถใช้งานได้ในหลากหลายสาขา ตั้งแต่การตรวจจับการฉ้อโกงไปจนถึงการวินิจฉัยทางการแพทย์ ซึ่งมีส่วนสำคัญต่อการสร้างแบบจำลองในการทำนายให้เท่าเทียมและแม่นยำมากขึ้น

2.3.4 Ensemble Stacking Models

Stacking ในโมเดลผสมเป็นวิธีการหนึ่งที่จะใช้หลาย ๆ โมเดลมาช่วยในการทำนายผล โดยโครงสร้างของมันจะประกอบไปด้วยหลายชั้น ซึ่งแต่ละชั้นจะรู้จักกันในชื่อว่า Level-N ซึ่ง N คือเลขของชั้นนั้น ๆ ในแต่ละชั้นก็จะมีโมเดลที่หลากหลาย ช่วงเริ่มต้น โมเดลที่อยู่ในชั้นแรก หรือ Level-0 จะประมวลผลจากข้อมูลดิบ และให้ผลของการทำนาย ต่อจากนั้น ในชั้นถัดไปจะนำผลลัพธ์ที่ได้มาจากโมเดลในชั้นแรกมาเป็นข้อมูลให้กับโมเดลในชั้นนั้นเพื่อมาวิเคราะห์และทำนายต่อ โดยหลักการนี้จะถูกทำซ้ำไปเรื่อยจนกว่าจะครบถ้วน ในชั้นที่สูงขึ้นจะรวบรวมและปรับปรุงผลลัพธ์ที่ได้มาจากชั้นก่อนหน้าเพื่อลดความลำเอียงและความแปรปรวนที่อาจเกิดขึ้นจากการใช้โมเดลแบบเดี่ยว ตัวอย่างในรูปที่ 2.2 แสดงให้เห็นถึงสถาปัตยกรรมในการใช้ Ensemble Stacking Models โดยนำเสนอจากชั้น Level-0 ไปยังชั้น Level-N



รูปที่ 2.2 สถาปัตยกรรมการใช้โมเดลผสมที่มีรูปแบบ Stacking (Ensemble Stacking Models)

2.3.5 การวิเคราะห์การถดถอย

Autoregressive Integrated Moving Average (ARIMA) เป็นสิ่งจำเป็นสำหรับการวิเคราะห์อนุกรมเวลา โดยมีประสิทธิภาพในการตรวจจับการทำนายแบบไม่มีฤดูกาล (Non-Seasonal) ด้วยคุณสมบัติการถดถอยอัตโนมัติ (Autoregressive) และเส้นค่าเฉลี่ยเคลื่อนที่ (Moving Averages) ในทางกลับกัน Seasonal Autoregressive Integrated Moving Average (SARIMA) ซึ่งถูกพัฒนาต่อยอดมาจาก ARIMA โดยมีการนำความ

แปรผันตามฤดูกาล (Seasonality) มาช่วยในการทำนาย ซึ่งเป็นที่เหมาะสมกับการทำนายที่มีรูปแบบแปรผันตามฤดูกาลในชุดข้อมูลที่เป็นอนุกรมเวลา

Exponential Smoothing (ETS) เป็นเทคนิคการทำนายที่ถูกใช้มาอย่างยาวนานซึ่งกำหนดน้ำหนักที่ลดลงแบบ Exponential ตามเวลา เทคนิคนี้ออกแบบมาเพื่อตรวจจับและจัดการกับแนวโน้มและความแปรผันตามฤดูกาลในชุดข้อมูลอนุกรมเวลา ETS มีความเรียบง่ายและมีประสิทธิภาพในการทำนายระยะสั้นเนื่องจากสามารถปรับตัวได้ดีตามการเปลี่ยนแปลงในข้อมูล

Linear Regression (LR) เป็นโมเดลพื้นฐานในการวิเคราะห์ทางสถิติ โดยเหมาะสำหรับการทำนายค่าของตัวแปรตาม (Dependent Variable) จากตัวแปรอิสระหนึ่งตัวหรือหลายตัว เทคนิคนี้จะสมมุติความสัมพันธ์แบบเส้นตรงระหว่างตัวแปรที่เกี่ยวข้อง ใน LR แบบเรียบง่ายจะใช้ตัวแปรอิสระเพียงตัวเดียว ขณะที่ LR แบบหลายตัวแปรจะขยายไปถึงการใช้ตัวแปรอิสระหลายตัว โมเดลนี้มีประสิทธิภาพเป็นพิเศษในสถานการณ์ที่มีความสัมพันธ์แบบเส้นตรง โดยมีสัมประสิทธิ์บ่งชี้ถึงอิทธิพลของแต่ละตัวแปรต่อตัวแปรตาม

Random Forest (RF) สร้าง Decision Trees หลายต้นเพื่อปรับปรุงความแม่นยำในการทำนาย โดย RF ใช้ประโยชน์จากการ Bagging หรือนั้นก็คือกระบวนการรวมผลลัพธ์จากโมเดลหลาย ๆ ตัวที่ถูกฝึกอิสระต่อกัน และการสุ่มคุณลักษณะ (Feature Randomness) เพื่อสร้างแต่ละต้นไม้ เทคนิคนี้ช่วยลดการเกิด Overfitting และเพิ่มความหลากหลายในการทำนาย วิธีนี้มีประสิทธิภาพในการจัดการกับชุดข้อมูลที่ซับซ้อนและให้ประสิทธิภาพที่เชื่อถือได้ในการใช้งานที่หลากหลาย

Feedforward Neural Networks (FNN) มีโครงสร้างที่ชัดเจนประกอบด้วยชั้น Input ชั้น Hidden และชั้น Output โดยมีการเชื่อมต่อระหว่างนิวรอนตามลำดับ โครงสร้างนี้เชี่ยวชาญในการเข้าใจและทำนายรูปแบบที่ซับซ้อนที่มีความหลากหลายและประสิทธิภาพสูง FNN มีความเหมาะสมอย่างยิ่งในการจัดการกับชุดข้อมูลขนาดใหญ่ โดยความสามารถในการจำลองความสัมพันธ์แบบไม่เชิงเส้นระหว่าง Input และ Output ทำให้ FNN ถูกใช้งานอย่างแพร่หลายในหลายสาขา เช่น การวิเคราะห์ผลเสียงหรือภาพ

Convolutional Neural Networks (CNN) เป็นโมเดลที่รู้จักดีในด้านประสิทธิภาพการประมวลผลภาพ อีกทั้ง CNN สามารถวิเคราะห์และทำนายชุดข้อมูลที่เป็นอนุกรมเวลาด้วยความสามารถในการให้ความสำคัญกับ

รูปแบบผ่านชั้น Convolutional ซึ่งทำให้สามารถใช้ประโยชน์กับงานที่หลากหลายเกินกว่าการวิเคราะห์ข้อมูลภาพ ตัวอย่างเช่น การประมวลผลเสียงและภาษา เป็นต้น

Temporal Convolutional Networks (TCN) เป็นโมเดลที่รวมจุดแข็งของ Convolutional Neural Networks (CNN) กับความไวต่อเวลาที่เหมาะสมสำหรับการทำ Sequence Modeling โดย TCN ได้รับการออกแบบมาเพื่อจัดการกับลำดับข้อมูลที่ยาวโดยใช้การคำนวณที่ไม่อนุญาตให้ข้อมูลจากอนาคตกระทบข้อมูลในอดีต ทำให้ TCN เหมาะสมอย่างยิ่งในการทำนายชุดข้อมูลอนุกรมเวลาที่ซับซ้อน

Recurrent Neural Networks (RNN) เป็นโมเดลที่สำคัญในการวิเคราะห์ลำดับข้อมูล โดยมีคุณสมบัติเฉพาะที่สามารถจดจำข้อมูลในลำดับตามเวลา ทำให้เหมาะสำหรับการประมวลผลข้อมูลที่มีลักษณะต่อเนื่องเช่น ข้อความ เสียง และชุดข้อมูลที่เป็นอนุกรมเวลา RNN มีข้อจำกัดในการจำข้อมูลระยะยาวเนื่องจากปัญหาของแกรเดียนท์ที่หายไป (Vanishing Gradient) ซึ่งส่งผลให้ประสิทธิภาพในการเรียนรู้ลดลงเมื่อข้อมูลมีความยาวมาก ด้วยเหตุนี้ จึงนำไปสู่รูปแบบใหม่ที่ถูกพัฒนาเพิ่ม นั้นก็คือ Gated Recurrent Unit (GRU) และ Long Short-Term Memory (LSTM) โดย GRU มีกลไกการประตู (Gates) ที่เรียบง่ายเพื่อจัดการกับปัญหาของแกรเดียนท์ที่หายไป (Vanishing Gradient) โมเดลนี้สร้างสมดุลระหว่างประสิทธิภาพการคำนวณและความสามารถในการเรียนรู้ข้อมูลระยะยาว ข้อได้เปรียบนี้ทำให้ GRU เหมาะสำหรับการทำงานที่ต้องการความสามารถในการจำแนกและเข้าใจลำดับข้อมูลที่ซับซ้อน เช่น การจำลองภาษาและการวิเคราะห์ชุดข้อมูลอนุกรมเวลาที่มีความละเอียดอ่อน ในทางกลับกัน LSTM ใช้เทคนิคการควบคุมด้วยประตู (Gates) เพื่อคัดเลือกข้อมูลที่จะเก็บหรือลืม ซึ่งทำให้โมเดลนี้เหมาะกับการเรียนรู้ข้อมูลลำดับระยะยาว (Extended Sequences) ความสามารถพิเศษนี้ทำให้ LSTM เหมาะสำหรับการจัดการกับงานที่มีความซับซ้อนและต้องการความแม่นยำในการจำลองข้อมูลที่ยาวนาน เช่น การสร้างข้อความอัตโนมัติ, การทำนายชุดข้อมูลอนุกรมเวลาที่ซับซ้อน, และการแปลภาษา

2.3.6 การวิเคราะห์การจำแนกประเภท

Decision Tree เป็นที่รู้จักในการทำให้กระบวนการตัดสินใจที่ซับซ้อนให้เป็นเรื่องง่าย โมเดลนี้จะมีโครงสร้างที่คล้ายต้นไม้โดยการแบ่งชุดข้อมูลออกเป็นสาขาตามค่าคุณลักษณะ วิธีนี้แบ่งข้อมูลออกเป็นชุดย่อยที่เล็กลงและจัดการได้ง่ายขึ้น ซึ่งสามารถจัดการกับความสัมพันธ์ที่ไม่เป็นเชิงเส้นระหว่างตัวแปรได้อย่างมีประสิทธิภาพ และส่งเสริมกระบวนการตัดสินใจแบบโปร่งใสและแบบมีชั้นเชิง

Support Vector Machine (SVM) สามารถใช้ในงานทั้งการจำแนกประเภทและการวิเคราะห์การถดถอย โดย SVM มีประสิทธิภาพสูงในพื้นที่มิติสูง เนื่องจากมีหลักการของการขยายระยะห่างสูงสุด (Margin Maximization) ซึ่งสร้างระนาบที่เรียกว่า Hyperplane เพื่อแยกข้อมูลอย่างชัดเจน ประสิทธิภาพนี้ทำให้ SVM มีความแม่นยำสูงแม้ในสภาพแวดล้อมที่มีจำนวนมิติของข้อมูลมาก และช่วยลดปัญหา Overfitting ได้

K-Nearest Neighbors (KNN) เป็นหลักการที่ใช้วิธีการเรียนรู้ตามตัวอย่าง (Instance-based Learning) ในการจำแนกข้อมูล โดยการวิเคราะห์และนับคลาสที่ปรากฏมากที่สุดจาก “K” เพื่อนบ้านที่ใกล้ที่สุด อัลกอริทึมนี้พึ่งพาการวัดความคล้ายคลึงผ่านระยะทาง ซึ่งมักใช้ระยะทางแบบยุคลิด (Euclidean Distance) จำนวน “K” และการเลือกระยะทางมีผลต่อความแม่นยำของการทำนาย ทำให้ KNN เป็นเทคนิคที่สามารถปรับใช้ได้กับชุดข้อมูลต่าง ๆ ได้อย่างมีประสิทธิภาพ และสามารถจัดการกับการเสมอกันของคะแนนโหวตในการจำแนกคลาสได้

Logistic Regression มีความสำคัญสำหรับการจำแนกประเภทแบบเลขฐานสองในการเรียนรู้ของเครื่อง โดยใช้ฟังก์ชัน Logistic ที่แปลงค่าทำนายเป็นความน่าจะเป็นระหว่าง 0 ถึง 1 ซึ่งช่วยให้สามารถรองรับความสัมพันธ์ระหว่างตัวแปรที่ไม่เป็นเส้นตรง โมเดลนี้ใช้วิธีการประมาณค่าความน่าจะเป็นสูงสุด (Maximum Likelihood Estimation) เพื่อหาสัมประสิทธิ์ที่ดีที่สุด ทำให้เป็นเครื่องมือที่มีบทบาทสำคัญในหลายสาขา เช่น การวินิจฉัยทางการแพทย์และการตรวจจับสแปม

Neural Networks ได้รับแรงบันดาลใจจากโครงสร้างของเซลล์ประสาทในสิ่งมีชีวิต ประกอบด้วยชั้นของเซลล์ประสาทเทียมหรือ Node ที่เชื่อมต่อกันอย่างซับซ้อน โครงสร้างเริ่มต้นด้วยชั้น Input ซึ่งรับข้อมูลเข้า ตามด้วยชั้น Hidden หนึ่งชั้นหรือมากกว่าที่ทำหน้าที่ประมวลผลข้อมูลด้วยการคำนวณผลรวมถ่วงน้ำหนักของข้อมูลนำเข้า และใช้ฟังก์ชัน Activation เพื่อเพิ่มความไม่เป็นเส้นตรงในการประมวลผล โมเดลจะสิ้นสุดที่ชั้น Output ที่ส่งผลการประมวลผลออกมา

2.3.7 วิธีการประเมินผล

Root Mean Square Error (RMSE) เป็นเมตริกสำคัญในการประเมินการทำนายที่เกี่ยวกับการถดถอย โดยคำนวณรากที่สองของค่าเฉลี่ยของความแตกต่างยกกำลังสองระหว่างค่าที่ทำนายและค่าจริง โดยการยกกำลังสองของ Error ก่อนการคำนวณค่าเฉลี่ยจะทำให้สามารถเห็นถึงความแตกต่างของ Error ที่ชัดเจน ทำให้มีการ

ตอบสนองต่อข้อผิดพลาดในการทำนายที่เล็กน้อยที่สูง สูตรของ RMSE ระบุไว้ในสมการที่ 2.1 โดยที่ y_k แทนค่าจริง $\hat{y}_k$ หมายถึงค่าที่ทำนายไว้ และ n หมายถึงจำนวนของการสังเกต โดย RMSE ที่ต่ำกว่าบ่งบอกถึงความแม่นยำในการทำนายที่ดีกว่าเมื่อเปรียบเทียบกับ RMSE ที่สูงกว่า

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y}_k)^2} \tag{2.1}$$

F1-Score เป็นเมตริกหลักในการประเมินผลการทำนายที่เกี่ยวข้องกับการจำแนกประเภท โดยคำนวณจากค่าเฉลี่ยฮาร์โมนิก (Harmonic Mean) ระหว่าง Precision และ Recall โดย Precision คืออัตราส่วนของ True Positives ต่อการทำนายผลบวกทั้งหมด (Positive Predictions) ซึ่งบ่งบอกถึงความแม่นยำในการจำแนกประเภทบวก ในขณะที่ Recall คืออัตราส่วนของ True Positives ต่อผลบวกจริงทั้งหมด (Actual Positives) ซึ่งสะท้อนถึงความสามารถในการระบุเหตุการณ์ที่เกี่ยวข้องทั้งหมด สูตรการคำนวณเหล่านี้มีรายละเอียดเพิ่มเติมในสมการที่ 2.2, 2.3 และ 2.4 สำหรับการคำนวณตามคลาส n

$$Precision_n = \frac{TP_n}{TP_n + FP_n} \tag{2.2}$$

$$Recall_n = \frac{TP_n}{TP_n + FN_n} \tag{2.3}$$

$$F1\text{-}Score_n = 2 \times \frac{Precision_n \times Recall_n}{Precision_n + Recall_n} \tag{2.4}$$

ในสมการ TP หมายถึง True Positives, FP หมายความว่า False Positives และ FN แสดงถึง False Negatives โดยที่คะแนน F1-Score 1 นั้นหมายถึงการที่มี Precision และ Recall ที่สมบูรณ์แบบ โดยบ่งบอกว่าการทำนายทั้งหมดนั้นแม่นยำและครบถ้วน ในทางตรงกันข้าม คะแนน 0 แสดงถึงความแม่นยำที่ต่ำที่สุด ซึ่งโมเดลไม่สามารถระบุค่า True Positive ใด ๆ ได้เลย

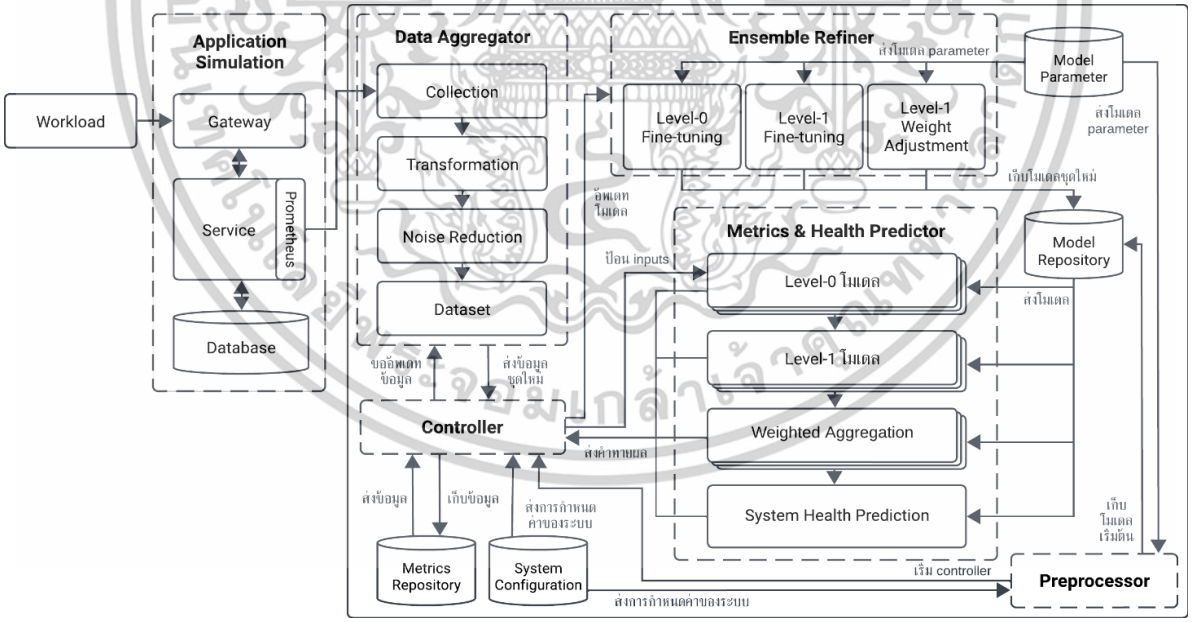
บทที่ 3

กระบวนการทำงานของซีสิสการ์ด (cSysGuard)

บทนี้จะอธิบายให้เห็นถึงภาพรวมอย่างละเอียดเกี่ยวกับวิธีการที่ใช้ในงานวิจัย โดยเน้นไปที่สถาปัตยกรรมของซีสิสการ์ด (cSysGuard) และการจัดการเตรียมข้อมูลเพื่อใช้ในการทำนายผล

3.1 สถาปัตยกรรมของซีสิสการ์ด (cSysGuard)

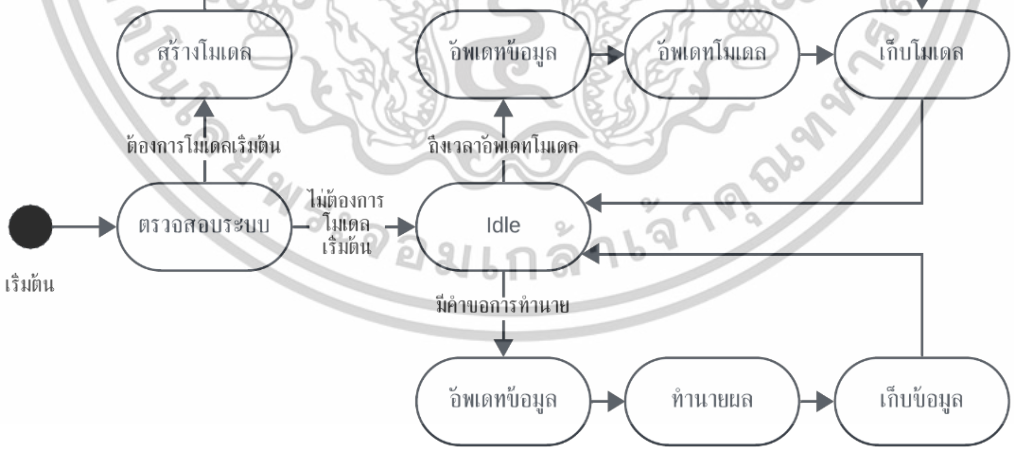
รูปที่ 3.1 แสดงถึงโครงสร้างของซีสิสการ์ด (cSysGuard) ซึ่งประกอบด้วย 6 ส่วนหลัก: การจำลองแอปพลิเคชั่น (Application Simulation), ตัวรวบรวมข้อมูล (Data Aggregator), ตัวควบคุม (Controller), ตัวทำนายตัวชี้วัดประสิทธิภาพและสถานะ (Metrics and Health Predictor), ตัวปรับปรุงโมเดล (Ensemble Refiner), และตัวเตรียมระบบ (Preprocessor) โดยแต่ละส่วนนั้นเป็นส่วนสำคัญของกระบวนการในการทำนายเป็นอย่างมาก



รูปที่ 3.1 สถาปัตยกรรมซีสิสการ์ด (cSysGuard)

3.1.1 ขั้นตอนการทำงานของซีสิสการ์ด (cSysGuard)

รูปที่ 3.2 แสดงถึงขั้นตอนการทำงานของซีสิสการ์ด (cSysGuard) ในช่วงเริ่มต้นระบบจะตัดสินใจว่าต้องการสร้างโมเดลใหม่หรือไม่โดยตรวจสอบผ่านค่าตัวแปรที่ถูกกำหนดไว้ในระบบ ในกรณีเช่นนั้น ระบบจะสร้างโมเดลโดยนำโมเดลมาฝึกด้วยข้อมูลที่เตรียมไว้ และหลังจากนั้นระบบจะเก็บโมเดลเหล่านี้ไว้สำหรับการวิเคราะห์ในอนาคตลงในที่เก็บโมเดลของระบบ แต่ทว่าถ้าระบบนั้นไม่มีความจำเป็นต้องสร้างโมเดลใหม่เนื่องจากมีโมเดลอยู่แล้ว ทางระบบจะข้ามขั้นตอนการเตรียมโมเดลที่กล่าวมาข้างต้น และดำเนินการขั้นตอนต่อไปโดยจะใช้โมเดลที่ถูกเก็บไว้ในระบบอยู่แล้วแทน ในขั้นตอนหลัก ระบบจะอยู่ในสถานะไม่ได้ใช้งาน (Idle) ไปจนกว่ามีคำของานปรากฏขึ้น ซึ่งแบ่งได้เป็น 2 กรณี ในกรณีที่หนึ่ง เมื่อถึงเวลาที่ต้องทำนาย ตัวระบบจะดึงข้อมูลจากแอปพลิเคชั่นที่เป็นเป้าหมายเพื่อเข้าถึงชุดข้อมูลล่าสุด และจากนั้นจะเปิดใช้งานตัวทำนาย โดยจะดึงโมเดลที่ถูกเก็บไว้ในระบบเพื่อสร้างและเก็บการผลของทำนาย ในอีกกรณีหนึ่ง เพื่อคงรักษาประสิทธิภาพให้เหมาะสมกับสภาพแวดล้อมปัจจุบัน ตัวระบบจะมีการปรับปรุงโมเดลอย่างต่อเนื่อง กระบวนการปรับปรุงจะเปิดใช้งานในช่วงเวลาที่กำหนดไว้ โดยเริ่มจาก การดึงข้อมูลล่าสุดจากแอปพลิเคชั่นที่เป็นเป้าหมายเพื่อนำมาใช้ในการปรับโมเดลให้สอดคล้องกับแนวโน้มของข้อมูลปัจจุบัน หลังจากการปรับปรุง โมเดลที่อัพเดตจะถูกเก็บไว้ในระบบแทนที่ตัวเก่าไว้สำหรับการวิเคราะห์ทำนายอนาคต



รูปที่ 3.2 กระบวนการทำงานของซีสิสการ์ด (cSysGuard)

3.1.2 การจำลองแอพพลิเคชั่น (Application Simulation)

เพื่อให้วิธีการเข้าถึงชุดข้อมูลในการศึกษานี้ องค์ประกอบนี้จะจำลองสภาพแวดล้อมแอปพลิเคชั่นคลาวด์โดยมีสามเครื่องเสมือน (Virtual Machine) ซึ่งแต่ละส่วนจะมีหน้าที่ที่ไม่เหมือนกัน โดยเครื่องแรกจะทำหน้าที่เป็น API Gateway ซึ่งมีหน้าที่ในการรับคำขอของผู้ใช้บริการแอปพลิเคชั่น และส่งคำขอไปยังส่วนที่สองนั้นก็คือแอปพลิเคชั่นเซอร์วิส (Application Service) เพื่อนำคำขอของผู้ใช้งานมาประมวลผล โดยแอปพลิเคชั่นส่วนนี้จะเขียนถูกด้วยภาษาโปรแกรมมิ่งด้วย Golang และมีรูปแบบที่ไม่ถือข้อมูล เพราะฉะนั้น เราจึงจำเป็นต้องมีส่วนประกอบที่สามที่จะเป็นฐานข้อมูล (Database) เพื่อใช้ไว้ในการเก็บข้อมูลแอปพลิเคชั่น ในการศึกษานี้งานวิจัยจะสังเกตการณ์ตัวชี้วัดประสิทธิภาพและสถานะของระบบในส่วนประกอบที่สองหรือนั้นก็คือตัวแอปพลิเคชั่นเซอร์วิส (Application Service) ซึ่งจะเป็นแหล่งข้อมูลสำคัญที่เราจะนำมาใช้ในการวิเคราะห์ทำนายผล ด้วยเหตุนี้ เราจึงได้ลงเครื่องมือในการตรวจสอบที่เรียกว่า Prometheus ซึ่งเป็นที่เก็บรวบรวมข้อมูลตัวชี้วัดของระบบเซิร์ฟเวอร์ที่เป็นเป้าหมาย นอกจากนี้ เรามีสคริปต์ที่เอาไว้จำลองคำขอของผู้ใช้บริการที่เปลี่ยนแปลงได้ตามเวลาเข้ามายังในแอปพลิเคชั่นที่เป็นเป้าหมายของงานวิจัย ซึ่งทำให้เราสามารถประเมินตัวชี้วัดและสถานะของระบบภายใต้เงื่อนไขที่หลากหลายและเปลี่ยนแปลงอยู่ตลอดเวลา

3.1.3 ตัวรวบรวมข้อมูล (Data Aggregator)

ส่วนประกอบนี้มีหน้าที่ในการอัพเดตและจัดเตรียมชุดข้อมูลเพื่อใช้ในการสร้างโมเดลและทำนายผล ในเริ่มแรก ตัวระบบจะรวบรวมข้อมูลที่เป็นตัวชี้วัดและสถานะจากแอปพลิเคชั่นที่เป็นเป้าหมาย และแปลงข้อมูลนั้นให้อยู่ในรูปแบบที่เหมาะสมเพื่อใช้ในขั้นตอนการพัฒนาโมเดลและทำนายผล หลังจากนั้น ทางระบบจะทำกระบวนการลดสัญญาณรบกวน (Noise Reduction) ในข้อมูล ข้อมูลที่ได้รับการปรับปรุงจะทำให้โมเดลสามารถให้ผลได้แม่นยำมากขึ้น ในท้ายที่สุด ระบบจะส่งข้อมูลที่ได้รับการประมวลผลไปยังตัวควบคุม (Controller) เพื่อใช้ในขั้นตอนต่อไป

3.1.4 ตัวควบคุม (Controller)

ส่วนประกอบนี้มีความสำคัญต่อการจัดการขั้นตอนการทำงานของแต่ละส่วนประกอบในซีสิสการ์ด (cSysGuard) เริ่มต้นด้วย ทางระบบจะไปอ่านการตั้งค่าที่ทางผู้ควบคุมนั้นได้ตั้งเอาไว้ เช่น กำหนดการปรับปรุง

หรือเวลาการทำนายผล โมเดลในระบบ ทางระบบจะได้รับข้อมูลล่าสุดอย่างสม่ำเสมอจากตัวรวมข้อมูล (Data Aggregator) เพื่อรับชุดข้อมูลล่าสุดมาใช้ในการทำนายหรือปรับปรุงของโมเดล ตัวควบคุมจะเริ่มต้นการทำนายโดยจะใช้โมเดลที่เก็บไว้มาทำนายตัวชี้วัดและสถานะของระบบ หลังจากนั้น ระบบจะจัดเก็บผลลัพธ์ทำนายและชุดข้อมูลที่ได้รับการอัพเดทลงในฐานข้อมูล นอกจากงานเหล่านี้แล้ว ตัวควบคุมยังจัดการการบำรุงรักษาโมเดลโดยส่งสัญญาณให้ตัวปรับปรุงปรับแต่งโมเดลตามกำหนดการอย่างสม่ำเสมอ การประสานงานของแต่ละองค์ประกอบที่เกิดจากตัวควบคุมนั้นมีผลต่อการดำเนินงานอย่างราบรื่นและน่าเชื่อถือ ซึ่งทำให้ซีสิสการ์ด (cSysGuard) มีประสิทธิภาพในการจัดการงานต่าง ๆ อย่างเป็นระบบและถูกต้องตามกำหนดอย่างอัตโนมัติ

3.1.5 ตัวทำนายตัวชี้วัดประสิทธิภาพและสถานะ (Metrics and Health Predictor)

ส่วนประกอบนี้ดูแลกระบวนการทำนายเป็นหลัก โดยจะประกอบไปด้วยการทำนายตัวชี้วัดประสิทธิภาพและการประเมินสถานะของเซิร์ฟเวอร์ เมื่อตัวควบคุมส่งสัญญาณมาให้เริ่มการทำนาย ตัวระบบจะเริ่มต้นด้วยการนำชุดข้อมูลของตัวชี้วัดประสิทธิภาพล่าสุดมาเป็นข้อมูลเพื่อป้อนโมเดลเพื่อที่จะทำนายตัวชี้วัดที่อาจเกิดขึ้นในอนาคต ในส่วนนี้ชุดโมเดลจะถูกเรียกว่าโมเดลการถดถอยแบบผสม (Ensemble Regression Model) ซึ่งประกอบไปด้วยสามชั้น นั้นก็คือ Level-0, Level-1 และ Weight Aggregation โดยรายละเอียดการทำงานในแต่ละชั้นจะถูกอธิบายเพิ่มเติมในบทที่ 4 นอกจากนี้ หลังจากเสร็จการทำนายตัวชี้วัด ตัวระบบจะรวบรวมผลลัพธ์จากแต่ละตัวทำนายที่รับผิดชอบในการทำนายตัวชี้วัดนั้น ๆ เพื่อจะนำมาเป็นข้อมูลให้กับการประเมินสถานะของระบบเซิร์ฟเวอร์ (System Health Prediction) ที่มีแนวโน้มจะเป็น ณ เวลานั้น ๆ โดยบทที่ 5 จะมีการอธิบายเพิ่มเติมในการทำงาน โดยผลลัพธ์ของแต่ละชั้นจะถูกส่งไปยังตัวควบคุมเพื่อจัดเก็บรักษาเพื่อนำมาใช้ปรับปรุงน้ำหนักหรือความน่าเชื่อถือของแต่ละโมเดลในชั้น Level-1 ส่วนประกอบนี้มีโครงสร้างที่ครอบคลุมและเป็นระบบเพื่อที่จะสามารถทำนายตัวชี้วัดประสิทธิภาพและสถานะของระบบอย่างอัตโนมัติ

3.1.6 ตัวปรับปรุงโมเดล (Ensemble Refiner)

เนื่องจากงานวิจัยนี้ได้นำเสนอถึงปัญหาของการใช้ทรัพยากรของซอฟต์แวร์ที่อยู่ในสภาพแวดล้อมที่ซับซ้อน เราจึงได้พัฒนาส่วนประกอบนี้เพื่อจะปรับปรุงประสิทธิภาพการพยากรณ์ของโมเดลการถดถอยเพื่อให้เข้ากับแนวโน้มของข้อมูลในปัจจุบันได้ดียิ่งขึ้น เมื่อถึงเวลาที่ถูกตั้งค่าไว้ ตัวปรับปรุงจะเริ่มโดยการนำโมเดลในชั้น Level-0 และ Level-1 มาฝึกใหม่ด้วยชุดข้อมูลล่าสุดที่ได้มาจากตัวรวบรวมข้อมูล (Data Aggregator) พร้อมกับ

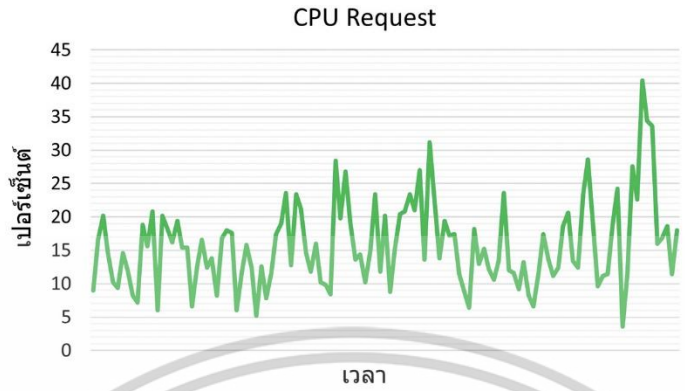
โมเดลพารามิเตอร์เดิมที่เก็บไว้ในระบบ อีกทั้งทางระบบจะทำการปรับน้ำหนักของโมเดลในชั้นที่ Level-1 อีกด้วย ในการปรับน้ำหนัก Level-1 ผลลัพธ์ในการคำนวณน้ำหนักของแต่ละโมเดลจะอิงตาม RMSE ที่โมเดลมีก่อนหน้านี้ โดยโมเดลที่มีประสิทธิภาที่สูงจะมีค่าน้ำหนักที่มากกว่าเมื่อเทียบกับโมเดลที่ให้ประสิทธิภาที่ต่ำกว่า (ซึ่งบทบาทของน้ำหนักหรือขั้นตอนการปรับแบบละเอียดจะถูกระบุอยู่ในบทที่ 4.3) หลังจากนั้น ทางระบบจะนำโมเดลและน้ำหนักที่ได้รับการปรับปรุงใหม่มาเก็บลงในฐานโมเดลและข้อมูล โดยขั้นตอนนี้จะถูกทำซ้ำอยู่เรื่อย ๆ เพื่อรักษาความแม่นยำของโมเดลในสภาพแวดล้อมที่อาจจะทำให้รูปแบบของข้อมูลเปลี่ยนแปลงอยู่เสมอ

3.1.7 ตัวเตรียมระบบ (Preprocessor)

ส่วนประกอบนี้จะเตรียมโมเดลที่ใช้ในระบบ โดยจะเริ่มต้นด้วยการดูรายการโมเดลที่ถูกตั้งค่าในระบบ จากนั้นทางระบบจะรวบรวมโมเดลโดยใช้พารามิเตอร์ที่ถูกกำหนดไว้และฝึกโมเดลเหล่านั้นด้วยชุดข้อมูลที่ถูกเตรียมไว้เบื้องต้น เมื่อเสร็จสิ้น โมเดลที่ได้รับการฝึกจะถูกเก็บไว้ในที่เก็บในฐานโมเดลเพื่อนำไปใช้ในอนาคต อย่างไรก็ตาม ขั้นตอนการเตรียมระบบเป็นแค่ทางเลือก ซึ่งสามารถข้ามได้หากตัวระบบนั้นมีโมเดลอยู่แล้ว

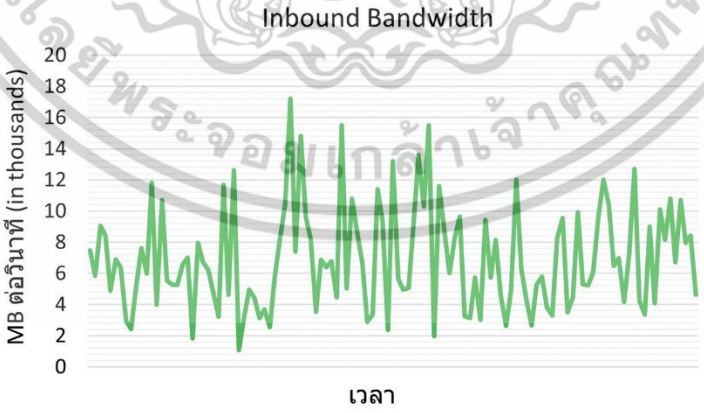
3.2 การเตรียมข้อมูล

ในการเก็บรวบรวมข้อมูล งานวิจัยได้ใช้สคริปต์เพื่อเข้าถึง Prometheus ที่ทำหน้าที่เก็บข้อมูลของแอปพลิเคชั่นที่เป็นเป้าหมาย ในการดึงข้อมูลครั้งนี้ ชุดข้อมูลมีประมาณอยู่ 8,000 จำนวน แต่ละอันจะถูกเก็บทุก ๆ 5 วินาที โดยตัวชี้วัดจะประกอบไปด้วยคำขอของ CPU และหน่วยความจำซึ่งแสดงเป็นเปอร์เซ็นต์, เน็ตเวิร์คแบนด์วิดท์ขาเข้าและออกซึ่งวัดเป็น Gigabyte หรือ Megabyte ต่อวินาที, จำนวนการทำธุรกรรมต่อวินาที และเวลาตอบสนองของคำขอโดยเฉลี่ยซึ่งเป็นวินาทีหรือมิลลิวินาที ดังที่แสดงในรูปที่ 3.3 – 3.8 ซึ่งแสดงตัวชี้วัดประสิทธิภาพของเซิร์ฟเวอร์ในช่วงเวลา 10 นาที นอกจากนี้ ตัวสคริปต์ยังเก็บสถานะของแอปพลิเคชั่น (System Health) แบบเรียลไทม์ ซึ่งถูกกำหนดโดยตรวจสอบจากรหัสการตอบกลับของ HTTP หรือที่เรียกกันว่า HTTP Response Code เมื่อสคริปต์ตรวจพบรหัสการตอบกลับที่เกี่ยวกับข้อผิดพลาดของเซิร์ฟเวอร์ (Server Error Code: 5xx) ระบบจะกำหนดสถานะของระบบเป็น Unhealthy มิฉะนั้น ระบบจะกำหนดสถานะเป็น Healthy แม้ว่าแอปพลิเคชั่นจะมีรูปแบบที่ไม่ถือข้อมูลซึ่งจะไม่การใช้ข้อมูลดิสก์บ่อยและงานวิจัยนี้จะไม่มาใช้เป็นข้อมูลในการป้อนหรือฝึกโมเดล ทางเรายังคงตรวจสอบข้อมูลดิสก์เพื่อให้แน่ใจได้ว่าตัวแอปพลิเคชั่นยังมีพฤติกรรมที่ไม่ถือข้อมูลอย่างถูกต้อง

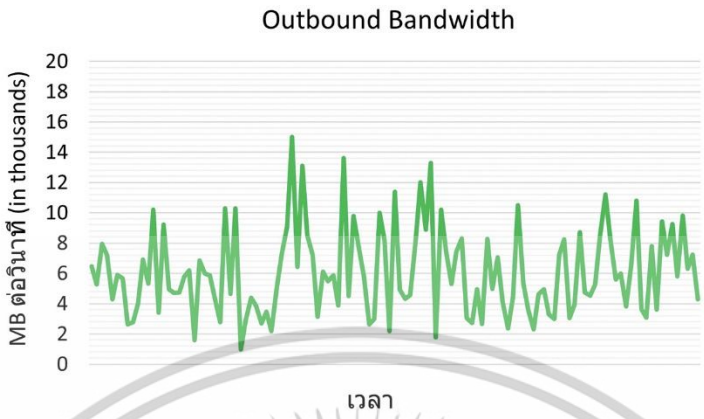


รูปที่ 3.3 การใช้งานของ CPU ในช่วง 10 นาที

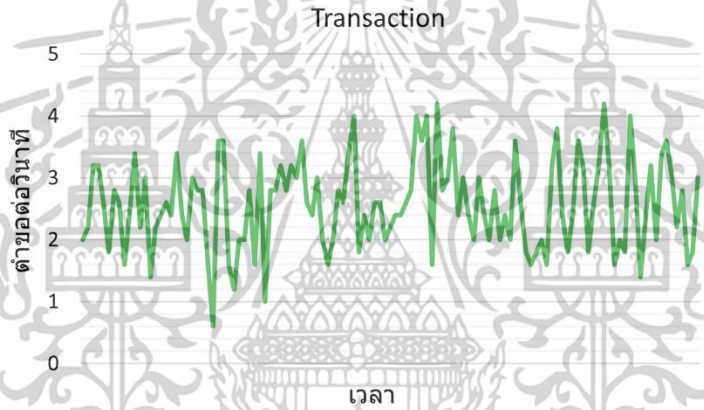
รูปที่ 3.4 การใช้งานของหน่วยความจำ (Memory) ในช่วง 10 นาที



รูปที่ 3.5 แบนด์วิดท์ขาเข้า (Inbound Bandwidth) ในช่วง 10 นาที



รูปที่ 3.6 แบนด์วิดท์ขาออก (Outbound Bandwidth) ในช่วง 10 นาที



รูปที่ 3.7 การทำธุรกรรมต่อวินาที (Transactions Per Second) ในช่วง 10 นาที

รูปที่ 3.8 เวลาตอบสนองเฉลี่ย (Average Response Time) ในช่วง 10 นาที

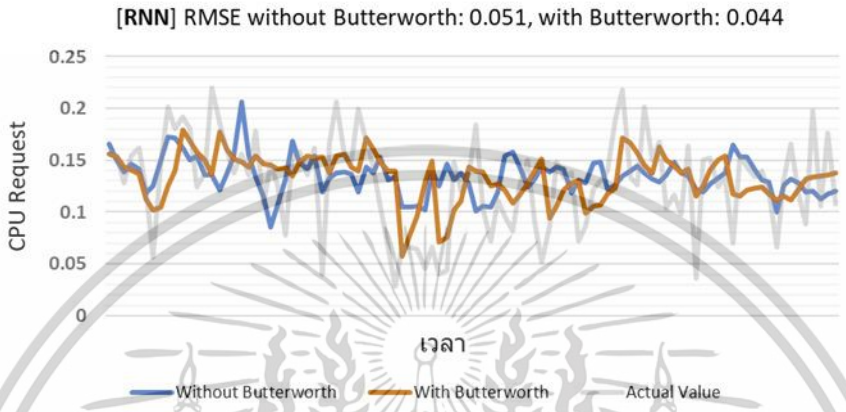
ต่อมา ระบบดำเนินการแปลงชุดข้อมูลเพื่อจัดรูปแบบข้อมูลให้เหมาะสมสำหรับการฝึกโมเดลและการวิเคราะห์ในส่วนต่อไป ตารางที่ 3.1 แสดงผลลัพธ์ของการแปลงข้อมูลที่ถูกปรับเปลี่ยนเพื่อใช้การวิเคราะห์ของโมเดล ระบบได้แปลงคำขอของ CPU และหน่วยความจำให้อยู่รูป Normalization เน็ตเวิร์คแบนด์วิดท์ขาเข้าและออกเป็น Megabyte ต่อวินาที (MB/s) และเวลาตอบสนองของคำขอโดยเฉลี่ยเป็นมิลลิวินาที (ms) อย่างไรก็ตาม ระบบไม่ได้เปลี่ยนคุณสมบัติของการทำธุรกรรมต่อวินาทีซึ่งถูกเก็บไว้อยู่ในรูปแบบเดิม ท้ายที่สุด ระบบได้แปลงสถานะของแอปพลิเคชั่นจากชื่อเดิมที่เป็นรูปแบบข้อความ (Healthy และ Unhealthy) ไปเป็นรูปแบบตัวเลขหรือ Binary โดยให้ Healthy เป็น 0 และ Unhealthy เป็น 1 ซึ่งทำให้การวิเคราะห์ของโมเดลง่ายขึ้น

ตารางที่ 3.1 ชุดข้อมูลแปลงของตัวชี้วัดประสิทธิภาพพร้อมกับสถานะของระบบใน 60 วินาที

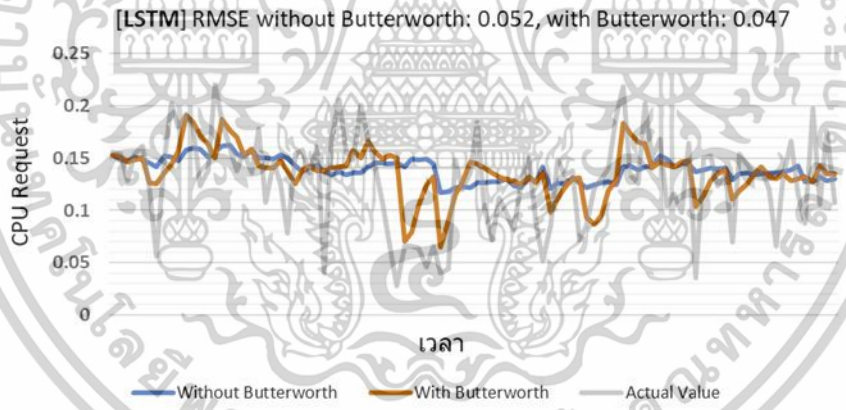
คำขอ CPU (normalized)	**คำขอหน่วยความจำ** (normalized)	**เน็ตเวิร์คแบนด์วิดท์ขาเข้า** (หน่วย: MB/s)	**เน็ตเวิร์คแบนด์วิดท์ขาออก** (หน่วย: MB/s)	**การทำธุรกรรมต่อวินาที**	**เวลาตอบสนองของคำขอโดยเฉลี่ย** (หน่วย: ms)	**สถานะ** (binary)
0.112	0.506	2870	2630	2.4	1890	0
0.142	0.545	5810	5050	2.2	1370	0
0.122	0.54	10600	9090	2.2	924	0
0.074	0.532	2430	2200	1.6	1970	0
0.116	0.551	5990	6020	3	2370	1
0.096	0.535	3660	3670	3.2	2920	1
0.136	0.555	7990	6650	2.4	3390	0
0.144	0.532	9030	7820	3	766	0
0.118	0.539	6340	5650	2	3200	0
0.084	0.535	2990	2730	3.4	3800	1
0.082	0.541	2180	1990	3.4	3250	1
0.136	0.566	5180	4790	1.8	1980	0

ต่อมา ระบบจะดำเนินกระบวนการลดสัญญาณรบกวน (Noise Reduction) ด้วยตัวกรองความถี่ต่ำ Butterworth เพื่อลดความไม่สอดคล้องของข้อมูล รูปที่ 3.9 – 3.12 แสดงถึงผลกระทบของการลด Noise รบกวน โดยการเปรียบเทียบการทำนายด้วย RMSE ของโมเดล RNN, LSTM, GRU และ ARIMA โดยใช้ข้อมูลสองแบบ นั้นก็คือข้อมูลที่ผ่านตัวกรองและข้อมูลดิบที่ได้มาจากแอปพลิเคชั่นที่เป็นเป้าหมายโดยตรง จากผลลัพธ์ที่ได้แสดงให้

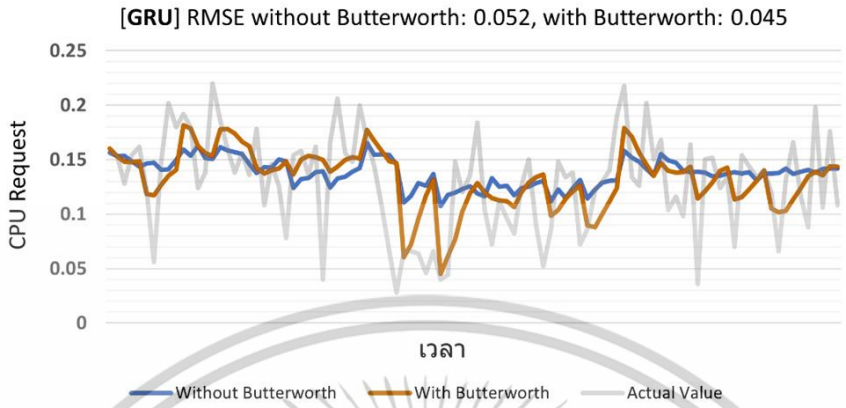
เห็นถึงการเพิ่มประสิทธิภาพในการทำนายที่ได้รับข้อมูลที่ผ่านการใช้ตัวกรอง ในท้ายที่สุด ระบบจะผลิตและเก็บชุดข้อมูลดิบและข้อมูลที่ได้รับการปรับแต่งไว้ลงในฐานข้อมูลเพื่อนำไปใช้ในการวิเคราะห์ในอนาคตถัดไป



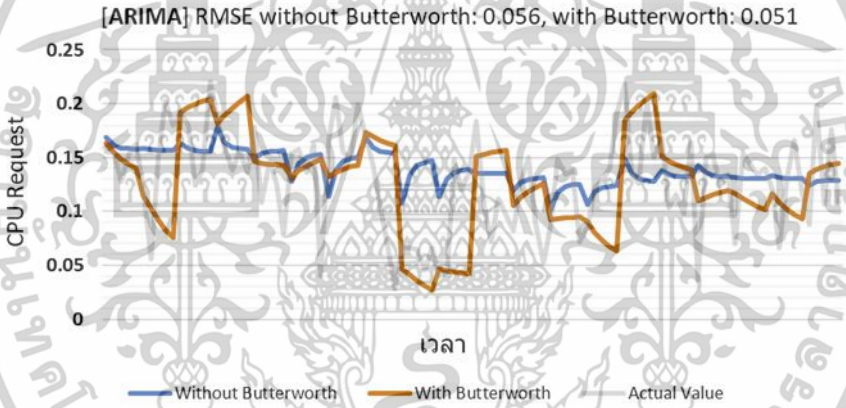
รูปที่ 3.9 การเปรียบเทียบ RMSE ของ RNN โดยใช้ข้อมูลที่ไม่ผ่านและผ่านตัวกรองความถี่ต่ำ Butterworth



รูปที่ 3.10 การเปรียบเทียบ RMSE ของ LSTM โดยใช้ข้อมูลที่ไม่ผ่านและผ่านตัวกรองความถี่ต่ำ Butterworth



รูปที่ 3.11 การเปรียบเทียบ RMSE ของ GRU โดยใช้ข้อมูลที่ไม่ผ่านและผ่านตัวกรองความถี่ต่ำ Butterworth



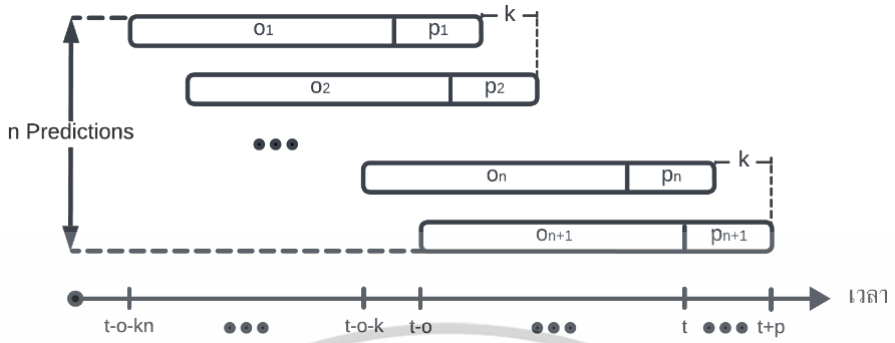
รูปที่ 3.12 การเปรียบเทียบ RMSE ของ ARIMA โดยใช้ข้อมูลที่ไม่ผ่านและผ่านตัวกรองความถี่ต่ำ Butterworth

บทที่ 4
กระบวนการทำนายตัวชี้วัดประสิทธิภาพของเซิร์ฟเวอร์

บทนี้จะอธิบายกระบวนการการทำงานของโมเดลที่มีรูปแบบถดถอยเพื่อทำนายตัวชี้วัดประสิทธิภาพอย่างละเอียด ซึ่งครอบคลุมไปถึงการทำงานของโมเดลในระดับชั้น Level-0 ถึง Level-1 และการรวมผลลัพธ์ด้วยการหาค่าเฉลี่ยแบบถ่วงน้ำหนัก (Weight Aggregation) อีกทั้ง บทนี้จะมีการอธิบายวิธีการเปรียบเทียบประสิทธิภาพของแต่ละโมเดลในซีสิสการ์ด (cSysGuard) และการปรับเลือกค่าพารามิเตอร์ต่าง ๆ ของโมเดลที่ถูกเลือกเพื่อเพิ่มประสิทธิภาพการทำนายให้ดียิ่งขึ้น

4.1 ชั้นระดับ Level-0 โมเดล

ชั้นที่ Level-0 จะประกอบด้วยโมเดลพื้นฐานที่ออกแบบมาสำหรับการทำนายชุดข้อมูลที่เป็นอนุกรมเวลาซึ่งได้มาจากแอปพลิเคชั่นโดยตรง โมเดลเหล่านี้จะใช้ชุดข้อมูลล่าสุดเพื่อทำนายค่าที่อาจเกิดขึ้นในอนาคต โดยข้อมูลที่จะทำนายจะประกอบไปด้วย CPU หน่วยความจำ เน็ตเวิร์คแบนด์วิดท์ขาเข้าและขาออก จำนวนการทำธุรกรรมต่อวินาที และเวลาตอบสนองของคำขอโดยเฉลี่ย การพิจารณามีสององค์ประกอบหลัก: จำนวนข้อมูลที่จะนำมาพิจารณา (Observation Steps) และปริมาณข้อมูลที่จะทำนาย (Prediction Steps) รูปที่ 4.1 แสดงถึงลำดับของการทำนายของโมเดล โดยจำนวนข้อมูลที่จะนำมาพิจารณาเป็น o และปริมาณข้อมูลที่จะทำนายเป็น p ในช่วงเวลาการทำนายใหม่ในทุก ๆ k ครั้ง ตลอดการทำนาย n โดยระบบซีสิสการ์ด (cSysGuard) อนุญาตให้ผู้ใช้สามารถปรับพารามิเตอร์เหล่านี้ได้เพื่อตอบสนองความต้องการให้เข้ากับสภาพแวดล้อมนั้น ๆ ในการศึกษานี้งานวิจัยได้กำหนดจำนวนข้อมูลที่จะนำมาพิจารณาเท่ากับ 30 และปริมาณข้อมูลที่จะทำนายเท่ากับ 5 และกำหนดรอบเวลาการทำนายใหม่ให้เท่ากับปริมาณข้อมูลที่จะทำนายที่เราได้กำหนดไว้ข้างต้น



รูปที่ 4.1 ลำดับการทำนายการถดถอยในซีสิสการ์ด (cSysGuard)

งานวิจัยนี้ได้เลือกใช้โมเดลที่หลากหลายในการทำนายในชั้น Level-0 ซึ่งประกอบไปด้วย 8 โมเดล: ARIMA, SARIMA, ETS, CNN, TCN, RNN, GRU และ LSTM โดยแต่ละโมเดลมีลักษณะวิธีการทำนายที่เป็นเอกลักษณ์ ทำให้เราสามารถเห็นแง่มุมและรูปแบบต่าง ๆ กับผลลัพธ์ที่ได้ออกมา ในการใช้โมเดลการเรียนรู้เชิงลึกหรือ Deep Learning งานวิจัยนี้ตั้งต้นด้วยการใช้ Hidden Layer สองชั้น โดยชั้นแรกและชั้นที่สองจะมี 64 และ 32 นิวรอนตามลำดับ และใช้ Activation ฟังก์ชันเป็น ReLU ทั้งคู่ ในการตั้งค่านี้ งานวิจัยนี้ได้ใช้ Libraries ของ Python ที่เป็น Statsmodels และ TensorFlow เพื่อสร้างโมเดลและทำนายผล อย่างไรก็ตาม จากการทดลองเบื้องต้น เราพบว่าค่าพารามิเตอร์ตั้งต้นของ TensorFlow ทำให้การทำนายผลของโมเดล Deep Learning นั้นไม่แม่นยำกับชุดข้อมูลของงานวิจัยเท่าที่ควรจะเป็น ดังนั้นการปรับพารามิเตอร์ตั้งต้นจึงเป็นสิ่งจำเป็น จากที่แสดงในตารางที่ 4.1 ตารางได้ระบุที่ค่าพารามิเตอร์ต่าง ๆ ที่ถูกใช้ในการฝึกโมเดล Deep Learning โดยจะมีบางพารามิเตอร์ที่จะถูกปรับเปลี่ยนไปจากเดิม เพื่อรักษาความสามารถในการเรียนรู้เริ่มต้นของโมเดลได้ หลังจากโมเดลในชั้นที่ Level-0 ถูกฝึกและทำนายเสร็จ ตัวโมเดลจะมอบผลลัพธ์ของตนไปยังโมเดลในชั้นที่ Level-1 ถัดต่อไป

ตารางที่ 4.1 พารามิเตอร์ที่ใช้ในการฝึก Deep Learning

พารามิเตอร์	ค่าพารามิเตอร์ที่ใช้	ค่าพารามิเตอร์ก่อนปรับ
Epochs	50	1
Batch Sizes	32	none
Shuffle	true	
Validation Split	0.0	
Validation Steps	none	
Validation Batch Size	none	
Validation Frequency	1	
Initial Epoch	0	
Steps Per Epoch	none	
Class Weight	none	
Sample Weight	none	

4.2 ชั้นระดับ Level-1 โมเดล

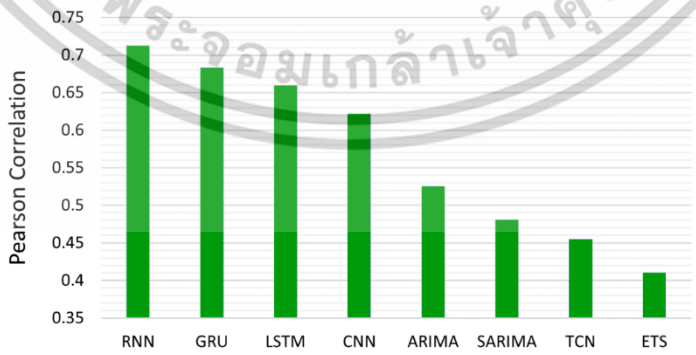
โมเดลในชั้นที่ Level-1 ทำหน้าที่เป็นเมต้าโมเดลที่ออกแบบมาเพื่อปรับปรุงผลลัพธ์จากโมเดลชั้นที่ Level-0 โดยจะนำผลลัพธ์จากชั้นที่แล้วมาเป็นข้อมูลให้กับโมเดลในชั้นนี้เพื่อใช้ในการฝึกหรือทำนาย โดยการทำงานนี้จะช่วยปรับปรุงการทำนายจากโมเดลก่อนหน้าได้ ซึ่งทำให้ซีสิสการ์ด (cSysGuard) มีประสิทธิภาพการทำนายโดยรวมที่ดียิ่งขึ้น โดยชั้นนี้จะประกอบด้วยกัน 3 โมเดล: Linear Regression, Random Forest และ Feedforward Neural Networks สำหรับ Linear Regression และ Random Forest งานวิจัยนี้ใช้พารามิเตอร์ดั่งเดิมที่ Scikit-learn Library ได้ให้มา และสำหรับ Feedforward Neural Networks งานวิจัยนี้ได้ใช้โครงสร้างโมเดลและการตั้งค่าพารามิเตอร์ให้เหมือนกับ Deep Learning โมเดลในชั้น Level-0 ที่ใช้ TensorFlow

ถึงแม้การนำผลลัพธ์ของทุกโมเดลพื้นฐานมาเป็นข้อมูลให้กับเมต้าโมเดลจะทำให้ประสิทธิภาพการทำนายที่ดีขึ้นโดยรวมดีที่สุด อย่างไรก็ตาม งานวิจัยเราต้องการสร้างสมดุลระหว่างการใช้ทรัพยากรในการคำนวณและความแม่นยำในการทำนายของซีสิสการ์ด (cSysGuard) อย่างเหมาะสม เราจึงนำเสนอวิธีในการเลือกโมเดลพื้นฐานเพื่อกำหนดว่าโมเดลใดที่จะเป็นตัวที่เอาไว้ป้อนข้อมูลให้ไปยังกับเมต้าโมเดลนั้น ๆ การเลือกนี้ขึ้นอยู่กับ

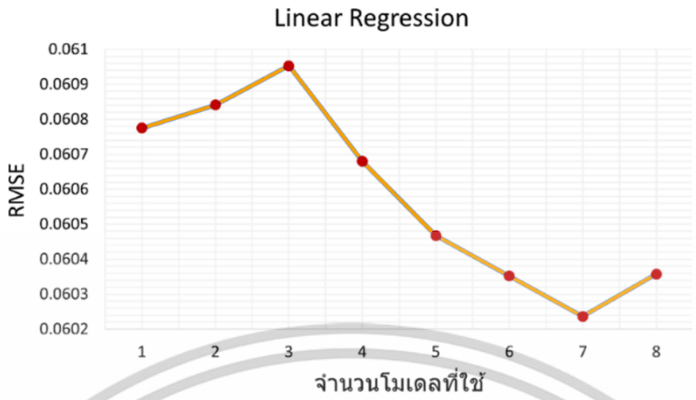
ประสิทธิภาพการทำนายของแต่ละโมเดลที่มีผลกับตัวชี้วัดนั้น ๆ ซึ่งประเมินโดยสัมประสิทธิ์ของ Pearson Correlation (r) ซึ่งระบุตามสมการ (4.1) ดังนี้

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \quad (4.1)$$

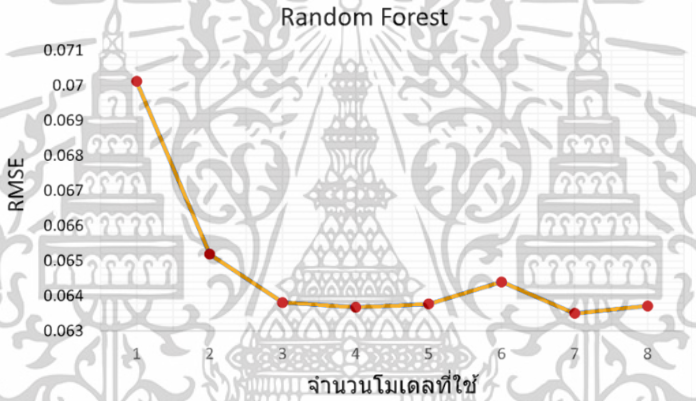
โดย x_i แทนค่าทำนายตัวชี้วัดของโมเดล Level-0, $\bar{x}$ หมายถึงค่าเฉลี่ยทำนายตัวชี้วัดของโมเดลพื้นฐานทั้งหมด, y_i แสดงค่าตัวชี้วัดจริง, และ $\bar{y}$ สะท้อนถึงค่าเฉลี่ยของตัวชี้วัดจริง หลังจากคำนวณคะแนนแล้ว เราได้จัดเรียงโมเดลพื้นฐานตามลำดับจากมากไปหาน้อยโดยอิงจากค่าที่ได้ออกมา เพื่อประเมินความสอดคล้องระหว่างข้อมูลจริงและผลลัพธ์การทำนายของโมเดลนั้น ๆ จากตัวอย่างในรูปที่ 4.2 ได้แสดงความสัมพันธ์ของ Pearson ของแต่ละโมเดลพื้นฐานในการทำนายคำขอ CPU โดยเราจะนำลำดับที่ได้มาใช้ในขั้นตอนต่อไป เราเริ่มด้วยการนำผลลัพธ์ของโมเดลพื้นฐานตัวแรกที่มีค่า Pearson สูงสุดก่อนมาเป็นข้อมูลให้กับเมต้าโมเดล โดยกระบวนการนี้จะถูกดำเนินซ้ำไปจนกว่าโมเดลพื้นฐานทั้งหมดจะถูกใช้และประเมิน เพื่อประเมินประสิทธิภาพการทำนายของแต่ละเมต้าโมเดลที่เพิ่มขึ้นโดยวัดจาก RMSE ตัวอย่างเช่น รูปที่ 4.3 - 4.5 แสดงถึงการปรับปรุงประสิทธิภาพของเมต้าโมเดลสำหรับการทำนายคำขอ CPU ด้วยการเพิ่มจำนวนของโมเดลพื้นฐานให้กับโมเดลนั้น ๆ การวิเคราะห์ในงานวิจัยนี้มุ่งหวังที่จะระบุจุดศอก (Elbow Point) ซึ่งเป็นจุดที่การเพิ่มขึ้นของประสิทธิภาพการทำนายจะเริ่มมีเพียงเล็กน้อยหลังจากนั้นเท่านั้น การคัดสรรนี้ช่วยทำให้มีการใช้ทรัพยากรในการคำนวณที่เหมาะสม ในขณะเดียวกันก็ยังให้ผลการทำนายที่แม่นยำอยู่ กระบวนการนี้ทำให้ผู้ใช้ซีสิสการ์ด (cSysGuard) นั้นมีความหยืดหยุ่นและมีทางเลือกในการเลือกโมเดลที่จะนำมาใช้ เพื่อบริหารจัดการสมดุลระหว่างการใช้ทรัพยากรในการคำนวณและความแม่นยำการทำนายให้ออกมาเหมาะสมที่สุดกับชุดข้อมูลหรือสภาพแวดนั้น ๆ



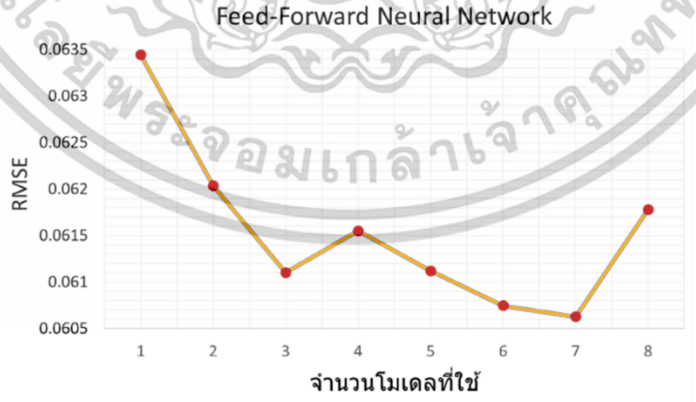
รูปที่ 4.2 การวิเคราะห์ความสัมพันธ์ของโมเดลพื้นฐานในการทำนายคำขอ CPU



รูปที่ 4.3 RMSE ของ Linear Regression เทียบกับจำนวนโมเดลในชั้น Level-0 ที่เพิ่มขึ้น



รูปที่ 4.4 RMSE ของ Random Forest เทียบกับจำนวนโมเดลในชั้น Level-0 ที่เพิ่มขึ้น



รูปที่ 4.5 RMSE ของ Feed-Forward Neural Network เทียบกับจำนวนโมเดลในชั้น Level-0 ที่เพิ่มขึ้น

4.3 ชั้นรวมผลลัพธ์การทำนาย (Weight Aggregation)

ชั้นนี้จะเป็นขั้นตอนสุดท้ายในการทำนายตัวชี้วัดแบบถดถอย โดยใช้ประโยชน์จากการมีส่วนร่วมของแต่ละเมต้าโมเดลให้มีประสิทธิภาพมากที่สุด ชั้นนี้จะรับผลลัพธ์การทำนายของโมเดลจากชั้น Level-1 และสรุปผลทำนายโดยใช้วิธีการหาค่าเฉลี่ยตามน้ำหนัก โดยเริ่มจากกำหนดน้ำหนักที่สูงกว่าให้กับโมเดลที่มีความแม่นยำที่มากกว่า ในการคำนวณน้ำหนัก ระบบได้ใช้ Performance Vector (PV) ที่เปรียบเป็นประสิทธิภาพของโมเดลในชั้น Level-1 ตามรอบปรับปรุงก่อนหน้า (t ครั้ง) โดย PV แสดงเป็นเมทริกซ์ $3 \times t$ ตามสมการ (4.2) ดังนี้

$$PV = \begin{bmatrix} PE_{LR,1} & PE_{LR,2} & \cdots & PE_{LR,t} \\ PE_{RF,1} & PE_{RF,2} & \cdots & PE_{RF,t} \\ PE_{FF,1} & PE_{FF,2} & \cdots & PE_{FF,t} \end{bmatrix} \quad (4.2)$$

โดย $PE_{x,t}$ คือ RMSE ที่ถูกคำนวณในการทำนายของเมต้าโมเดล x จากช่วงเวลาที่ 1 ไปยัง t ในกรณีนี้ LR คือ Linear Regression, RF คือ Random Forest และ FF คือ Feedforward Neural Networks กระบวนการนี้ช่วยทำให้เรารู้ถึงประสิทธิภาพของเมต้าโมเดลแต่ละตัวในสภาพแวดล้อมที่มีลักษณะเปลี่ยนแปลงอยู่ตลอดเวลา

อย่างไรก็ตาม การนำ RMSE มาเป็นน้ำหนักของเมต้าโมเดลโดยตรงอาจไม่มีประสิทธิภาพอย่างเท่าที่ควร เนื่องจากอาจไม่สะท้อนถึงสหสัมพันธ์ของโมเดลได้อย่างเพียงพอ โดยเฉพาะในกลุ่มที่แต่ละโมเดลมี RMSE ที่มีใกล้เคียงกัน เพื่อที่จะแก้ปัญหานี้ งานวิจัยได้พัฒนาระบบการถ่วงน้ำหนักให้ปรับอยู่ในรูปแบบ Normalization โดยการใช้ฟังก์ชัน Softmax เพื่อปรับค่า RMSE ที่มาจาก PV ให้ในระหว่าง 0 ถึง 1 ($NormWeight_{i.j}$) โดยสมการถูกกำหนดใน (4.3) ดังนี้

$$NormWeight_{i.j} = \frac{exp(-\alpha \times PE_{i,j})}{\sum_{k \in \{LR,RF,FF\}} exp(-\alpha \times PE_{k,j})} \quad (4.3)$$

โดยน้ำหนักของแต่ละโมเดลจะถูกคำนวณด้วยฟังก์ชัน Exponential พร้อมกับค่าผกผันของ RMSE โมเดลนั้น ๆ กำหนดโดย $exp(-\alpha \times PE_{i,j})$ การคำนวณครั้งนี้ทำให้โมเดลที่มีข้อผิดพลาดเพียงเล็กน้อยนั้นมีค่าความต่างที่ชัดเจนมากขึ้น โดย α ทำหน้าที่เป็นตัวควบคุม Sensitivity เพื่อปรับความละเอียดอ่อนความแตกต่างของ RMSE ในแต่ละโมเดล และยังเป็นตัวป้องกันเพื่อไม่ให้เกิดปัญหา Integer Overflow ในการคำนวณของระบบด้วยฟังก์ชัน Exponential โดยการศึกษานี้มุ่งหวังที่จะทำให้ค่าน้ำหนักของแต่ละโมเดลนั้นมีความแตกต่างกันมากที่สุด ด้วยการ

ปรับค่า α ให้ผลลัพธ์ของฟังก์ชัน Exponential ออกมามีค่ามากที่สุด ขณะที่ผลลัพธ์เหล่านั้นจะต้องอยู่ในช่วงที่ทำให้ไม่เกิดปัญหาของ Integer Overflow ที่กล่าวมาข้างต้น กลยุทธ์นี้ช่วยให้สามารถสร้างความแตกต่างในน้ำหนักของโมเดลได้อย่างชัดเจนมากขึ้น ซึ่งเป็นประโยชน์อย่างมากโดยเฉพาะเมื่อโมเดลในกลุ่มทดลองนั้นมีข้อผิดพลาดที่ใกล้เคียงกัน

เพื่อสรุปผลการทำนายสุดท้าย ระบบได้รวมผลลัพธ์จากโมเดลในชั้นที่ Level-1 และหาผลทำนายสุดท้าย P_m ณ จุดเวลา m ด้วยการคูณของค่าทำนายและค่าเฉลี่ยตามน้ำหนักของแต่ละโมเดล ซึ่งถูกกำหนดตามสมการ (4.4) ดังนี้

$$P_m = \sum_{k \in \{LR,RF,FF\}} NormWeight_{i.j} \times P_{k,m} \tag{4.4}$$

จากสมการที่ได้ ทำนายของแต่ละเมต้าโมเดล ซึ่งแทนด้วย $P_{k,m}$ จะคูณด้วยน้ำหนักของตนเองที่ถูกปรับปรุงแล้ว หรือนั้นก็คือ $NormWeight_{k.m}$ กระบวนการนี้ทำให้แต่ละโมเดลนั้นมีส่วนร่วมของต่อการทำนายสุดท้ายตามสัดส่วนกับความน่าเชื่อถือของโมเดลนั้น ๆ หรืออีกนัยหนึ่ง วิธีนี้ทำให้โมเดลที่มีประสิทธิภาพสูงในการทำนายนั้นมีอิทธิพลมากกว่ากับผลลัพธ์เมื่อเทียบกับโมเดลที่มีประสิทธิภาพการทำนายที่ต่ำกว่า ซึ่งนำไปสู่การทำนายรูปแบบผสมโดยรวมให้มีความน่าเชื่อถือและแม่นยำได้ยิ่งขึ้น และให้ผลที่ดีกว่าการหาค่าเฉลี่ยแบบธรรมดาอีกด้วยซึ่งถูกยืนยันในบทที่ 6.3

4.4 การเปรียบเทียบโมเดล

งานวิจัยมีวิธีประเมินประสิทธิภาพของซีสิสการ์ด (cSysGuard) ในการทำนายตัวชี้วัดแบบถดถอย โดยการเปรียบเทียบด้วย RMSE กับโมเดลในชั้น Level-0 เพื่อมาเป็นเกณฑ์มาตรฐานของประสิทธิภาพในตัวชี้วัดต่าง ๆ วิธีนี้อธิบายความแม่นยำและความน่าเชื่อถือของงานวิจัยนี้อย่างชัดเจน โดยเน้นความสามารถในการจัดการสภาพแวดล้อมที่เปลี่ยนแปลงตลอดเวลาอย่างมีประสิทธิภาพ

4.5 การปรับแต่งไฮเปอร์พารามิเตอร์

หลังจากที่เปรียบเทียบโมเดลแล้ว งานวิจัยได้มุ่งเน้นไปที่การเพิ่มประสิทธิภาพของระบบต่อไป เพื่อบรรลุวัตถุประสงค์นี้ งานวิจัยได้เลือกแต่ละโมเดลจากชั้น Level-0 และ Level-1 เพื่อปรับโมเดลพารามิเตอร์ด้วยวิธี Bayesian Optimization และสำรวจผลการปรับปรุงของระบบด้วย RMSE เพื่อมาเป็นเกณฑ์มาตรฐานของงานวิจัย

เนื่องจากข้อจำกัดของการกำหนดค่าของระบบที่ระบุไว้ในบทที่ 6.4 และเพื่อเน้นหาวิธีการเพิ่มประสิทธิภาพนำทายให้มีประสิทธิภาพที่เพิ่มขึ้นแบบโดยรวม การศึกษานี้จึงจำกัดการปรับแต่งพารามิเตอร์ไฮเปอร์เพียงตัวชี้วัดและโมเดลแค่บางตัว โดยงานวิจัยได้มุ่งเน้นการปรับแต่ง Recurrent Neural Network (RNN) และ Random Forest ซึ่งเป็นโมเดลจาก Level-0 และ Level-1 ตามลำดับ โดยเน้นการทำนายคำขอ CPU ในตาราง 4.2 และ 4.3 ได้ระบุขอบเขตของพารามิเตอร์ของแต่ละโมเดลที่งานวิจัยได้กำหนดไว้ โดยแต่ละพารามิเตอร์นั้นจะผลต่อประสิทธิภาพการทำนายของโมเดล พารามิเตอร์ของ RNN จะประกอบไปด้วยองค์ประกอบในการจัดโครงสร้างข้อมูล (Observation Steps) โครงสร้างโมเดล (Learning Rate, Number of Neurons, and Activation Functions) และค่าการฝึกโมเดล (Epochs and Batch Size) และสำหรับพารามิเตอร์ของ Random Forest ที่พิจารณาได้แก่ จำนวนต้นไม้ (Estimators), ความลึกของต้นไม้ (Max Depth), ตัวอย่างขั้นต่ำสำหรับการแบ่ง (Min Sample Split), ตัวอย่างขั้นต่ำต่อใบ (Min Sample Leaf), จำนวน Feature สูงสุดที่พิจารณาสำหรับการแบ่ง Node (Max Features), จำนวนสูงสุดใบใน Node (Max-leaf Nodes), ค่าการลดลงขั้นต่ำของความไม่บริสุทธิ์ (Min Impurity Decrease), การใช้ตัวอย่าง Bootstrap และฟังก์ชันที่ใช้วัดคุณภาพของการแบ่ง (Criterion) ผลลัพธ์หลังจากการปรับพารามิเตอร์จะสามารถให้มุมมองว่าการเพิ่มประสิทธิภาพของโมเดลนั้นส่งผลต่อชั้นถัดไปอย่างไร รวมไปถึงความแม่นยำโดยรวมในการทำนายของซีสิสการ์ด (cSysGuard)

ตารางที่ 4.2 ช่วงของพารามิเตอร์ในการปรับแต่ง Recurrent Neural Networks ในการทำนายการใช้ CPU

พารามิเตอร์	ช่วงของค่าพารามิเตอร์ที่ทำการทดลอง
Observation Steps	5 - 10
Learning Rate	0.001 – 0.1
Number of Neurons in Layer 1	2 - 80
Number of Neurons in Layer 2	2 - 80
Activation Functions in Layer 1	ReLu, Tanh, Sigmoid
Activation Functions in Layer 2	ReLu, Tanh, Sigmoid
Epochs	10 - 150
Batch Size	16 - 64

ตารางที่ 4.3 ช่วงของพารามิเตอร์ในการปรับแต่ง Random Forest ในการทำนายการใช้ CPU

พารามิเตอร์	ช่วงของค่าพารามิเตอร์ที่ทำการทดลอง
Number of Estimators	10 - 1000
Max Depth	3 - 30
Min Sample Split	2 - 20
Min Sample Leaf	1 - 20
Max Features	Auto, Sqrt, Log2
Max Leaf Nodes	10 - 1000
Min Impurity Decrease	0 - 0.1
Bootstrap	True, False
Criterion	Squared Error, Absolute Error, Poisson, Friedman MSE

บทที่ 5

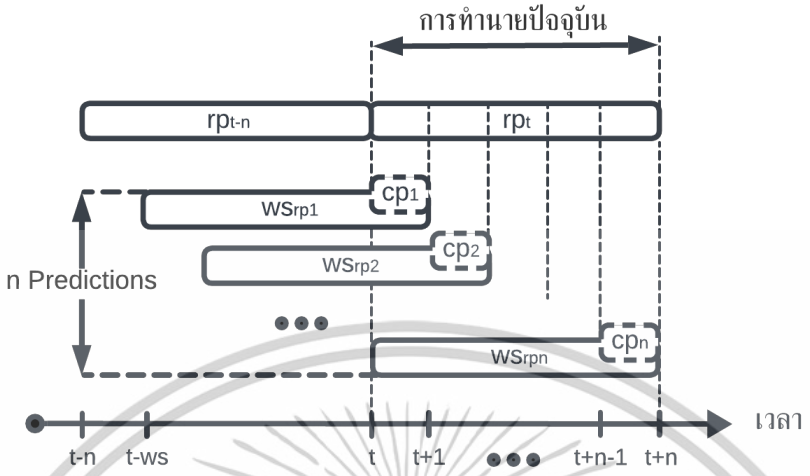
กระบวนการทำนายสถานะของเซิร์ฟเวอร์

บทนี้จะอธิบายกระบวนการการทำนายของโมเดลที่มีรูปแบบจำแนกประเภทเพื่อทำนายสถานะของระบบเซิร์ฟเวอร์อย่างละเอียด ซึ่งครอบคลุมไปถึงการเตรียมข้อมูลและใช้โมเดลในงานวิจัยนี้ รวมถึงการเปรียบเทียบประสิทธิภาพของแต่ละโมเดล และการปรับแต่งพารามิเตอร์ของโมเดลที่ถูกเลือกเพื่อเพิ่มประสิทธิภาพการทำนายในส่วนสุดท้ายของซีสิสการ์ด (cSysGuard)

5.1 ชั้นการประเมินสถานะเซิร์ฟเวอร์

ชั้นนี้จะเป็นชั้นสุดท้ายในทำนายของซีสิสการ์ด (cSysGuard) ซึ่งมาพร้อมกับโมเดลการจำแนกประเภทที่ถูกออกแบบมาเพื่อประเมินสถานะของระบบ โดยโมเดลนี้จะจำแนกสถานะของระบบเป็น Healthy หรือ Unhealthy โดยจะอิงตามผลลัทธ์ที่ได้มาจากชั้นการทำนายตัวชี้วัดประสิทธิภาพที่อยู่ก่อนหน้า หรืออีกนัยหนึ่ง ชั้นนี้จะนำผลลัพธ์จากชั้นที่เป็นการทำนายแบบถดถอยมาเป็นข้อมูลที่จะระบุว่า ณ เวลานั้นของการทำนายนั้น ตัวแอปพลิเคชั่นมีสถานะเป็นเช่นไร ในส่วนนี้จะทำให้ผู้ดูแลแอปพลิเคชั่นทราบถึงความน่าจะเป็นที่ระบบอาจล้มเหลวในอนาคตได้

ในการเตรียมข้อมูลเพื่อป้อนให้กับโมเดล งานวิจัยได้ใช้เทคนิค Sliding Window เพื่อรวบรวมแนวโน้มของข้อมูลเพื่อที่เพิ่มความแม่นยำในการทำนายสถานะของเซิร์ฟเวอร์ ณ เวลานั้น ระบบจะนำข้อมูลภายในช่วงที่กำหนดให้โมเดลได้พิจารณา ซึ่งช่วงที่กำหนดจะถูกเรียกว่า Windows Size รูปที่ 5.1 แสดงถึงการทำนายสถานะของเซิร์ฟเวอร์ที่นำข้อมูลจากชั้นที่ทำนายตัวชี้วัดประสิทธิภาพมาก่อนหน้านี้ โดย rp (หรือ Regression Prediction) คือการทำนายของการถดถอยที่จะเป็นผลลัพธ์ของตัวชี้วัดประสิทธิภาพ ณ เวลา t และ cp (หรือ Classification Prediction) คือการทำนายแบบจำแนกประเภทที่จะเป็นผลลัพธ์ของสถานะเซิร์ฟเวอร์ โดยมี ws (Windows Size) เป็นจำนวนชุดข้อมูลที่ใช้ในการวิเคราะห์สถานะเซิร์ฟเวอร์ ณ เวลานั้น ซึ่งการทำนายของการจำแนกประเภทจะมีการเรียก n ครั้งต่อการทำนายการถดถอยครั้งหนึ่ง หรืออีกนัยหนึ่ง n ก็คือจำนวนชุดข้อมูลที่โมเดลการถดถอยทำนายต่อครั้ง หรือคือ Prediction Steps นั้นเอง



รูปที่ 5.1 ลำดับการทำนายการจำแนกของซีสิสการ์ด (cSysGuard)

นอกจากนี้ ชุดข้อมูลของงานวิจัยนี้มีลักษณะที่ไม่สมดุลโดยมีสถานะ Healthy ที่จะถูกพบบ่อยกว่า Unhealthy เป็นอย่างมาก ซึ่งอาจมีความเสี่ยงที่โมเดลจะเอนเอียงไปทางกลุ่มข้างมากและทำงานได้ไม่ดีในกลุ่มที่มีปริมาณน้อย ซึ่งเป็นกลุ่มที่สำคัญแต่หาได้ยากกว่า เพื่อช่วยบรรเทาปัญหานี้ งานวิจัยได้ใช้เทคนิค SMOTE เพื่อสร้างข้อมูลสังเคราะห์ให้กับข้อมูลกลุ่มน้อย ซึ่งทำให้โมเดลทำงานได้ดีขึ้นด้วยการมีตัวอย่างเทียมของกลุ่มนี้เพื่อปรับปรุงในการตรวจจับข้อมูล

5.2 การเปรียบเทียบโมเดล

งานวิจัยนี้ได้คัดสรรและเลือกมา 5 โมเดลเพื่อมาทำนายสถานะของเซิร์ฟเวอร์ นั้นก็คือ Decision Trees, Support Vector Machine, K-Nearest Neighbors, Logistic Regression และ Neural Networks โดยโมเดลเหล่านี้เริ่มต้นด้วยพารามิเตอร์เริ่มต้นที่ Scikit-learn และ TensorFlow ให้มาเบื้องต้น และโครงสร้างของ Neural Networks นั้นมีสองชั้น: ชั้นแรกมี 6 นิวรอนโดยมี Activation ฟังก์ชันเป็น ReLU และชั้นที่สองประกอบด้วยนิวรอนเดียวโดยมี Sigmoid เป็น Activation ฟังก์ชัน

ในกระบวนการประเมินผล งานวิจัยได้ใช้เทคนิค Cross Validation เพื่อดำเนินการฝึกและทดสอบหลายรอบบนการแบ่งข้อมูลที่แตกต่างกันเพื่อหาคะแนนเฉลี่ย โดยเกณฑ์การวัดจะเป็น Precision, Recall และ F1 Score ด้วยการหาค่าแบบ Macro Average ซึ่งเป็นการให้น้ำหนักแต่ละหมวดหมู่เท่าเทียมกันในการประเมินผล

ตลอดกระบวนการตรวจสอบ ในงานวิจัยนี้ โมเดลที่มี F1 Score สูงที่สุดจะเป็นโมเดลที่ถูกเลือก แต่ละโมเดลจะถูกกำหนดตามสมการ (5.1), (5.2) และ (5.3) ดังนี้

$$Precision_{macro} = \frac{Precision_{healthy} + Precision_{unhealthy}}{2} \quad (5.1)$$

$$Recall_{macro} = \frac{Recall_{healthy} + Recall_{unhealthy}}{2} \quad (5.2)$$

$$F1\text{-}Score_{macro} = 2 \times \frac{Precision_{macro} \times Recall_{macro}}{Precision_{macro} + Recall_{macro}} \quad (5.3)$$

5.3 การปรับแต่งไฮเปอร์พารามิเตอร์

โมเดลจำแนกประเภทที่ถูกเลือกจะได้การปรับแต่งพารามิเตอร์โดยผ่านวิธีการ Bayesian Optimization เพื่อเพิ่มความสามารถในการทำนายให้เข้ากับชุดข้อมูลของงานวิจัยนี้ได้อย่างมีประสิทธิภาพ งานวิจัยนี้จะทดสอบช่วงของค่าพารามิเตอร์เพื่อหาค่าที่ให้ F1 Score สูงสุด กระบวนการนี้ช่วยให้เราสามารถเพิ่มความแม่นยำของโมเดลและปรับโมเดลให้เฉพาะเจาะจงกับชุดข้อมูลได้อย่างมีประสิทธิภาพสูงสุด

ตารางที่ 5.1 แสดงถึงพารามิเตอร์ของ Decision Trees (เป็นโมเดลที่ให้ผลดีที่สุด) ที่งานวิจัยได้ศึกษา โดยจะประกอบไปด้วยความลึกของต้น (Max Depth), ตัวอย่างขั้นต่ำในการแบ่งและต่อใบ (Min Sample Split & Min Sample Leaf), ฟังก์ชันที่ใช้วัดคุณภาพการแบ่ง (Criterion), จำนวน Feature สูงสุด (Max Features), จำนวนใบสูงสุด (Max-leaf Nodes) และค่าลดลงขั้นต่ำของความไม่บริสุทธิ์ (Min Impurity Decrease)

ตารางที่ 5.1 ช่วงของพารามิเตอร์ไฮเปอร์สำหรับการปรับแต่ง Decision Trees ในการทำนายสถานะเซิร์ฟเวอร์

พารามิเตอร์	ช่วงของค่าพารามิเตอร์ที่ทำการทดลอง
Max Depth	3 - 500
Min Samples Split	2 - 100
Min Samples Leaf	1 - 100
Criterion	Gini, Entropy
Max Features	None, Auto, Sqrt, Log2
Max Leaf Nodes	10 - 1000
Min Impurity Decrease	0 - 5

บทที่ 6

ผลการวิจัย

บทนี้จะนำเสนอผลการค้นคว้าของงานวิจัย โดยครอบคลุมไปถึงการประเมินผลของการทำนายตัวชี้วัดและสถานะของระบบเซิร์ฟเวอร์ที่ได้มาจากโมเดลต่าง ๆ รวมถึงการทำนายจากซีสิสการ์ด (cSysGuard) อีกด้วย นอกจากนี้ งานวิจัยก็ได้นำเสนอข้อมูลเชิงลึกเกี่ยวกับผลลัพธ์ที่พบเจอมาเพื่อเพิ่มมุมมองในการวิเคราะห์ผลประเมินอีกด้วย

6.1 การทำนายตัวชี้วัดประสิทธิภาพของเซิร์ฟเวอร์

ส่วนย่อยนี้จะอธิบายรายละเอียดผลการทดลองในการทำนายตัวชี้วัดประสิทธิภาพของระบบเซิร์ฟเวอร์ ซึ่งมีอยู่สองหมวดหมู่หลัก นั้นก็คือผลลัพธ์หลังจากการเปรียบเทียบโมเดลและการปรับแต่งพารามิเตอร์ไฮเปอร์

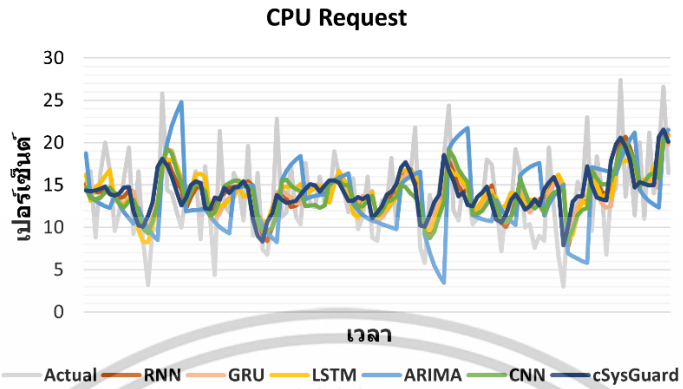
6.1.1 การเปรียบเทียบโมเดล

ตารางที่ 6.1 แสดงถึงประสิทธิภาพในการทำนายของซีสิสการ์ด (cSysGuard) เมื่อเปรียบเทียบกับโมเดลพื้นฐานทั่วไป ที่มีความสามารถที่เหนือกว่าเป็นส่วนใหญ่อย่างชัดเจน โดยเฉพาะอย่างยิ่งในการทำนายเวลาตอบสนองโดยเฉลี่ย ที่ซีสิสการ์ด (cSysGuard) ได้บรรลุความต่างของประสิทธิภาพถึง 2.28 เท่าเมื่อเปรียบเทียบกับ ETS การเปรียบเทียบนี้ยังแสดงถึงความสามารถที่ยังเหนือกว่า Deep Learning โมเดลต่าง ๆ เช่น GRU และ LSTM ซึ่งถูกรู้จักกันในเรื่องประสิทธิภาพการทำนายที่ดี ในการเปรียบเทียบนี้ได้ยืนยันถึงความสามารถในการปรับตัวในสถานการณ์การทำนายตัวชี้วัดระบบในสภาพแวดล้อมคลาวด์ที่หลากหลายได้เป็นอย่างดี

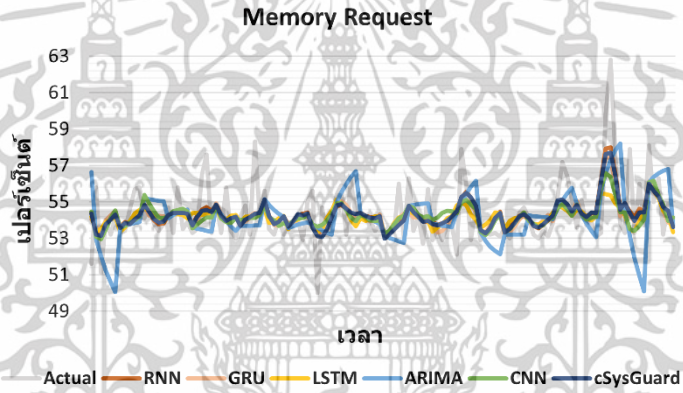
มากไปกว่านี้ รูปที่ 6.1-6.6 แสดงถึงผลการทำนายในรูปแบบกราฟของซีสิสการ์ด (cSysGuard) และโมเดลพื้นฐานที่ถูกนำมาทำนายกับตัวชี้วัดต่าง ๆ อยู่บ่อยครั้ง โดยแต่ละโมเดลจะประกอบไปด้วย GRU, LSTM, RNN, ARIMA และ CNN จากกราฟที่แสดงให้เห็น เราพบว่าซีสิสการ์ด (cSysGuard) นั้นให้ผลการทำนายที่มีแนวโน้มที่ใกล้เคียงกับค่าจริงต่าง ๆ ในแต่ละช่วงเวลาเป็นอย่างดี และแสดงให้เห็นว่ามีความสามารถในทำนายที่ยอดเยี่ยมกับข้อมูลที่มีรูปแบบหลากหลายได้อีกด้วย

ตารางที่ 6.1 การเปรียบเทียบประสิทธิภาพ RMSE ระหว่างซีสิสการ์ด (cSG) กับโมเดลพื้นฐานที่ถูกเลือก

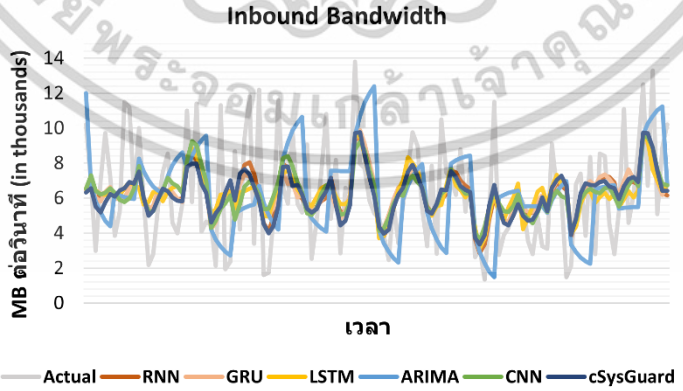
	cSG	GRU	LSTM	RNN	CNN	ARIMA	SARIMA	TCN	ETS
คำขอ CPU (normalized)	**0.0479**	0.0492	0.0501	0.0480	0.0525	0.0682	0.0772	0.058	
คำขอหน่วยความจำ (normalized)	**0.0149**	0.0152	0.0155	0.0149	0.0159	0.0228			
เน็ตเวิร์คแบนด์วิดท์ขาเข้า (หน่วย: MB/s)	**2778**	2795	2827	2787	2897	3658		3070	
เน็ตเวิร์คแบนด์วิดท์ขาออก (หน่วย: MB/s)	**2368**	2388	2416	2368	2471	3122		2632	
การทำธุรกรรมต่อวินาที	**0.6348**	0.6405	0.6459	0.6364	0.6737	0.8338	0.8313	0.707	
เวลาตอบสนองของคำขอโดยเฉลี่ย (หน่วย: ms)	**640**	649	666	642	742	1049		848	1460



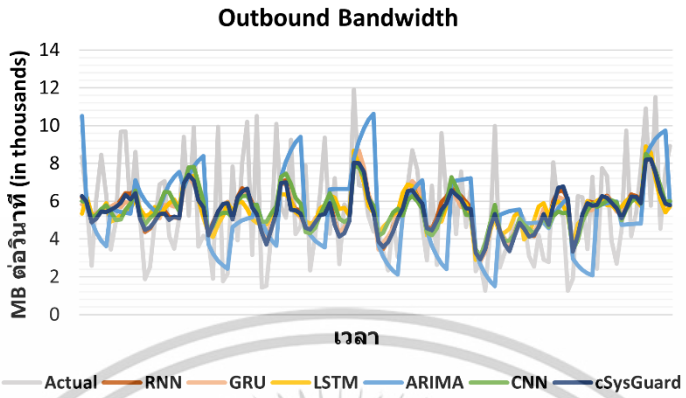
รูปที่ 6.1 ค่าจริงและผลการทำนายการใช้งานของ CPU ของ cSysGuard เมื่อเทียบกับโมเดลอื่น ๆ



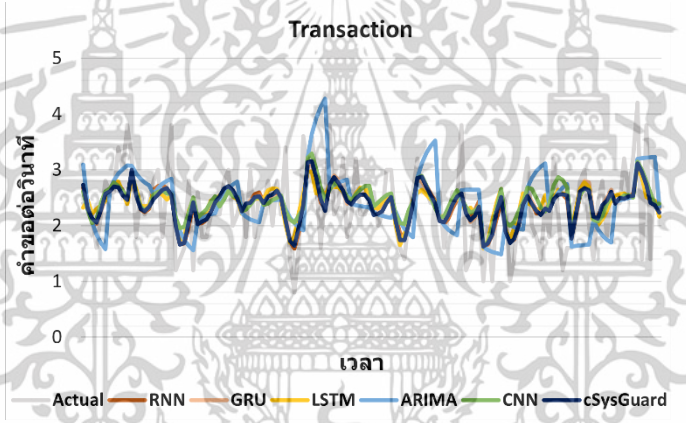
รูปที่ 6.2 ค่าจริงและผลการทำนายการใช้งานของหน่วยความจำของ cSysGuard และโมเดลอื่น ๆ



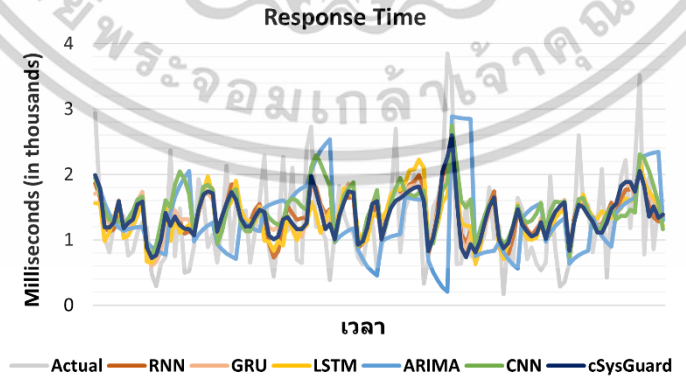
รูปที่ 6.3 ค่าจริงและผลการทำนายแบนด์วิดท์ขาเข้าของ cSysGuard และโมเดลอื่น ๆ



รูปที่ 6.4 ค่าจริงและผลการทำนายแบนด์วิดท์ขาออกของ cSysGuard และโมเดลอื่น ๆ



รูปที่ 6.5 ค่าจริงและผลการทำนายทำธุรกรรมต่อวินาทีของ cSysGuard และโมเดลอื่น ๆ



รูปที่ 6.6 ค่าจริงและผลการทำนายเวลาตอบสนองเฉลี่ยของ cSysGuard และโมเดลอื่น ๆ

6.2.1 การปรับแต่งไฮเปอร์พารามิเตอร์

ตารางที่ 6.2 และ 6.3 แสดงถึงพารามิเตอร์ที่เหมาะสมที่สุดสำหรับ RNN และ Random Forest ในการทำนายการใช้ CPU ในขณะเดียวกัน ตารางที่ 6.4 แสดงถึงประสิทธิภาพของแต่ละโมเดลที่ได้รับการปรับปรุงมา รวมถึงประสิทธิที่เพิ่มขึ้นของซีสิสการ์ด (cSysGuard) อีกด้วย ผลลัพธ์นี้บ่งชี้ถึงผลกระทบแบบกลุ่ม การปรับปรุงส่วนประกอบหนึ่งของระบบส่งผลให้มีประสิทธิภาพโดยรวมที่ดีขึ้น ซึ่งทำให้เห็นถึงความเชื่อมโยงของโมเดลภายในโครงสร้างของซีสิสการ์ด (cSysGuard)

ตารางที่ 6.2 พารามิเตอร์ที่ถูกปรับแต่งใน RNN เพื่อใช้ในการทำนายการใช้ CPU

พารามิเตอร์	ช่วงของค่าพารามิเตอร์ที่ทำการทดลอง
Observation Steps	64
Learning Rate	0.001
Number of Neurons in Layer 1	49
Number of Neurons in Layer 2	55
Activation Functions in Layer 1	ReLu
Activation Functions in Layer 2	ReLu
Epochs	100
Batch Size	16

ตารางที่ 6.3 พารามิเตอร์ที่ถูกปรับแต่งใน Random Forest เพื่อใช้ในการทำนายการใช้ CPU

พารามิเตอร์	ช่วงของค่าพารามิเตอร์ที่ทำการทดลอง
Number of Estimators	545
Max Depth	27
Min Sample Split	5
Min Sample Leaf	10
Max Features	Auto
Max Leaf Nodes	13
Min Impurity Decrease	0.00035269496460264014
Bootstrap	True
Criterion	Friedman MSE

ตารางที่ 6.4 การเปรียบเทียบประสิทธิภาพ RMSE ของ Recurrent Neural Networks (RNN), Random Forest (RF) และซีสิสการ์ด (cSysGuard) ในการใช้ CPU

	Recurrent Neural Networks	Random Forest	cSysGuard
Base Performance	0.0480	0.0486	0.0479
Level-0: Tuned RNN	0.0470	**0.0474**	**0.0469**
Level-1: Tuned RF	0.0470	0.0470	**0.0468**

6.2 การทำนายสถานะของเซิร์ฟเวอร์

ส่วนย่อยนี้จะอธิบายถึงผลการทดลองของการทำนายสถานะของระบบเซิร์ฟเวอร์โดยแบ่งออกเป็นสองหมวดหลัก นั้นก็คือผลลัพธ์การเปรียบเทียบโมเดลและการปรับแต่งไฮเปอร์พารามิเตอร์

6.2.1 การเปรียบเทียบโมเดล

ตารางที่ 6.5 นำเสนอการวิเคราะห์เปรียบเทียบประสิทธิภาพของโมเดลพื้นฐานต่าง ๆ โดยเน้นที่ Precision, Recall และ F1 Score ในรูปแบบ Macro Average การวิเคราะห์เผยให้เห็นว่า Decision Trees มีประสิทธิภาพเหนือกว่าโมเดลพื้นฐานอื่น ๆ โดยการบรรลุ F1 Score สูงที่สุดในบรรดาโมเดลอื่น ๆ ประสิทธิภาพที่สมดุลนี้เน้นถึงความเหมาะสมของ Decision Trees สำหรับการทำนายสถานะของระบบที่เชื่อถือได้ ทำให้เป็นตัวเลือกที่เหมาะสมในการประเมินกรณีนี้ จากการประเมิน งานวิจัยนี้จึงนำ Decision Trees มาเพื่อวิเคราะห์เชิงลึกในการปรับแต่งไฮเปอร์พารามิเตอร์เพื่อเพิ่มประสิทธิในการทำนายสถานะของระบบซีสิสการ์ด (cSysGuard)

ตารางที่ 6.5 ประสิทธิภาพของโมเดลในการทำนายสถานะเซิร์ฟเวอร์

	Macro-avg Precision	Macro-avg Recall	Macro-avg F1 Score
Decision Trees	0.8856	0.8802	0.8812
Support Vector Machine	0.8255	0.8231	0.8223
K-Nearest Neighbors	0.8372	0.7819	0.7736
Neural Networks	0.8515	0.8159	0.8060
Logistic Regression	0.8380	0.8380	0.8368

6.2.2 การปรับแต่งไฮเปอร์พารามิเตอร์

ด้วยการระบุว่า Decision Trees เป็นโมเดลมาตรฐานที่มีประสิทธิภาพสูงสุด ตารางที่ 6.6 แสดงถึงพารามิเตอร์ที่เหมาะสมที่สุดในการผลลัพธ์ทำนายที่ดีที่สุด นอกจากนี้ ตารางที่ 6.7 นำเสนอผลเปรียบเทียบการปรับแต่งพารามิเตอร์ต่อประสิทธิภาพของโมเดล โดยผลลัพธ์บ่งชี้ว่า Decision Trees ที่ได้รับการปรับแต่งนั้นมีประสิทธิภาพที่เพิ่มขึ้น โดยแสดงให้เห็นถึงการปรับปรุงที่เพิ่มขึ้นอยู่ประมาณ 1.56% ใน Precision, 1.66% ใน Recall และ 1.67% ใน F1 Score

ตารางที่ 6.6 พารามิเตอร์ที่ถูกปรับแต่งใน Decision Trees ในการทำนายสถานะเซิร์ฟเวอร์

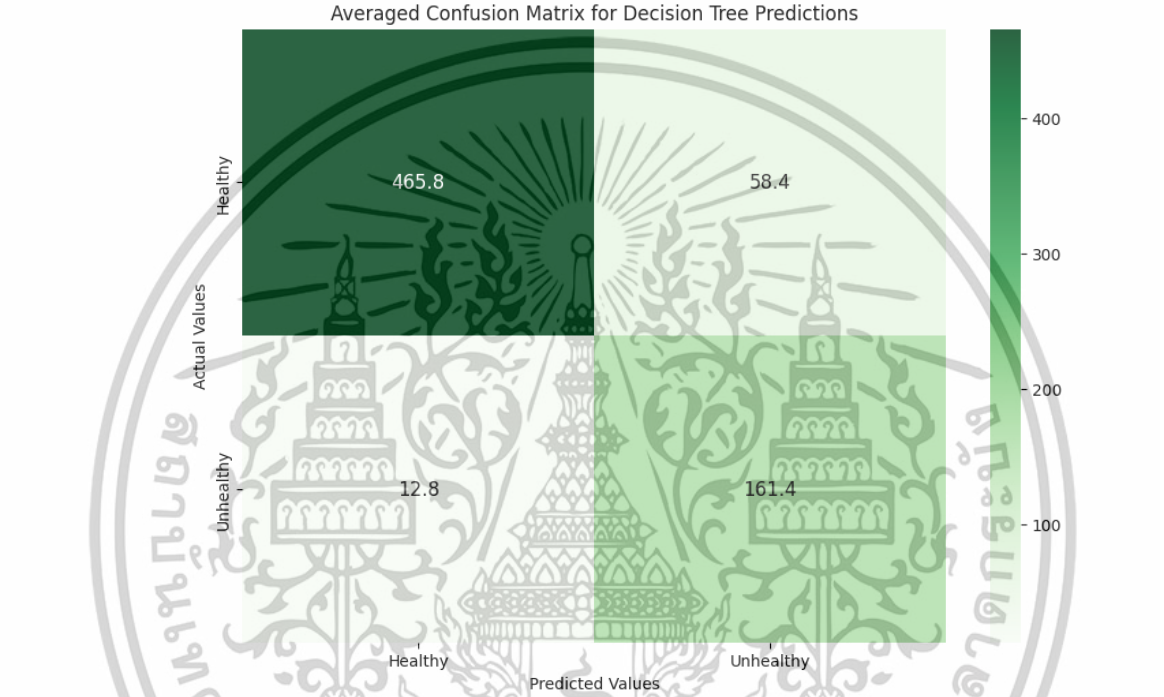
พารามิเตอร์	ช่วงของค่าพารามิเตอร์ที่ทำการทดลอง
Max Depth	129
Min Samples Split	2
Min Samples Leaf	1
Criterion	Gini
Max Features	None
Max Leaf Nodes	637
Min Impurity Decrease	0

ตารางที่ 6.7 ประสิทธิภาพของโมเดลพื้นฐานและ Decision Trees ที่ถูกปรับแต่งในการทำนายสถานะเซิร์ฟเวอร์

	Macro-avg Precision	Macro-avg Recall	Macro-avg F1 Score
Decision Trees	0.8856	0.8802	0.8812
Support Vector Machine	0.8255	0.8231	0.8223
K-Nearest Neighbors	0.8372	0.7819	0.7736
Neural Networks	0.8515	0.8159	0.8060
Logistic Regression	0.8380	0.8380	0.8368
Tuned Decision Trees	**0.9012**	**0.8968**	**0.8979**

หลังจากกระบวนการปรับแต่งพารามิเตอร์ไฮเปอร์อย่างละเอียด งานวิจัยได้เริ่มทำการวิเคราะห์ความแม่นยำในการจำแนกประเภทของโมเดลอย่างละเอียด โดยเราได้สร้าง Confusion Matrix แบบเฉลี่ย ซึ่งกำหนดในรูปที่ 6.7 ซึ่งเป็นผลลัพธ์ที่มาจากการรวบรวมผลในแต่ละรอบของการตรวจสอบแบบ Cross Validation ที่

ดำเนินการด้วยชุดข้อมูลตัวอย่างของงานวิจัย Confusion Matrix นี้ระบุสองคลาสหลักที่บ่งบอกถึงสถานะของเซิร์ฟเวอร์: Healthy และ Unhealthy โดยการแสดงข้อมูลแบบกราฟิกนี้เป็นเครื่องมือสำคัญสำหรับการประเมินความแม่นยำและความน่าเชื่อถือของโมเดลการทำนายของซีสิสการ์ด (cSysGuard) ในการแยกแยะสถานการณ์การทำงานของเซิร์ฟเวอร์ได้เป็นอย่างดี



รูปที่ 6.7 Confusion Matrix แบบเฉลี่ยที่แสดงถึงการจำแนกประเภทของซีสิสการ์ด (cSysGuard)

6.3 การวิเคราะห์เปรียบเทียบวิธีการรวมกลุ่มที่แตกต่างกัน

การศึกษานี้นำเสนอโมเดลการรวมกลุ่มแบบ Stacking ที่เป็นนวัตกรรมเพื่อเพิ่มความแม่นยำในการทำนายแบบถดถอย โดยมีโครงสร้างอยู่มีสามชั้น ซึ่งประกอบด้วยโมเดลพื้นฐานในชั้น Level-0 เมต้าโมเดลในชั้น Level-1 และใช้วิธีการหาค่าเฉลี่ยตามน้ำหนักในชั้นที่สุดท้าย งานวิจัยได้ดำเนินการทดลองเพิ่มเติม เพื่อทดสอบประสิทธิภาพกับงานวิจัยอื่นและกับรูปอื่นที่เราได้คิดค้นขึ้น ในงานวิจัยแรก เราได้นำงานของ Mehmood et al. โดยใช้เมต้าโมเดลเดี่ยวที่เป็น Decision Trees เพื่อทำนายผลสุดท้าย และในงานวิจัยที่สองซึ่งจะเป็นของ Kim et al. (CloudInsight) โดยจะหาค่าเฉลี่ยตามน้ำหนักในชั้นที่สองเพื่อทำนายผลสุดท้ายแทนที่ในการใช้เมต้าโมเดล

และอีกทั้ง เราได้ลองมาใช้วิธีการหาค่าเฉลี่ยธรรมดาในชั้นที่สามแทนการใช้การหาค่าเฉลี่ยตามน้ำหนักบนซีสิสการ์ด (cSysGuard) โดยการทดลองเหล่านี้มีเป้าหมายเพื่อกำหนดว่าโครงสร้างโมเดลที่เรียบง่ายกว่าหรือวิธีการรวมผลลัทธ์ที่ตรงไปตรงมากว่านั้นสามารถให้ผลลัพธ์ที่ดีกว่าหรือไม่ ท้ายที่สุดจากตารางที่ 6.8 ได้แสดงถึงประสิทธิภาพของวิธีการเหล่านี้โดยใช้ RMSE เป็นตัวชี้วัด

ตารางที่ 6.8 การเปรียบเทียบ RMSE ระหว่างการใช้ Meta Model ตัวเดียว, หาค่าเฉลี่ยตามน้ำหนักในชั้นที่สอง (CloudInsight), ซีสิสการ์ดที่ใช้การหาค่าเฉลี่ยแบบธรรมดา (cSG-SA) และแบบตามน้ำหนัก (cSG-WA)

	Single Meta-Model (Mehmood et al.)	CloudInsight (Kim et al.)	cSG-SA	cSG-WA
CPU Request (normalized)	0.06671	0.04913	0.04692	0.04690
Memory Request (normalized)	0.02159	0.01544	0.01495	0.01494
Inbound Bandwidth (MB/s)	4052.5	2887.43	2778.22	2777.58
Outbound Bandwidth (MB/s)	3536.6	2462.86	2369.01	2368.72
TPS	0.9117	0.63771	0.63457	0.63479
Response Time (ms)	978.21	780.786	640.410	640.204

จากผลลัพธ์ ไม่ว่าจะเป็นการใช้เมต้าโมเดลตัวเดียวหรือการหาค่าเฉลี่ยตามน้ำหนัก (CloudInsight) ในชั้นที่สอง พอเมื่อเปรียบเทียบกับซีสิสการ์ด (cSysGuard) ในรูปแบบปัจจุบัน หรือ cSG-WA นั้นไม่สามารถเทียบได้ใกล้เคียงกับประสิทธิภาพที่มีอยู่เลย ในอีกทางหนึ่ง การแทนการทำงานของชั้นที่สุดท้ายด้วยการหาค่าเฉลี่ยแบบธรรมในซีสิสการ์ด (cSysGuard) หรือ cSG-SA จะมีประสิทธิภาพที่ต่ำกว่าเล็กน้อยเมื่อเปรียบเทียบกับแบบปัจจุบัน ถึงแม้ว่าประสิทธิภาพของทั้งสองวิธีจะใกล้เคียงกัน แต่ข้อได้เปรียบหลักของการหาค่าเฉลี่ยตามน้ำหนักคือความยืดหยุ่นในการปรับแต่งการคำนวณ ซึ่งช่วยให้ผู้ใช้สามารถปรับแต่งปัจจัยการคำนวณน้ำหนักตามความต้องการเฉพาะ เช่นค่า Sensitivity ต่อข้อผิดพลาดหรือระยะช่วงการเก็บข้อมูลเพื่อนำมาปรับน้ำหนักโมเดล ความยืดหยุ่นนี้สามารถทำให้การคำนวณนั้นเหมาะกับชุดข้อมูลของผู้ใช้มากขึ้น ซึ่งย้ำถึงความสำคัญของการจัดโครงสร้างแบบชั้นสามในซีสิสการ์ด (cSysGuard) ได้เป็นอย่างดี

6.4 การประเมินเวลาการทำงานของระบบซีสิสการ์ด (cSysGuard)

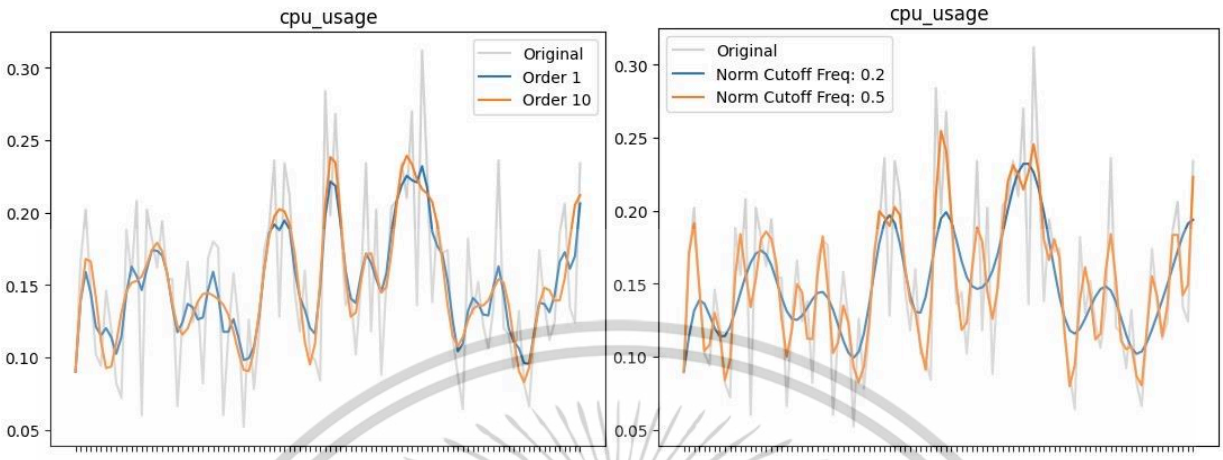
ต่อมา งานวิจัยได้ประเมินประสิทธิภาพของระบบโดยทดลองใช้ซีสิสการ์ด (cSysGuard) บนเครื่องที่มีโปรเซสเซอร์ Intel Core i7-7820HK QuadCore ความเร็ว 2.9 GHz และ RAM ประเภท DDR4 ขนาด 16GB ตาราง 6.9 นำเสนอผลลัพธ์การเปรียบเทียบ โดยแสดงถึงความแตกต่างของเวลาการทำนายและการปรับปรุงโมเดลโดยเฉลี่ยของทั้ง 3 โมเดล (RNN, GRU, และ LSTM) และซีสิสการ์ด (cSysGuard) โดยจากผลลัพธ์แสดงให้เห็นว่าซีสิสการ์ด (cSysGuard) นั้นมีเวลาการประมวลผลที่สูงกว่าเล็กน้อยเมื่อเทียบกับโมเดลที่พื้นฐานที่ถูกเลือก อย่างไรก็ตาม จากที่เราได้ค้นพบ ระยะเวลาการประมวลผลของซีสิสการ์ด (cSysGuard) นั้นขึ้นอยู่กับความซับซ้อนของโมเดลที่ถูกใช้ภายใน การใช้โมเดลภายในที่เรียบง่ายกว่าก็สามารถลดเวลาการประมวลผลได้เป็นอย่างดี อีกทั้งการเพิ่มขนาดทรัพยากรของเครื่องโฮสต์ที่ใช้งานอยู่นั้นก็สามารถลดเวลาได้เช่นกัน

ตารางที่ 6.9 การวิเคราะห์เปรียบเทียบเวลาทำนายและปรับปรุงสำหรับโมเดลที่ถูกเลือกและซีสิสการ์ด (cSG)

(หน่วย: วินาที)	RNN	GRU	LSTM	cSysGuard
เวลาการทำนาย	0.382	0.362	1.061	2.366
เวลาการปรับปรุงโมเดล	74.29	132.40	141.84	152.55

6.5 การประเมิน RMSE ของลำดับตัวกรอง Butterworth กับข้อมูลที่มีขอบเขตสูง

ในการศึกษานี้ เราได้ใช้ตัวกรองแบบ Butterworth แบบ Low Pass ในลำดับที่ 3 และ Normalized Cutoff Frequency ที่ 0.32 เพื่อกรองสัญญาณรบกวนที่มีความถี่สูงออก ในขณะที่ยังคงรักษาสัญญาณหลักของข้อมูลเราไว้ ต่อมา งานวิจัยได้มุ่งเน้นทำความเข้าใจประสิทธิภาพของตัวกรองในช่วงข้อมูลที่มีความผันผวนสูง เพื่อให้ซีสิสการ์ด (cSysGuard) ตรวจจับจุด Peak ของข้อมูลได้ดียิ่งขึ้น ซึ่งมักจะเป็นช่วงเวลาที่สามารถบ่งชี้ถึงความล้มเหลวของระบบได้เป็นอย่างดี รูปที่ 6.8 แสดงถึงข้อมูลการใช้ CPU ที่ถูกปรับโดยใช้ระหว่างค่าลำดับและ Normalized Cutoff Frequency ของตัวกรอง Butterworth



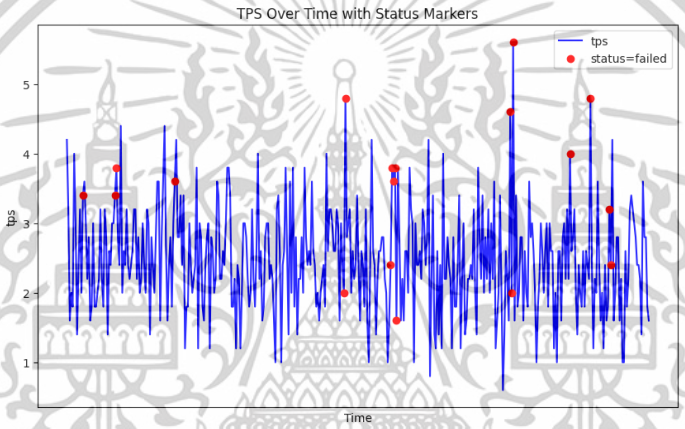
รูปที่ 6.8 ผลการปรับ Order และ Normalized Cutoff Frequency ของ Butterworth เพื่อทำนายการใช้ CPU

จากผลที่แสดงให้เห็น เราค้นพบว่าถ้าหากต้องการที่จะให้ซีสิสการ์ด (cSysGuard) มีความสามารถในการตรวจจับข้อมูลที่จุด Peak ได้ดียิ่งขึ้น เราต้องปรับค่า Normalized Cutoff Frequency ซึ่งเป็นค่าที่มีปัจจัยสำคัญในการส่งผลการเปลี่ยนแปลงความถี่ของข้อมูล เพราะเหตุนี้ งานวิจัยได้ทดลองเพิ่มเติมโดยการปรับค่า Cutoff ในช่วงที่เราได้กำหนดไว้ พร้อมกับ Order 3 ที่เหมือนเดิม เพื่อสังเกตการเปลี่ยนแปลงของ RMSE โดยรวม และ RMSE ของข้อมูลจุดสูงสุด 10% จากทั้งหมด ด้วยการทำนายการใช้ CPU และผลได้แสดงให้เห็นตาราง 6.10 ด้านล่างดังนี้

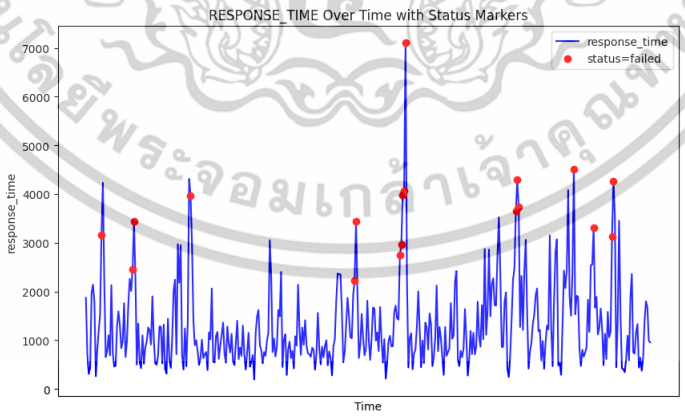
ตารางที่ 6.10 เปรียบเทียบ RMSE โดยรวมและ RMSE ของข้อมูลจุดสูงสุด 10% ในการทำนายคำขอ CPU ด้วยค่า Normalized Cutoff Frequency ของ Butterworth ที่แตกต่างกันไป

Normalized Cutoff Frequency	RMSE โดยรวม	RMSE ของข้อมูลจุดสูงสุด 10%
0.05	0.053	0.082
0.1	0.046	0.079
0.2	0.050	0.066
0.3	0.054	0.083
0.4	0.058	0.105
0.5	0.060	0.106

โดยแต่ค่าของ Cutoff ได้แสดงผล RMSE ที่ต่างกันไป บางค่านั้นสามารถให้ RMSE รวมได้ดีกว่าเมื่อเทียบกันค่าอื่น แต่ถ่าก็ไม่สามารถให้ค่า RMSE ของข้อมูลจุดสูงสุดได้ไม่ได้ดีเท่า (ตัวอย่างอย่างค่า Cutoff ที่ 0.1 และ 0.2) ด้วยเหตุนี้ นอกจากที่เราควรจะคำนึงถึงค่า RMSE โดยรวมแล้ว เราก็ควรที่จะ RMSE ที่ตัวทำนายจะสามารถตรวจจับค่า ณ จุดสูงสุดของข้อมูลอีกด้วย ซึ่งอาจเป็นส่วนสำคัญในการบ่งบอกการทำงานที่ล้มเหลวของระบบได้เป็นอย่างดี ตัวอย่างในรูปที่ 6.9 และ 6.10 ที่แสดงถึงจำนวนการทำธุรกรรมต่อวินาที (TPS) และเวลาตอบสนองโดยเฉลี่ย (RESPONSE TIME) พร้อมกับจุดที่บอกถึงช่วงเวลาที่ระบบล้มเหลวอยู่ตามลำดับ โดยที่ช่วงเวลาที่ล้มเหลวมักจะเกิดบริเวณ (หรือบ้างอาจเกิดหลัง) จุด Peak ของตัวชี้วัดนั้น ๆ



รูปที่ 6.9 จำนวนการทำธุรกรรมต่อวินาที (TPS) พร้อมกับจุดที่บอกช่วงล้มเหลวของระบบ



รูปที่ 6.10 เวลาตอบสนองโดยเฉลี่ยพร้อมกับจุดที่บอกช่วงล้มเหลวของระบบ

6.6 ข้อจำกัดของการกำหนดค่าระบบที่เฉพาะเจาะจง

งานวิจัยนี้ได้ดำเนินการติดตั้งระบบด้วยขนาดทรัพยากรที่เฉพาะเจาะจง ตัวอย่างเช่น CPU หน่วยความจำ หรือระบบปฏิบัติการที่ใช้ สิ่งสำคัญคือต้องยอมรับว่าความแตกต่างในข้อกำหนดเหล่านี้ว่ามันสามารถส่งผลกระทบต่อกระบวนการจัดทำโมเดลโดยรวมและผลลัพธ์การทำนายได้ การเปลี่ยนแปลงดังกล่าวอาจส่งผลต่อทำงานโดยรวมของซีสิสการ์ด (cSysGuard) ดังนั้น เราจึงแนะนำให้ผู้ใช้พิจารณาเลือกใช้โมเดลและปรับแต่งพารามิเตอร์ให้เหมาะสมให้สอดคล้องกับเซิร์ฟเวอร์ของตนเพื่อรักษาประสิทธิภาพการทำนายไว้ให้ที่ดีที่สุด โดยการปรับแต่งนี้เป็นสิ่งสำคัญในการบรรลุประสิทธิภาพสูงที่สุดในการตั้งค่าระบบที่แตกต่างกัน

บทที่ 7

บทสรุป

งานวิจัยนี้ได้นำเสนอระบบที่มีชื่อว่าซีสิสการ์ด (cSysGuard) โดยจะประกอบไปด้วยสองส่วนหลัก ในส่วนแรก ระบบจะมีการใช้การทำนายแบบถดถอย (Regression) ในรูปแบบ Stacking โดยจะประกอบไปด้วยโมเดลต่าง ๆ เพื่อช่วยกันพยากรณ์การใช้งานทรัพยากรและประสิทธิภาพของซอฟต์แวร์ที่อยู่บนคลาวด์ด้วยสภาพแวดล้อมที่ซับซ้อน (Dynamic Workload) ให้ออกมามีประสิทธิภาพที่ดีขึ้น โดยข้อมูลจะประกอบไปด้วยการใช้งาน CPU หน่วยความจำ (Memory) แบนด์วิดท์ขาเข้าและออก (Inbound & Outbound Bandwidth) จำนวนการทำธุรกรรมต่อวินาที (Transactions Per Second) และเวลาตอบกลับของซอฟต์แวร์โดยเฉลี่ย (Average Response Time) และหลักจากการพยากรณ์ที่เสร็จสิ้น ทางระบบจะนำผลลัพธ์ที่ได้ส่งไปยังในส่วนที่สอง นั้นก็คือการประเมินสถานะของซอฟต์แวร์ ณ เวลาทำนายตอนนั้นโดยใช้หลักการจำแนกประเภท (Classification) ของข้อมูล ซึ่งจะประกอบไปด้วยสถานะปกติ (Healthy) และผิดปกติ (Unhealthy) โดยจากการค้นพบในการศึกษานี้ ซีสิสการ์ด (cSysGuard) มีประสิทธิภาพที่เพิ่มขึ้น (สูงสุดถึง 2.28 เท่า) เมื่อเทียบ RMSE กับกว่าโมเดลถดถอยทั่วไป ซึ่งประกอบไปด้วยรูปแบบที่ใช้ทางสถิติหรือการเรียนรู้ขั้นสูง อีกทั้งระบบได้ใช้ Decision Trees ที่ได้รับการปรับแต่งไฮเปอร์พารามิเตอร์ เพื่อที่จะประเมินสถานะของซอฟต์แวร์ โดยให้ F1 Score อยู่ที่ 89.79% ซึ่งมีคะแนนสูงสุดเมื่อเทียบกับโมเดลจำแนกที่ทางงานวิจัยได้เลือกมาทั้งหมด

ในงานวิจัยในอนาคต เราตั้งใจที่จะขยายความสามารถของซีสิสการ์ด (cSysGuard) ให้เกินกว่าการมุ่งเน้นไปที่การตรวจจับโอกาสล้มเหลวของระบบเซิร์ฟเวอร์ตามการใช้ทรัพยากร ณ เวลานั้น งานวิจัยในอนาคตจะพิจารณาที่จะเพิ่มการตรวจสอบโอกาสล้มเหลวของระบบในรูปแบบอื่น ๆ เช่น ปัญหาที่เกี่ยวข้องกับเครือข่าย การละเมิดความปลอดภัย และข้อผิดพลาดในระดับแอปพลิเคชั่น เป็นต้น ด้วยการจัดการกับมิติที่เพิ่มเติมเหล่านี้ซีสิสการ์ด (cSysGuard) จะเป็นระบบที่มีความหลากหลายมากขึ้น สามารถปรับให้เข้ากับสถานการณ์ที่กว้างขึ้น และมีความสามารถในการตรวจสอบสถานะระบบเซิร์ฟเวอร์อย่างครอบคลุมในแง่มุมต่าง ๆ ที่มากขึ้นไป

เอกสารอ้างอิง

A. Bankole and S. A. Ajila. “Predicting cloud resource provisioning using machine learning techniques,” in Proc. 26th IEEE Canadian Conf. on Electrical and Computer Engineering (CCECE), Regina, SK, Canada, 2013, pp. 1-4, DOI: 10.1109/CCECE.2013.6567848.

A. C. Harvey. “ARIMA Models,” in J. Eatwell, M. Milgate, and P. Newman, Eds., Time Series and Statistics, The New Palgrave. London: Palgrave Macmillan, 1990. [Online]. Available: https://doi.org/10.1007/978-1-349-20865-4_2.

A. Tariq. “What is the difference between micro and macro averaging?” Educative. [Online]. Available: https://www.educative.io/answers/what-is-the-difference-between-micro-and-macro-averaging. 2023.

Amazon Web Services. “EC2 Auto Scaling.” [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html. 2023.

B. Müller, J. Reinhardt, and M. T. Strickland. “Neural Networks: An Introduction”, 2nd ed., Physics of Neural Networks. Springer Berlin, Heidelberg, 1995. [Online]. Available: https://doi.org/10.1007/978-3-642-57760-4.

B. Saji. “Elbow Method for Finding the Optimal Number of Clusters in K-Means.” Analytics Vidhya. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning. 2024

Boston, Springer. “Butterworth Filters,” in Design and Analysis of Analog Filters, The International Series in Engineering and Computer Science, vol. 617, Boston, MA: Springer, 2003. [Online]. Available: https://doi.org/10.1007/0-306-48012-3_3.

C. Cortes and V. Vapnik. “Support-vector networks,” Mach. Learn., vol. 20, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018.

C. Lea, R. Vidal, A. Reiter, and G. D. Hager. “Temporal Convolutional Networks: A Unified Approach to Action Segmentation,” in G. Hua and H. Jégou, Eds., Computer Vision – ECCV 2016 Workshops, ECCV 2016, Lecture Notes in Computer Science, vol. 9915, Cham: Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-49409-8_7.

D. Lee, D. Lee, M. Choi, and J. Lee. “Prediction of Network Throughput using ARIMA,” in Proc. 2020 Int. Conf. on Artificial Intelligence in Information and Communication (ICAIIC), Fukuoka, Japan, 2020, pp. 1-5, doi: 10.1109/ICAIIC48513.2020.9065083.

D. Schumacher, “Understanding Weighted Average.” SERP AI. [Online]. Available: https://serp.ai/weighted-average. 2024

Digital Ocean. [Online]. Available: https://www.digitalocean.com. 2024.

E. Caron, F. Desprez, and A. Muresan. “Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching,” in Proc. IEEE Second Int. Conf. on Cloud Computing Technology and Science, Indianapolis, IN, USA, 2010, pp. 456-463, DOI: 10.1109/CloudCom.2010.65.

G. James, D. Witten, T. Hastie, and R. Tibshirani. “Linear Regression,” in An Introduction to Statistical Learning, Springer Texts in Statistics, New York, NY: Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-1-0716-1418-1_3.

Google Cloud. “Autoscaling groups of instances” [Online]. Available: https://cloud.google.com/compute/docs/autoscaler. 2023

Google Cloud. “Scaling based on predictions.” [Online]. Available: https://cloud.google.com/compute/docs/autoscaler/predictive-autoscaling. 2024

Google Cloud. “What is a Virtual Machine?” [Online]. Available: https://cloud.google.com/learn/what-is-a-virtual-machine. 2024.

I. K. Kim, W. Wang, Y. Qi, and M. Humphrey. “Forecasting Cloud Application Workloads With CloudInsight for Predictive Resource Management,” in IEEE Trans. on Cloud Computing, vol. 10, no. 3, pp. 1848-1863, Jul.-Sept. 2022, DOI: 10.1109/TCC.2020.2998017.

J. Brownlee. “A Gentle Introduction to Imbalanced Classification.” Machine Learning Mastery. [Online]. Available: https://machinelearningmastery.com/what-is-imbalanced-classification. 2020.

J. Gao, H. Wang, and H. Shen. “Machine Learning Based Workload Prediction in Cloud Computing,” in Proc. 29th Int. Conf. on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 2020, pp. 1-9, DOI: 10.1109/ICCCN49398.2020.9209730.

J. H. Challis. “Signal Processing,” in Experimental Methods in Biomechanics, Cham: Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-52256-8_4.

J. R. Quinlan. “Induction of decision trees,” Mach. Learn., vol. 1, pp. 81–106, 1986. [Online]. Available: https://doi.org/10.1007/BF00116251.

J. S. Cramer. “The origins and development of the logit model”' in Logit Models from Economics and Other Fields, Cambridge: Cambridge University Press, 2003, pp. 149-157. DOI: 10.1017/CBO9780511615412.010.

K. Dmytro, T. Sergii, and P. Andiy. “ARIMA forecast models for scheduling usage of resources in IT-infrastructure,” in Proc. 12th Int. Sci. and Tech. Conf. on Computer Sciences and Information Technologies (CSIT)}, Lviv, Ukraine, 2017, pp. 356-360, DOI: 10.1109/STC-CSIT.2017.8098804.

K. Smagulova and A. P. James. “Overview of Long Short-Term Memory Neural Networks,” in A. James, Ed., Deep Learning Classifiers with Memristive Networks, Modeling and Optimization in Science and Technologies, vol. 14, Cham: Springer, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-14524-8_11.

L. Rosencrance, S. Louissaint, and K. Brush. “Service-Level Agreement (SLA).” [Online]. Available: https://www.techtarget.com/searchitchannel/definition/service-level-agreement. 2024

M. A. S. Netto, C. Cardonha, R. L. F. Cunha, and M. D. Assuncao. “Evaluating Auto-scaling Strategies for Cloud Computing Environments,” in Proc. IEEE 22nd Int. Symp. on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, Paris, France, 2014, pp. 187-196, DOI: 10.1109/MASCOTS.2014.32.

M. Hassan, H. Chen, and Y. Liu. “DEARS: A Deep Learning Based Elastic and Automatic Resource Scheduling Framework for Cloud Applications,” in Proc. IEEE Intl Conf. on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), Melbourne, VIC, Australia, 2018, pp. 541-548, DOI: 10.1109/BDCloud.2018.00086.

MDN Web Docs: HTTP response status codes. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. 2023

N. Chairatana and R. Chawuthai. “Cloud Stateless Server Failover Prediction Using Machine Learning on Proactive System Metrics,” in Proc. 18th Int. Joint Symp. on Artificial Intelligence and Natural Language Processing (iSAI-NLP), Bangkok, Thailand, 2023, pp. 1-6, DOI: 10.1109/iSAI-NLP60301.2023.10354585.

N. Chairatana and R. Chawuthai. “Cloud Stateless System Performance Metrics and Status.” IEEE Dataport, DOI: https://dx.doi.org/10.21227/8wf2-2y40. 2024.

N. Suksriupatham and A. Hoonlor. “Workload Prediction with Regression for Over and Under Provisioning Problems in Multi-agent Dynamic Resource Provisioning Framework,” In Proc. 17th Int. Joint Conf. on Computer Science and Software Engineering (JCSSE), Bangkok, Thailand, 2020, pp. 128-133, DOI: 10.1109/JCSSE49651.2020.9268289.

P. N, S. Chhetri, S. R. Kini, G. G. Mishra, and S. Kumar. “Resource Provisioning in Cloud using ARIMA and LSTM Technique,” in Proc. IEEE 2nd Mysore Sub Section Int. Conf. (MysuruCon), Mysuru, India, 2022, pp. 1-7, DOI: 10.1109/MysuruCon55714.2022.9972689.

P. Sekwatlakwatla, M. Mphahlele, and T. Zuva. “Traffic flow prediction in cloud computing," in Proc. Int. Conf. on Advances in Computing and Communication Engineering (ICACCE), Durban, South Africa, 2016, pp. 123-128, DOI: 10.1109/ICACCE.2016.8073735.

P. Wenda. “Autoscaling: Introducing Compute Engine Predictive Autoscaling.” Google Cloud, [Online]. Available: https://cloud.google.com/blog/products/compute/introducing-compute-engine-predictive-autoscaling. 2021.

Prometheus. [Online]. Available: https://prometheus.io. 2023.

Q. Zhang and R. Boutaba. “Dynamic workload management in heterogeneous Cloud computing environments,” Proc. IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 2014, pp. 1-7, DOI: 10.1109/NOMS.2014.6838288.

R. Dey and R. Mathur. “Ensemble Learning Method Using Stacking with Base Learner, A Comparison,” in N. Chaki, N. D. Roy, P. Debnath, and K. Saeed, Eds., Proceedings of International Conference on Data Analytics and Insights, ICDAI 2023, Lecture Notes in Networks and Systems, vol. 727, Singapore: Springer, 2023. [Online]. Available: https://doi.org/10.1007/978-981-99-3878-0_14.

R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder. “Forecasting with Exponential Smoothing: The State Space Approach,” Springer, 2008. [Online]. Available: https://doi.org/10.1007/978-3-540-71918-2.

R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, “Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS,” in IEEE Trans. on Cloud Computing, vol. 3, no. 4, pp. 449-458, Oct.-Dec. 2015, DOI: 10.1109/TCC.2014.2350475.

Red Hat. “Stateful vs stateless.” [Online]. Available: https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless. 2023.

S. A. Marhon, C. J. F. Cameron, and S. C. Kremer. “Recurrent Neural Networks,” in M. Bianchini, M. Maggini, and L. Jain, Eds., Handbook on Neural Information Processing, Intelligent Systems Reference Library, vol. 49, Berlin, Heidelberg: Springer, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-36657-4_2.

S. Bhagavathiperumal and M. Goyal. “Workload Analysis of Cloud Resources using Time Series and Machine Learning Prediction,” in Proc. IEEE Asia-Pacific Conf. on Computer Science and Data Engineering (CSDE), Melbourne, VIC, Australia, 2019, pp. 1-8, DOI: 10.1109/CSDE48274.2019.9162385.

S. Kostadinov. “Understanding GRU Networks,” Towards Data Science. [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be. 2017

S. Li, J. Bi, H. Yuan, M. Zhou, and J. Zhang. “Improved LSTM-based Prediction Method for Highly Variable Workload and Resources in Clouds,” in Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 2020, pp. 1206-1211, DOI: 10.1109/SMC42975.2020.9283029.

S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen. “Cost-effective cloud HPC resource provisioning by building Semi-Elastic virtual clusters,” in Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis (SC '13), Denver, CO, USA, 2013, pp. 1-12.

S. Satpathy. “SMOTE for Imbalanced Classification with Python.” Analytics Vidhya. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques.

S. Skansi. “Feedforward Neural Networks,” in Introduction to Deep Learning, Undergraduate Topics in Computer Science, Cham: Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-73004-2_4.

S. T. Singh, M. Tiwari, and A. S. Dhar. “Machine Learning based Workload Prediction for Auto-scaling Cloud Applications,” in Proc. OPJU Int. Tech. Conf. on Emerging Tech. for Sustainable Development (OTCON), Raigarh, Chhattisgarh, India, 2023, pp. 1-6, DOI: 10.1109/OTCON56053.2023.10114033.

S. Tripathy and R. Singh. “Convolutional Neural Network: An Overview and Application in Image Classification,” in R. C. Poonia, V. Singh, D. Singh Jat, M. J. Diván, and M. S. Khan, Eds., Proceedings of Third International Conference on Sustainable Computing, Advances in Intelligent Systems and Computing, vol. 1404, Singapore: Springer, 2022. [Online]. Available: https://doi.org/10.1007/978-981-16-4538-9_15.

S. Turney. “Pearson Correlation Coefficient.” Scribbr. [Online]. Available: https://www.scribbr.com/statistics/pearson-correlation-coefficient. 2023.

ScienceDirect, “Integer Overflow.” [Online]. Available: https://www.sciencedirect.com/topics/computer-science/integer-overflow. 2024

Scikit-learn. [Online] Available: https://www.scikit-learn.org. 2023.

Statsmodels. [Online]. Available: https://www.statsmodels.org. 2023.

Synopsys. “How Cloud Computing Flexibility Enables Design Changes.” [Online]. Available: https://www.synopsys.com/cloud/insights/cloud-computing-flexibility.html. 2024.

T. Cover and P. Hart. “Nearest neighbor pattern classification,” in IEEE Trans. on Information Theory, vol. 13, no. 1, pp. 21-27, Jan. 1967, DOI: 10.1109/TIT.1967.1053964.

T. Mehmood, S. Latif, and S. Malik. “Prediction Of Cloud Computing Resource Utilization,” in Proc. 15th Int. Conf. on Smart Cities: Improving Quality of Life Using ICT & IoT (HONET-ICT), Islamabad, Pakistan, 2018, pp. 38-42, DOI: 10.1109/HONET.2018.8551339.

TensorFlow. [Online] Available: https://www.tensorflow.org. 2023.

VMware by Broadcom. “What is Cloud Scalability?” [Online]. Available: https://www.vmware.com/topics/glossary/content/cloud-scalability.html. 2024.

W. Chen, C. Lu, K. Ye, Y. Wang, and C. -Z. Xu. “RPTCN: Resource Prediction for High-dynamic Workloads in Clouds based on Deep Learning,” in Proc. IEEE Int. Conf. on Cluster Computing (CLUSTER), Portland, OR, USA, 2021, pp. 59-69, DOI: 10.1109/Cluster48925.2021.00038.

W. Wang. “Bayesian Optimization Concept Explained in Layman Terms.” Towards Data Science. [Online]. Available: https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f. 2020

Y. Liu, Y. Wang, and J. Zhang. “New Machine Learning Algorithm: Random Forest,” in B. Liu, M. Ma, and J. Chang, Eds., Information Computing and Applications, Lecture Notes in Computer Science, vol. 7473, Berlin, Heidelberg: Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-34062-8_32.

ภาคผนวก

ภาคผนวก ก.

รหัสการตอบกลับของ HTTP

ตารางที่ ก.1 ความหมายของแต่ละ HTTP Response Code

	รหัส	นิยาม	ความหมาย
100-199 Informative Responses	100	Continue	การตอบกลับแบบชั่วคราวที่บ่งบอกว่า Client ควรดำเนิน Request ต่อหรือไม่ต้องสนใจ Response หาก Request ถูกดำเนินการเสร็จสิ้นแล้ว
	101	Switching Protocols	รหัสนี้ถูกส่งเพื่อเป็นการตอบกลับไปยัง Upgrade Request Header ที่มาจาก Client เพื่อบ่งบอกถึงโปรโตคอลที่เซิร์ฟเวอร์กำลังจะเปลี่ยนไป
	102	Processing	รหัสนี้บ่งบอกว่าเซิร์ฟเวอร์ได้รับ Request แล้วและกำลังดำเนินการ แต่ยังไม่มี Response ที่พร้อมจะให้ได้ในขณะนี้
	103	Early Hints	รหัสสถานะนี้มีวัตถุประสงค์หลักในการใช้ร่วมกับ Link Header เพื่อให้ User Agent เริ่มทำการโหลดทรัพยากรล่วงหน้าในขณะที่เซิร์ฟเวอร์เตรียม Response หรือเชื่อมต่อล่วงหน้าไปยังต้นกำเนิดของทรัพยากรที่ทางเว็บจะต้องใช้
200-299 Successful Responses	200	OK	Request สำเร็จ ความหมายของ “สำเร็จ” ขึ้นอยู่กับวิธีการเรียก HTTP: • **GET**: ทรัพยากรได้ถูกเรียกและส่งเนื้อหาผ่าน Message Body • **HEAD**: ส่วน Headers จะถูกรวมไว้ใน Response โดยไม่มี Message Body • **PUT หรือ POST**: ทรัพยากรที่ถูกดำเนินการตามคำขอจะถูกส่งผ่านใน Message Body • **TRACE**: Message Body มี Request Message ที่ถูกได้รับโดยเซิร์ฟเวอร์
	201	Created	Request สำเร็จและทรัพยากรใหม่ถูกสร้างขึ้น โดยทั่วไปเป็นการตอบกลับที่ส่งหลังจาก Request ที่เป็น POST หรือ PUT
	202	Accepted	Request ได้รับแล้วแต่ยังไม่ได้ดำเนินการ ไม่มีการรับประกัน เนื่องจากไม่มีวิธีใน HTTP ที่จะส่ง Response แบบ Asynchronous ที่บ่งบอกถึงผลลัพธ์ของ Request ได้ในภายหลัง มีไว้สำหรับกรณีที่กระบวนการอื่นหรือเซิร์ฟเวอร์อื่นจัดการกับ Request หรือสำหรับการประมวลผลเป็นชุด (Batching)
	203	Non-Authoritative Information	รหัสการตอบกลับนี้หมายความว่าข้อมูลที่ส่งกลับไม่เหมือนกับที่มีอยู่จริงที่เซิร์ฟเวอร์ต้นทาง แต่เป็นข้อมูลที่รวบรวมมาจาก Local หรือ Third-party Copy โดยทั่วไป สิ่งนี้จะถูกใช้ไว้สำหรับก็อปปี้หรือสำรองทรัพยากรอื่นเท่านั้น นอกจากกรณีนี้ การตอบกลับแบบ 200 OK มักจะได้รับความนิยมมากกว่า

	204	No Content	ไม่มีเนื้อหาที่จะส่งให้กับ Request นี้ แต่ส่วน Header ที่อาจมีประโยชน์ User Agent อาจอัพเดตส่วน Header ที่เก็บใน Cache ไว้สำหรับทรัพยากรอันใหม่ที่เข้ามา
	205	Reset Content	แจ้งให้ User Agent รีเซ็ตเอกสารที่ส่ง Request นี้
	206	Partial Content	รหัสการตอบกลับนี้ใช้เมื่อมีการส่งหัวข้อส่วน Range จากลูกค้าเพื่อขอเพียงบางส่วนของทรัพยากร
	207	Multi-Status	ส่งข้อมูลเกี่ยวกับทรัพยากรหลาย ๆ อย่าง ซึ่งไว้ใช้ในสถานการณ์ที่อาจจำเป็นต้องมีหลาย Status Code
	208	Already Response	ใช้ใน <dav:propstat> Response เพื่อหลีกเลี่ยงการเรียกซ้ำของ Internal Members ที่ไปยัง Collection เดียวกัน
	226	IM Used	เซิร์ฟเวอร์ได้ตอบสนอง Request GET และ Response เป็นการแสดงผลลัพธ์ของการจัดการหนึ่งหรือหลายตัวอย่างที่ใช้กับตัวปัจจุบัน
300-399 Redirection Messages	300	Multiple Choices	ไว้สำหรับ Request ที่สามารถมีคำตอบได้มากกว่าหนึ่ง และ User Agent หรือ User ควรเลือกหนึ่งในนั้น (ไม่มีวิธีมาตรฐานในการเลือกคำตอบ แต่แนะนำให้ใช้ลิงก์ HTML เพื่อให้ User สามารถเลือกได้)
	301	Moved Permanently	URL ของทรัพยากรที่ถูกเรียกขอมีการเปลี่ยนแปลงไปอย่างถาวร URL ใหม่จะถูกให้ในการตอบกลับ
	302	Found	รหัสการตอบกลับนี้หมายความว่า URI ของทรัพยากรที่ร้องขอมีการเปลี่ยนแปลงชั่วคราว อาจมีการเปลี่ยนแปลงเพิ่มเติมใน URI ในอนาคต ดังนั้น URI เดียวกันนี้ควรถูกใช้โดย Client ใน Request อนาคต
	303	See Other	เซิร์ฟเวอร์ส่ง Response นี้เพื่อนำให้ Client ไปรับทรัพยากรที่ร้องขอที่ URI อื่นด้วยคำขอ GET
	304	Not Modified	ใช้สำหรับในการเก็บข้อมูล Cache โดยจะบอก Client ว่า Response ไม่ได้มีการเปลี่ยนแปลง ดังนั้น Client สามารถใช้ Response ที่เก็บไว้ใน Cache เดิมได้
	305	Use Proxy	ถูกใช้ในเวอร์ชั่นก่อนของการกำหนด HTTP เพื่อระบุว่า Response ที่ร้องขอต้องถูกเข้าถึงผ่าน Proxy โดยรหัสการตอบกลับนี้ถูกถอดออกเนื่องจากมีปัญหาเกี่ยวกับความปลอดภัยในการตั้งค่า In-band ของ Proxy
	306	Unused	รหัสการตอบกลับนี้ไม่ได้ถูกใช้งานอีกต่อไป ไว้เพียงสำรองเท่านั้น จะถูกใช้ในเวอร์ชั่นก่อนหน้าที่เป็นของ HTTP/1.1
	307	Temporary Redirect	เซิร์ฟเวอร์ส่ง Response เพื่อนำ Client ไปรับทรัพยากรที่ร้องขอที่ URI อื่นด้วยวิธีการเดียวกับที่ใช้ใน Request ก่อนหน้า ซึ่งมีความหมายเดียวกันกับ

			รหัสการตอบกลับ HTTP 302 Found แต่ข้อแตกต่างคือ User Agent ต้องไม่เปลี่ยน HTTP Method ที่ใช้: ถ้าใช้ POST ใน Request แรก ต้องใช้ POST ใน Request ที่สองด้วย
	308	Permanent Redirect	รหัสการตอบกลับนี้หมายความว่าทรัพยากรอยู่อย่างถาวรที่ URI อื่น ระบุโดย Location ที่อยู่ใน HTTP Response Header ซึ่งมีความหมายเดียวกันกับรหัสการตอบกลับ HTTP 301 Moved Permanently แต่ข้อแตกต่างคือ User Agent ต้องไม่เปลี่ยน HTTP Method ที่ใช้: ถ้าใช้ POST ใน Request แรก ต้องใช้ POST ใน Request ที่สองด้วย
400-499 Client Error Responses	400	Bad Request	เซิร์ฟเวอร์ไม่สามารถหรือไม่ยอมประมวลผลคำขอเนื่องจากมองว่ามีข้อผิดพลาดจาก Client (เช่น ไวยากรณ์ Request ผิดพลาด, การจัดการข้อความ Request ไม่ถูกต้อง, หรือเป็นเส้นทาง Request ที่หลอกลวง)
	401	Unauthorized	แม้ว่า HTTP จะแปลว่า "ไม่ได้รับอนุญาต" Response นี้หมายถึง "ไม่ได้รับการตรวจสอบตัวตน" นั่นคือ Client ต้องยืนยันตัวตนเพื่อที่จะรับ Response จากที่ขอมา
	402	Payment Required	รหัสการตอบกลับนี้ถูกสำรองไว้สำหรับการใช้งานในอนาคต วัตถุประสงค์ในการสร้างรหัสนี้คือการใช้สำหรับระบบชำระเงินดิจิทัล อย่างไรก็ตาม รหัสสถานะนี้ถูกใช้งานน้อยมากและยังไม่มีมาตรฐานที่เป็นที่ยอมรับ
	403	Forbidden	Client ไม่มีสิทธิ์เข้าถึงเนื้อหา นั่นคือ ไม่ได้รับอนุญาต ดังนั้นเซิร์ฟเวอร์จึงปฏิเสธที่จะให้ทรัพยากรที่ร้องขอไว้ โดยต่างกับ 401 Unauthorized เนื่องจาก ตัวตนของ Client ในที่นี้จะเป็นที่รู้จักของเซิร์ฟเวอร์อยู่แล้ว
	404	Not Found	เซิร์ฟเวอร์ไม่สามารถหาทรัพยากรที่ถูกร้องขอได้ ใน Browser หมายความว่า URL นั้นเป็นที่ไม่รู้จัก ใน API อาจหมายความว่า endpoint มีอยู่แต่ทรัพยากรเองไม่มี บางครั้งเซิร์ฟเวอร์อาจส่งการตอบกลับนี้แทน 403 Forbidden เพื่อซ่อนการมีอยู่ของทรัพยากรจาก Client ที่ไม่ได้รับอนุญาต รหัสการตอบกลับนี้เป็นที่รู้จักมากที่สุดเนื่องจากเกิดขึ้นบ่อยครั้งบนเว็บ Browser
	405	Method Not Allowed	เป็น HTTP Method ที่เซิร์ฟเวอร์รู้จักแต่ไม่ได้รับการสนับสนุนโดยทรัพยากรที่เป็นเป้าหมาย ตัวอย่างเช่น API อาจไม่อนุญาตให้เรียก DELETE เพื่อลบทรัพยากรนั้น ๆ
	406	Not Acceptable	หลังจากทำการเลือกเนื้อหาข้อมูลโดยเซิร์ฟเวอร์ (Server-driven Content Negotiation) Response นี้จะถูกส่งเมื่อเว็บเซิร์ฟเวอร์ไม่พบเนื้อหาที่ตรงตามเกณฑ์ตามที่ User Agent ได้กำหนดไว้

	407	Proxy Authentication Required	คล้ายกับ 401 Unauthorized แต่ต้องการการตรวจสอบตัวตนผ่าน Proxy
	408	Request Timeout	Response นี้ถูกส่งบนการเชื่อมต่อในรูปแบบ Idle โดยบางเซิร์ฟเวอร์ แม้จะไม่มี Client Request ก่อนหน้านี้ ซึ่งหมายความว่าเซิร์ฟเวอร์ต้องการปิดการเชื่อมต่อที่ไม่ได้ใช้งานนี้ Response นี้ถูกใช้มากขึ้นเนื่องจากบาง Browser เช่น Chrome, Firefox 27+ หรือ IE9 ใช้กลไกการเชื่อมต่อล่วงหน้า (pre-connection) HTTP เพื่อเร่งการท่องเว็บ อีกทั้ง บางเซิร์ฟเวอร์อาจปิดการเชื่อมต่อโดยไม่ส่งข้อความนี้
	409	Conflict	Response นี้ถูกส่งเมื่อ Request ขัดแย้งกับสถานะปัจจุบันของเซิร์ฟเวอร์
	410	Gone	Response นี้ถูกส่งเมื่อเนื้อหาที่ร้องขอถูกลบออกจากเซิร์ฟเวอร์อย่างถาวร โดยไม่มีที่อยู่ส่งต่อ Client ถูกคาดหวังให้ลบข้อมูล Cache และลิงก์ไปยังทรัพยากรนั้น ข้อกำหนด HTTP ตั้งใจให้ใช้รหัสสถานะนี้ใช้สำหรับ "การให้บริการโปรโมชั่นชั่วคราว"
	411	Length Required	เซิร์ฟเวอร์ปฏิเสธคำขอเนื่องจาก Content-Length ใน HTTP Header ไม่ได้ถูกกำหนดแต่เซิร์ฟเวอร์ต้องการ
	412	Precondition Failed	Client ได้ระบุเงื่อนไขล่วงหน้าในส่วนของ HTTP Header แต่ไม่ตรงกับเงื่อนไขที่เซิร์ฟเวอร์ได้กำหนดไว้
	413	Payload Too Large	Entity ของ Request มีขนาดใหญ่กว่าขีดจำกัดที่เซิร์ฟเวอร์ได้กำหนดไว้ เซิร์ฟเวอร์อาจปิดการเชื่อมต่อหรือส่ง HTTP Header Retry-After กลับมา
	414	URI Too Long	URI ที่ Client ร้องขอยาวกว่าที่เซิร์ฟเวอร์จะนำมาใช้
	415	Unsupported Media Type	รูปแบบสื่อของข้อมูลที่ร้องขอไม่ได้รับการสนับสนุนจากเซิร์ฟเวอร์ ดังนั้นเซิร์ฟเวอร์จึงปฏิเสธ Request นี้
	416	Range Not Satisfiable	ช่วงที่ระบุใน HTTP Header Range ที่กำหนดใน Request ไม่สามารถเติมเต็มได้ ซึ่งอาจเป็นไปได้ว่าช่วงที่ระบุมานั้นอยู่เกินขอบเขตขนาดของข้อมูล URI ที่เป็นเป้าหมาย
	417	Expectation Failed	รหัส Response นี้หมายความว่าความคาดหวังที่ระบุโดย Expect ใน Request Header ไม่สามารถเป็นไปตามที่เซิร์ฟเวอร์ต้องการ
	418	I'm a teapot	เซิร์ฟเวอร์ปฏิเสธในความพยายามที่จะต้มกาแฟด้วยกาน้ำชา ซึ่งรหัสนี้ถูกใช้ในวันโกหกแห่งชาติ (April Fool's Jokes) ในปี 1998 และ 2014

	421	Misdirected Request	คำขอถูกส่งไปยังเซิร์ฟเวอร์ที่ไม่สามารถให้ Response ที่ต้องการได้ Response นี้อาจถูกส่งโดยเซิร์ฟเวอร์ที่ไม่ได้รับการตั้งค่าให้สามารถตอบได้ในการรวมกันของ Scheme และ Authority ที่อยู่ใน URI ของ Request
	422	Unprocessable Content	Request ถูกจัดอยู่ในรูปแบบเหมาะสมแต่ไม่สามารถดำเนินการคำขอได้เนื่องจากมีข้อผิดพลาดทางความหมาย
	423	Locked	ทรัพยากรที่กำลังเข้าถึงถูกล็อค
	424	Failed Dependency	Request ไม่สำเร็จเนื่องจาก Request ก่อนหน้าล้มเหลว
	425	Too Early	บ่งบอกว่าเซิร์ฟเวอร์ไม่สามารถที่จะเสี่ยงดำเนินการ Request ที่อาจถูกทำซ้ำไปแล้ว
	426	Upgrade Required	เซิร์ฟเวอร์ปฏิเสธที่จะดำเนินการ Request โดยใช้ Protocol ปัจจุบัน แต่อาจสามารถทำได้หลังจาก Client อัพเกรดไปใช้ Protocol อื่น เซิร์ฟเวอร์ส่ง Upgrade Header ใน 426 Response เพื่อระบุ Protocol ที่ต้องการ
	428	Precondition Required	เซิร์ฟเวอร์ต้นทางต้องการให้ Request มีเงื่อนไข โดย Response นี้มีวัตถุประสงค์เพื่อป้องกันปัญหา 'การอัพเดตที่หายไป'
	429	Too Many Requests	Client ส่ง Request มากเกินไปในช่วงเวลาที่กำหนดซึ่งถูกจำกัดโดย Rate Limiting
	431	Request Header Fields Too Large	เซิร์ฟเวอร์ไม่สามารถประมวลผล Request ได้เพราะ Header Fields มีขนาดที่ใหญ่เกินไป Request อาจถูกส่งใหม่หลังจากการลดขนาดของ Request Header Fields
	451	Unavailable For Legal Reasons	User Agent ร้องขอทรัพยากรที่ไม่สามารถให้ได้ตามกฎหมาย เช่น หน้าเว็บที่ถูกกำจัดการเข้าถึงโดยรัฐบาล
500-599 Server Error Responses	500	Internal Server Error	เซิร์ฟเวอร์พบปัญหาบางอย่างภายในที่ไม่สามารถจัดการได้
	501	Not Implemented	Request HTTP Method ไม่ได้รับการสนับสนุนจากเซิร์ฟเวอร์และไม่สามารถจัดการได้
	502	Bad Gateway	คำตอบผิดพลาดนี้หมายความว่าในขณะที่เซิร์ฟเวอร์ทำหน้าเป็น Gateway เพื่อรับ Response แต่เซิร์ฟเวอร์ได้รับ Response ไม่ถูกต้อง
	503	Service Unavailable	เซิร์ฟเวอร์ไม่พร้อมจัดการ Request สาเหตุทั่วไปได้แก่ เซิร์ฟเวอร์ที่ปิดเพื่อการปรับปรุงหรือมีโหลดที่มากเกินไป คำตอบนี้ควรใช้ในรูปแบบชั่วคราว และหากเป็นไปได้ Retry-After ใน HTTP Header ควรมีเวลาโดยประมาณที่เซิร์ฟเวอร์จะกู้คืนหรือกลับมา Webmaster มีหน้าที่ต้องดูแล Cache ที่

			เกี่ยวข้องกับ Header ที่ถูกส่งไปพร้อมกับ Response นี้ เนื่องจากการตอบกลับในรูปแบบชั่วคราวนี้ไม่ควรถูกเก็บใน Cache โดยปกติ
	504	Gateway Timeout	เป็น Response Error ที่เซิร์ฟเวอร์ทำหน้าที่เป็น Gateway และไม่สามารถรับ Response ทันเวลาได้
	505	HTTP Version Not Supported	เวอร์ชั่นของ HTTP ที่ใช้ใน Request ไม่ได้รับการสนับสนุนจากเซิร์ฟเวอร์
	506	Variant Also Negotiates	เซิร์ฟเวอร์มีข้อผิดพลาดในการตั้งค่าภายใน: ตัวแปรที่ถูกเลือกได้รับการตั้งค่าให้มีส่วนร่วมในการเลือกเนื้อหาข้อมูล (Content Negotiation) ด้วยตัวมันเอง ซึ่งไม่เหมาะสมที่จะเป็น Endpoint ในการเลือก
	507	Insufficient Storage	เซิร์ฟเวอร์ไม่สามารถดำเนินการได้เนื่องจากทรัพยากรนั้นไม่เพียงพอที่จะเก็บข้อมูลที่จำเป็นเพื่อทำให้ Request นั้นสำเร็จได้
	508	Loop Detected	เซิร์ฟเวอร์ตรวจพบลูปไม่สิ้นสุดในขณะประมวลผล Request
	510	Not Extended	จำเป็นต้องมีการขยาย Request เพิ่มเติมเพื่อให้เซิร์ฟเวอร์ดำเนินการต่อได้
	511	Network Authentication Required	ระบุว่าทาง Client ต้องยืนยันตัวตนเพื่อเข้าถึงเครือข่ายอินเตอร์เน็ต

ภาคผนวก ข.

ผลงานทางวิชาการที่ได้รับการตีพิมพ์

ผลงานทางวิชาการที่ได้รับการตีพิมพ์

N. Chairatana and R. Chawuthai, “Cloud Stateless Server Failover Prediction Using Machine Learning on Proactive System Metrics,” in Proc. 18th Int. Joint Symp. on Artificial Intelligence and Natural Language Processing (iSAI-NLP), Bangkok, Thailand, 2023, pp. 1-6, DOI: 10.1109/iSAI-NLP60301.2023.10354585

N. Chairatana and R. Chawuthai, "Stateless System Performance Prediction and Health Assessment in Cloud Environments: Introducing cSysGuard, an Ensemble Modeling Approach," in IEEE Access, vol. 12, pp. 78232-78247, 2024, doi: 10.1109/ACCESS.2024.3406670

Cloud Stateless Server Failover Prediction Using Machine Learning on Proactive System Metrics

Nutt Chairatana
Department of Robotics and AI Engineering,
School of Engineering,
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
0009-0009-9483-5925

Rathachai Chawuthai*
Department of Computer Engineering,
School of Engineering,
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
0000-0002-9303-4810

***Abstract*—Cloud computing, revered for its extraordinary scalability and elasticity, has revolutionized business operations by providing flexible resource options based on demand. However, this on-demand resource allocation poses distinct challenges. Due to the fluid nature of resource allocation and load distribution in the cloud, monitoring the health of servers using system metrics becomes problematic. This complexity can lead to unexpected server request failures and service interruptions due to resource insufficiencies, highlighting the need for more effective monitoring systems. Our research utilizes machine learning techniques to predict cloud server health based on resource usage and operational metrics, focusing specifically on stateless applications. Our study reveals that a Logistic Regression model trained on these system metrics delivers the most precise predictions. After hyperparameter tuning, the model exhibited robust performance, achieving a macro-averaged F1 Score of 97.7%. The paper outlines our methodology, findings, and the potential of this approach for cloud server health prediction.**

***Index Terms*—cloud server failover, stateless application, machine learning, non-functional testing, predictive maintenance, proactive system metrics, software engineering**

I. Introduction

Due to its remarkable capabilities, cloud computing has become a widespread infrastructure among industry and research organizations in recent years. Its scalability and flexibility allow for resource provisioning based on demand, ensuring optimal resource utilization and system performance with cost-effectiveness. The elasticity characteristic [1] has particularly captivated application developers, compelling them to migrate their applications to cloud environments.

Real-world application workloads' dynamic and unpredictable nature necessitates flexible resource allocation to maintain consistent server performance. However, the relationship between resource allocation and application performance is complex and frequently non-linear due to many influencing factors, including application characteristics, server configuration, and varying workload patterns. Consequently, relying solely on resource utilization metrics to determine the sufficiency of server resources can result in inaccuracy. It highlights the need for a more nuanced approach considering multiple factors when determining server health.

This paper aims to predict server failures in cloud environments utilizing machine learning techniques anchored on proactive system metrics. Our study centers on stateless applications [2], a significant category of server applications commonly used in microservices architecture [3]. By deeply comprehending these metrics' complex relationships and patterns, we constructed a predictive model to ascertain server sufficiency and potential failures.

Specifically, we gathered and used the system metrics to train the model. After determining the optimal base model, we assessed feature significance using feature importance, refined its performance with hyperparameter tuning, and evaluated the resulting enhancements. Furthermore, we investigated the impact of feature scaling on the models' performance and discussed the potential bias arising from system configurations.

II. Literature Review

This section provides a comprehensive synthesis of cloud computing backgrounds, reviews insights from prior studies, and discusses the techniques utilized in our research.

A. Cloud Adaptation: Stateless to Machine Learning

Cloud computing revolutionized business operations by providing scalability, flexibility, and elasticity. Scalability allows systems to adapt to growing workloads, flexibility facilitates quick adaptations to workload changes, and elasticity ensures automatic resource allocation based on demand. These form the core of resource allocation strategies in cloud services, allowing efficient handling of varying workloads [4].

In addition, stateless applications have become a key element in cloud computing amidst changing business operations. These applications do not store client data from one session to another, treating each request as a standalone transaction [5]. Such a design allows stateless applications to seamlessly adapt to the cloud's dynamic and ever-changing resource demands, ensuring efficient and reliable operation [6].

Stateless applications offer innovative solutions for managing cloud resources, but they need help in server resource monitoring. Despite having mechanisms like auto-scaling for

*Corresponding Author.

979-8-3503-7121-5/23/$31.00 ©2023 IEEE

dynamic resource adjustment, cloud resource management uses predefined rules or simple threshold-based techniques [7], [8], which may cause difficulties monitoring complex or extensive loads. These circumstances necessitate the surveillance of various metrics, making the task more difficult than traditional methods can efficiently handle.

Therefore, there is a growing interest in leveraging machine learning techniques within cloud computing environments to address the issue [9]–[13]. They offer optimistic results in detecting the onset of server insufficiency, enabling proactive issue resolution and optimal server performance.

B. Related Work

Substantial research on system failover prediction utilizes metrics from Google Trace data [14]. Asmawi, T. et al. [9], and Jassas, M.S. et al. [10] comprehensively evaluate predictive models for task failures using universal metrics such as Central Processing Unit (CPU) and memory utilization and disk storage. Their findings revealed a distinct correlation between failed jobs and these workload attributes. Similarly, our study employs CPU and memory requests to predict system failover. However, while many studies emphasize disk usage, often cited as a significant factor in various research, this approach does not align well with the characteristics of stateless applications, as they rely less on storage to fulfill requests, leading us not to employ storage as a model's input. To maintain robust prediction performance, we instead integrate more universally applicable metrics, such as bandwidth, Transactions Per Second (TPS), and average response time.

In addition, Liu X. et al. [11] propose a server crash prediction method for cloud service data centers, utilizing data from hardware indicators, kernel status, and system logs. However, reliance on log files introduces complications due to their unstructured nature, massive size, and the potential for irrelevant information. In contrast, our research utilizes globally consistent metrics readily accessible across all architectures.

Regarding hardware considerations, Hioual O. et al. [12] introduce a novel method for fault prediction in cloud computing. The proposed method assumes access to the hardware layer for measuring junction temperatures and adjusting cooling techniques, affording cloud load distribution decisions greater flexibility. However, this method's hardware-centric nature may limit its applicability for cloud-based users who primarily operate through virtual machines (VMs) [15]. In contrast to this method, our technique utilizes universally accessible system metrics for every cloud-based user.

C. Techniques

1) Feature Importance: In machine learning, feature importance is pivotal in optimizing model performance. This approach identifies features most informative to the output, improving the model's efficiency. Features of higher significance better predict the target variable [16], and by focusing on them, we can eliminate irrelevant data and reduce dimensionality.

2) Decision Trees: The models divide data into subsets based on feature values, forming a tree-like structure of decisions [17]. Due to its ability to handle both categorical and continuous data, this methodology has proven useful in diverse fields, including medical diagnosis, credit scoring, and energy consumption forecasting.

3) Support Vector Machines (SVM): Supervised learning models with associated learning algorithms that analyze data for classification and regression analysis [18] are highly efficient machine learning models renowned for their robustness, particularly in high-dimensional spaces [19]. SVM works on the principle of margin maximization, creating a hyperplane that separates data into classes with the maximum margin.

4) K-Nearest Neighbors (KNN): An instance-based learning algorithm built on the similarity principle [20]. This algorithm remains inactive until classification is needed. It has been successfully applied in various domains, such as recommendation systems, anomaly detection, and genetics.

5) Logistic Regression: A statistical model is employed for binary classification, estimating an event's probability using a logistic function [21]. It is widely used due to its simplicity, the absence of tuning parameters, and the well-calibrated predicted probabilities it offers. This model uses a logistic function to measure the relationship between a categorical dependent variable and one or more independent variables. Unlike other regression models, its coefficients are estimated using the maximum likelihood estimation method [22].

6) Neural Networks: The model draws inspiration from biological neuron structures, consisting of intricately linked layers of artificial neurons, also called nodes [23]. These layers comprise an initial input layer, one or several intermediate hidden layers, and a concluding output layer. Each node emulates the activation potential of a biological neuron. It is achieved by determining a weighted aggregate of its inputs and subsequently applying a non-linear transformation, frequently referred to as the activation function.

III. Methods

This section delineates the methodologies employed in our study for predicting server health through machine learning, covering methodology overview, data preparation, model comparison, feature importance, and hyperparameter tuning.

A. Methodology Overview

Fig. 1 presents an overview of our research methodology. Our approach follows a systematic process divided into three core components: application simulation, data handling, and model generation. Each stage is integral to predicting server health, using system metrics as the basis for analysis. To comprehensively understand our methodology, we will briefly describe each component, elucidating their functions and collaborative interactions within the system.

1) Application Simulation: This component emulates an application cloud environment, providing a means to acquire the system metrics dataset. In this study, we leveraged Digital Ocean's cloud service [24] to deploy three Linux virtual

TABLE I
TRANSFORMED SYSTEM METRICS WITH SERVER HEALTH IN 30 SECONDS

CPU Request	Memory Request	Inbound Bandwidth	Outbound Bandwidth	TPS	Average Response Time	Health
0.244	0.385	9060	8189.99	2.6	1080	0
0.442	0.374	21600	18100	3.2	1120	0
0.268	0.374	8950	7760	3.8	1890	0
0.342	0.367	11000	9510	2.8	2350	0
0.168	0.36	722	682	3.4	2820	1
0.134	0.36	760	653	4.8	4100	1

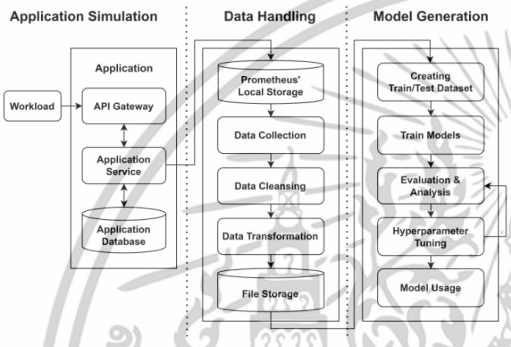


Fig. 1. This figure illustrates the process for predicting cloud server health based on the collected system metrics.

machines in this configuration, each with 1vCPU, 1GB memory, and a 10GB disk. The workload simulator, powered by Apache JMeter [25], generates the workload by randomly simulating various scenarios to let us evaluate the health of the cloud server under multiple conditions. The API gateway is responsible for routing requests to our stateless application service, a primarily focused component. To effectively carry out its functions, the application service utilizes the database as its primary storage for application data.

2) Data Handling: This component prepares a well-structured dataset for subsequent modeling and analysis. Prometheus [26], a system monitoring and alerting toolkit, captures and stores system metrics in its local storage mechanism. The data collector aggregates metrics from Prometheus' local storage. Following collection, the dataset undergoes data cleansing to ensure consistency. Subsequently, the data transformation organizes the metrics into a suitable format for predictive modeling. The structured dataset, serving as valuable input for feeding models, is stored in the file storage.

3) Model Generation: This component is designed to derive an optimal predictive model. It initially splits the structured dataset into training and testing sets to train and evaluate diverse models. The model with the highest performance on the testing dataset is chosen as the optimal choice. Subsequently, the previously selected model is analyzed for feature importance and refined via hyperparameter tuning to boost its performance, producing an optimal model apt for application in cloud environments.

B. Data Preparation

This subsection delineates the critical steps involved in data preparation: collection, cleansing, and transformation. A well-executed data preparation process ensures the accuracy and dependability of the developed models.

1) Data Collection: Collecting system metrics from the application, we used a custom script to access Prometheus' local storage. The metrics include CPU and memory requests (expressed as percentages), network inbound and outbound rates (measured in GB/s or MB/s), TPS, and average response time (in milliseconds). Additionally, we observed the server's real-time health status, which was determined by scrutinizing the HTTP response codes [27]. If predefined error response codes are detected, the server status is labeled as *unhealthy* at that time; otherwise, it is marked *healthy*. Despite disk storage being infrequently used in stateless applications and not included in the training input, we monitored it to ensure proper stateless behavior. The dataset comprises approximately 3,000 data points, each collected every 5 seconds.

2) Data Cleansing: Upon retrieving data from the storage, we examined each row for missing feature values to ensure data accuracy and completeness. In instances where a missing value is identified, the corresponding row is removed without affecting adjacent rows due to the independence of each data entry.

3) Data Transformation: Following the data cleansing process, we performed a series of transformations to prepare the data in a suitable format for subsequent analysis. In addition, Table I illustrates the adjustments made to different metrics through changes to align them with our analytical requirements. Specifically, CPU and memory requests were converted into ratios. Inbound and outbound bandwidth were standardized to megabytes per second (MB/s). The remaining features, such as TPS and average response time, were retained in their original form as collected. Lastly, server status was converted from original text designations to a numerical format—this conversion to binary encoded *healthy* as 0 and *unhealthy* as 1.

C. Model Comparison

An integral part of our methodology involves comparing the performance of different machine learning models. For

this purpose, we chose five models: Decision Trees, Support Vector Machines, K-Nearest Neighbors, Logistic Regression, and Neural Networks. Each algorithm was configured with the default parameter values provided by the library [28]–[32]. In addition to Neural Networks, we employed two hidden layers for our study. The first layer had six nodes with a ReLU activation function, while the second had one node with sigmoid activation. These algorithms were implemented in Python [33] with the scikit-learn library [34].

To bolster our models' robustness and generalizability, we employed cross-validation, involving multiple training and validation rounds on varied data partitions to obtain averaged scores. Due to the imbalanced nature of the dataset [35], characterized by a higher prevalence of *healthy* statuses, accuracy was deemed an unreliable performance indicator, potentially leading to misleading conclusions. Instead, we evaluated model performance using precision, recall, and F1 Score [36], employing macro averaging [37] to ensure equal weightage to each category. Among the models assessed, the one with the highest macro-averaged F1 Score was deemed the most optimal, showcasing a balanced trade-off between precision and recall as detailed in Equations (1), (2), and (3).

$$\text{Precision}_{\text{macro}} = \frac{\text{Precision}_{\text{healthy}} + \text{Precision}_{\text{unhealthy}}}{2} \quad (1)$$

$$\text{Recall}_{\text{macro}} = \frac{\text{Recall}_{\text{healthy}} + \text{Recall}_{\text{unhealthy}}}{2} \quad (2)$$

$$F1_{\text{macro}} = \frac{2 \times \text{Precision}_{\text{macro}} \times \text{Recall}_{\text{macro}}}{\text{Precision}_{\text{macro}} + \text{Recall}_{\text{macro}}} \quad (3)$$

D. Feature Importance

Our study assessed the importance of each feature by analyzing its coefficient within the model, thereby quantifying the influence of metrics: CPU and memory requests, inbound and outbound bandwidth, TPS, and average response time on the predictive model's output. Our investigation calculated the significance through cross-validation during every training validation iteration.

E. Hyperparameter Tuning

Subsequently, we adjusted the model's hyperparameters to evaluate the efficacy of our tuning and the associated improvement in predictive capability. The method systematically tested a range of parameter values, targeting the combination yielding the highest macro-averaged F1 Score for the model.

In this study, we examined several hyperparameters of Logistic Regression, which was identified as the top performer in Section IV. These included *Regularization* (*Penalty* and *C*), *Solver*, *Maximum iterations*, *Class weight*, *Fit intercept*, and *Multiclass*, all of which influence model performance. Parameter ranges under consideration are detailed in Table II.

IV. Result & Discussion

This section presents our empirical findings and their implications for machine learning models in server health prediction. Additionally, it delves into the effects of feature scaling and system specification bias on predictive accuracy.

TABLE II
HYPERPARAMETER RANGES IN LOGISTIC REGRESSION TUNING

Parameter	Range of Values
Penalty	l1, l2, elasticnet
C	0.001, 0.01, 0.1, 1, 10, 100, 1000
Solver	lbfgs, liblinear, newton-cg, newton-cholesky, sag, saga
Maximum Iterations	1, 10, 100, 1000, 10000
Class Weight	none, balanced
Fit Intercept	true, false
Multiclass	auto, ovr, multinomial

A. Result

This subsection delves into the experimental results, dividing them into three primary categories: model comparison, feature importance, and hyperparameter tuning.

1) Models Comparison: Table III compares the performance of various base models using macro-averaged precision, recall, and F1 Score. It illustrates that Logistic Regression outperforms others, achieving the highest average scores, indicating its balanced effectiveness in identifying positive instances while minimizing false positives, reflected in high recall and precision. This balance underscores Logistic Regression's aptitude for reliable server health status prediction, making it a robust choice for this task.

2) Feature Importance: In the subsequent analysis, the focus pivots toward feature importance as determined by our refined Logistic Regression model, identified as the optimal base model in the previous section. Fig. 2 displays the coefficients for each feature, with their magnitudes underscoring the relative importance in the prediction process.

Fig. 2 provides the coefficients of each feature in the pre-

TABLE III
PERFORMANCE OF BASE MODELS IN SERVER HEALTH PREDICTION

	Macro-avg Precision	Macro-avg Recall	Macro-avg F1 Score
Decision Trees	0.853	0.862	0.851
Support Vector Machine	0.937	0.913	0.924
K-Nearest Neighbors	0.937	0.909	0.922
Logistic Regression	0.950	0.969	0.951
Neural Networks	0.719	0.750	0.725

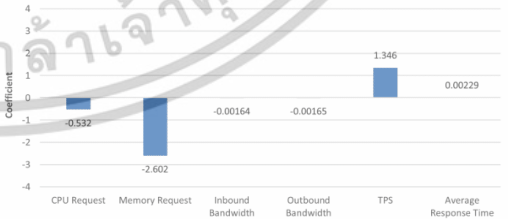


Fig. 2. This figure shows the coefficient of system metrics in server health prediction.

dictive model, elucidating their impacts. Negative coefficients for CPU and memory requests, and inbound and outbound bandwidth, imply an inverse relationship with the *unhealthy* class likelihood. Conversely, positive coefficients for TPS and average response time signify their proportional relationship with *unhealthy* class probability. This insight into feature contributions informed our decision to engage in hyperparameter tuning using all collected features.

3) Hyperparameter Tuning: Logistic Regression excelled among the base models, leading us to optimize its parameters for better predictive performance. Table IV encapsulates the optimization results, delineating our model's optimal parameter settings. Concurrently, Table V highlights the effects of hyperparameter tuning on model performance. It compares the top three base models from Table III against the tuned Logistic Regression model, demonstrating the latter's improved performance relative to its initial version and other base models.

B. Performance Impact of Feature Scaling

Moreover, we investigated the impact of normalizing features with large value ranges—such as inbound and outbound bandwidth, transactions per second (TPS), and average response time—to a normalized range of 0 to 1 to assess improvements in predictive accuracy. Fig. 3 presents model performance between scaled and unscaled variables using the macro-averaged F1 Score. Interestingly, scaling features resulted in decreased performance for most models. Notably, even the best-performing scaled model, SVM, with a score of 0.908, underperformed compared to the third-ranked unscaled model, KNN, which achieved a higher score of 0.922.

Based on the results, we opted to maintain the original feature set rather than the scaled versions to avoid the noted reduction in predictive performance, ensuring maximal accuracy and reliability consistent with our stated objectives.

TABLE IV
OPTIMAL PARAMETERS FOR TUNING LOGISTIC REGRESSION

Parameter	Values
Penalty	l1
C	1
Solver	liblinear
Maximum Iterations	100
Class Weight	none
Fit Intercept	false
Multiclass	auto

TABLE V
PERFORMANCE OF TOP THREE MODELS AND TUNED LR (LOGISTIC REGRESSION) MODEL

	Macro-avg Precision	Macro-avg Recall	Macro-avg F1 Score
K-Nearest Neighbors	0.937	0.909	0.922
Support Vector Machine	0.937	0.913	0.924
Logistic Regression	0.950	0.969	0.951
Tuned LR	**0.984**	**0.973**	**0.977**

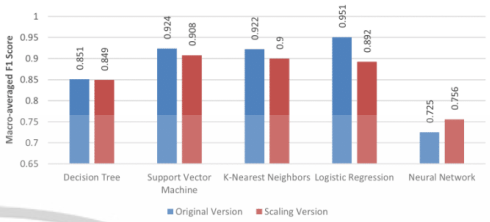


Fig. 3. This figure compares models' performance using original features to their counterparts using scaled features.

C. System Specification Bias

As outlined in Section III: Application Simulation, our methodology employed a server with specific configurations such as CPU, memory, and operating system. It is crucial to recognize that variations in these specifications could influence the server's sufficiency, thereby affecting our model's predictive accuracy and generalizability. Hence, it is advisable to consider tuning the model to ensure optimized performance tailored to the specific setup of various servers.

V. CONCLUSION AND FUTURE WORK

Our research was dedicated to predicting the current stateless application health based on system metrics. These metrics include Central Processing Unit (CPU) and memory requests, inbound and outbound bandwidth, Transactions per second (TPS), and average response time. We employed a diverse set of machine learning models to accomplish our objective. According to our findings, the Logistic Regression model with hyperparameter tuning provided the most effective prediction performance: a precision of 98.4%, a recall of 97.3%, and an F1 Score of 97.7% on macro averaging.

Our research offers a method to evaluate the health of stateless server applications based on current system metrics without understanding their intricate relationships. By leveraging the model, organizations can enhance their server monitoring frameworks with real-time health assessments, enabling timely interventions and efficient maintenance, which, in turn, mitigate potential issues and foster improved operational reliability in cloud environments. Furthermore, the system's predictive capabilities could significantly aid cloud administrators in proactive resource management. They can utilize the model to simulate various scenarios, determining the likelihood of server failure without the time-consuming and technically demanding process of manual testing.

In future work, we intend to expand our research to develop a model to predict future system metrics from historical patterns. This enhancement is expected to bolster proactive system maintenance strategies. By integrating predictive insights into our current classification framework, we aim to achieve a more nuanced, forward-looking server health evaluation, effectively creating an advanced early warning system to tackle system vulnerabilities proactively.

REFERENCES

[1] Islam, S., Venugopal, S., Liu, A.: Evaluating the impact of fine-scale burstiness on cloud elasticity. In: Proc. ACM Symp. Cloud Comput., pp. 250–261 (2015).

[2] Red Hat: Stateful vs stateless (2020). Available: https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless. Last accessed: 2023/06/14.

[3] Microservices.io, https://microservices.io, last accessed 2023/06/14.

[4] EC2 Tier List, https://aws.amazon.com/ec2/instance-types, last accessed 2023/07/04.

[5] Gupta, L.: Statelessness in REST APIs. REST API Tutorial (2022). Available: https://restfulapi.net/statelessness. Last accessed: 2023/07/06.

[6] Wilson, J.: Stateful and Stateless Horizontal Scaling for Cloud Environments. Rose Hosting (2021). Available: https://www.rosehosting.com/blog/stateful-and-stateless-horizontal-scaling-for-cloud-environments. Last accessed: 2023/07/06.

[7] EC2 Auto Scaling, https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html, last accessed 2023/07/04.

[8] Autoscaling groups of instances. Google Cloud (2023). Available: https://cloud.google.com/compute/docs/autoscaler. Last accessed: 2023/06/14.

[9] Asmawi, T.N., Ismail, A., Shen, J.: Cloud failure prediction based on traditional machine learning and deep learning. J Cloud Comp 11, 47 (2022). doi: 10.1186/s13677-022-00327-0.

[10] Jassas, M.S., Mahmoud, Q.H.: A Failure Prediction Model for Large Scale Cloud Applications using Deep Learning. In: 2021 IEEE International Systems Conference (SysCon), pp. 1–8. IEEE, Vancouver (2021). doi: 10.1109/SysCon48628.2021.9447141.

[11] Liu, X., et al.: Smart Server Crash Prediction in Cloud Service Data Center. In: 2020 19th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), pp. 1350–1355. IEEE, Orlando (2020). doi: 10.1109/ITherm45881.2020.9190321.

[12] Hioual, O., Hioual, O., Mounine, H.S., Ouail, A.: A New Method for Predicting Failures in Cloud Computing. In: Advanced Intelligent Systems for Sustainable Development (AI2SD'2020). Springer, February 2022. doi: 10.1007/978-3-030-90639-9_49.

[13] Islam, T., Manivannan, D.: Predicting Application Failure in Cloud: A Machine Learning Approach. In: 2017 IEEE International Conference on Cognitive Computing (ICCC), pp. 24–31. IEEE, Honolulu (2017). doi: 10.1109/IEEE.ICCC.2017.11.

[14] Wilkes, J.: More Google cluster data. Google Research Blog (2011). Available: https://ai.googleblog.com/2011/11/more-google-cluster-data.html. Last accessed: 2023/06/12.

[15] Cloudflare: Virtual Machine. Available: https://www.cloudflare.com/learning/cloud/what-is-a-virtual-machine. Last accessed: 2023/07/04.

[16] Shin, T.: Understanding Feature Importance and How to Implement it in Python. Towards Data Science (2021). Available: https://towardsdatascience.com/understanding-feature-importance-and-how-to-implement-it-in-python-ff0287b20285. Last accessed: 2023/07/04.

[17] Quinlan, J.R.: Induction of Decision Trees. Machine Learning 1, 81-106 (1986). doi: 10.1007/BF00116251.

[18] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20, 273–297 (1995). doi: 10.1007/BF00994018.

[19] Raj, A.: Everything About Support Vector Classification - Above and Beyond. Towards Data Science (2022). Available: https://towardsdatascience.com/everything-about-svm-classification-above-and-beyond-cc665bfd993e. Last accessed: 2023/07/04.

[20] Cover, T., Hart, P.: Nearest neighbor pattern classification. In: IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21–27 (1967). doi: 10.1109/TIT.1967.1053964.

[21] Hosmer Jr., D.W., Lemeshow, S., Sturdivant, R.X.: Applied Logistic Regression. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc. (2013). doi: 10.1002/9781118548387.

[22] Brooks-Bartlett, J.: Probability concepts explained: Maximum likelihood estimation. Towards Data Science (2018). Available: https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1. Last accessed: 2023/07/04.

[23] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature, vol. 521, pp. 436–444 (2015). doi: 10.1038/nature14539.

[24] DigitalOcean, https://www.digitalocean.com, last accessed 2023/07/04.

[25] Apache JMeter, https://jmeter.apache.org, last accessed 2023/06/15.

[26] Prometheus, https://prometheus.io, last accessed 2023/06/15.

[27] MDN Web Docs: HTTP response status codes, https://developer.mozilla.org/en-US/docs/Web/HTTP/Status, last accessed 2023/07/04.

[28] Scikit Learn: Decision Trees, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, last accessed 2023/06/24.

[29] Scikit Learn: KNeighborsClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html, last accessed 2023/06/24.

[30] Scikit Learn: SVC, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html, last accessed 2023/06/24.

[31] Scikit Learn: Logistic Regression, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, last accessed 2023/06/24.

[32] Scikit Learn: Neural network models, https://scikit-learn.org/stable/modules/neural_networks_supervised.html, last accessed 2023/06/24.

[33] Python, https://www.python.org, last accessed 2023/06/15.

[34] Pedregosa, et al.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825-2830 (2011).

[35] Brownlee J.: A Gentle Introduction to Imbalanced Classification. Machine Learning Mastery (2020). Available: https://machinelearningmastery.com/what-is-imbalanced-classification. Last accessed: 2023/06/15.

[36] Kanstrén, T.: A Look at Precision, Recall, and F1-Score. Towards Data Science (2020). Available: https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec. Last accessed: 2023/08/10.

[37] Tariq A.: What is the difference between micro and macro averaging? Educative (2023). Available: https://www.educative.io/answers/what-is-the-difference-between-micro-and-macro-averaging. Last accessed: 2023/06/15.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 April 2024, accepted 21 May 2024, date of publication 29 May 2024, date of current version 10 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3406670

RESEARCH ARTICLE

Stateless System Performance Prediction and Health Assessment in Cloud Environments: Introducing cSysGuard, an Ensemble Modeling Approach

NUTT CHAIRATANA [1] AND RATHACHAI CHAWUTHAI [2]
[1]Department of Robotics and AI Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
[2]Department of Computer Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
Corresponding author: Rathachai Chawuthai (rathachai.ch@kmitl.ac.th)

ABSTRACT Stateless cloud computing presents remarkable scalability and cost-effectiveness by offering dynamically adjustable resources tailored to fluctuating demands, eliminating the constraints of stateful architectures. However, the challenges presented by dynamic workload are substantial in the context of system health monitoring, frequently leading to service interruptions owing to insufficient resources. It underscores the need for the development of more efficient monitoring systems. Our study introduces cSysGuard, a novel framework designed to enhance monitoring capabilities within cloud environments. The methodology employs an ensemble regression model with a stacking strategy to forecast dynamic performance metrics. The algorithm also leverages a classification model to assess the system's health based on forecasted metrics, effectively identifying potential failures in the future. Under the configuration utilized, our evaluations demonstrated increased predictive performance with cSysGuard in forecasting various metrics compared to traditional models. The results showed an improvement of up to a remarkable 2.28-fold increase, varying significantly based on the specific metric under consideration. In addition, the effectiveness of health assessment was achieved through Decision Trees with hyperparameter tuning, resulting in a macro-averaged F1 score of 89.79%. This research contributes to both the theoretical and practical aspects of server monitoring, presenting a solution that assesses system performance metrics and health to tackle dynamic challenges in cloud infrastructure.

INDEX TERMS Cloud computing, stacking ensemble models, machine learning, non-functional testing, predictive maintenance, resource allocation, proactive system metrics, stateless application.

I. INTRODUCTION

Cloud computing, renowned for its scalability [1] and flexibility [2], has revolutionized the management of system resources. Within its broad spectrum, stateless cloud computing plays a crucial role. This paradigm facilitates the flexible allocation of resources without retaining session information [3], significantly aiding in instance scaling. In such cases, it significantly enhances system reliability and adaptability to fluctuating demands while maintaining optimal system utilization. As easily integrated with various architectures, stateless computing has become the preferred framework for application hosting, directly catering to the dynamic demands of the digital environment.

Nonetheless, the unpredictable nature of cloud computing [4] often complicates effective resource allocation, carrying the risk of unexpected system failures that violate service-level agreements or SLAs [5]. Even though cloud providers offer auto-scaling mechanisms [6] for dynamic resource adjustment, they often rely on predefined rules or simple threshold-based techniques [7], [8]. While certain providers also provide predictive methods [9] for auto-scaling, the

The associate editor coordinating the review of this manuscript and approving it for publication was Liangxiu Han.

systems are typically pattern-based [10]. These strategies often struggle with scaling within a reasonable timeframe during irregular workload patterns, such as traffic spikes, leading to inefficient resource management. A more robust framework is required to handle these challenges in such cases.

Furthermore, existing monitoring frameworks primarily focus on real-time monitoring data but rarely provide continuous analysis of system health based on current captured data. This limitation arises because the interplay between resource metrics and their sufficiency assessment—determining whether they are adequate or inadequate—is intricate. It is often characterized by nonlinear dependencies from various factors, such as application characteristics and system configuration. Therefore, relying solely on individual metrics to assess resource allocation adequacy effectively can be challenging.

To address these challenges, we require a sophisticated tool that accurately forecasts key metrics and assesses system health using these indicators in dynamic cloud environments. In response, we have developed cSysGuard, denoted as "Cloud System Guard," a comprehensive ensemble predictor framework designed to handle unpredictable nature effectively. It builds upon our previous research [11], which primarily focused on real-time system-health analysis. With this advancement, cSysGuard comprises two main components: a system performance metric forecasting module and a health assessment feature. It predicts performance metrics and then promptly uses these forecasts to assess potential system failures. Unlike traditional real-time monitoring tools [51], [52], cSysGuard emphasizes machine learning-based prediction with customizable internal architecture. This flexibility allows users to tailor the configuration to their needs, maximizing predictive accuracy and robustness in rapidly changing cloud environments.

This manuscript elucidates our research on stateless applications, introducing cSysGuard, an ensemble machine learning framework devised for precise system metrics forecasting and health status diagnosis. The narrative begins with a thorough literature review (Section II), establishing a foundational understanding and drawing parallels with similar endeavors in the field. We then transition to a comprehensive overview of cSysGuard's methodology (Section III), detailing the high-level workings, component functionalities, and data preprocessing strategies. The discussion elaborates on system metrics forecasting (Section IV), highlighting the regression analysis framework, input-output formats, and the model's evaluation and tuning processes. Subsequently, we explore the system health assessment phase (Section V), focusing on classification analysis, predictor structure, and performance optimization techniques. The paper culminates in the Results and Discussion sections (Section VI), presenting a critical evaluation of cSysGuard's predictive capabilities, followed by a conclusion (Section VII) that encapsulates the essential findings and suggests avenues for future research.

II. LITERATURE REVIEW

This section provides a comprehensive synthesis of cloud computing with machine-learning backgrounds, insights from prior studies, and the techniques utilized in our research.

A. ADAPTATION: CLOUD WITH MACHINE LEARNING

Although cloud applications introduce innovative resource management strategies, they also present challenges in monitoring resources. Its variable nature complicates the effective tracking of fluctuating system metrics. Consequently, resource utilization can result in inefficiency and diminished system performance. It underscores the need for more flexible monitoring methods to precisely assess and adapt to the ever-changing requirements of cloud environments.

Therefore, there is growing interest in leveraging machine learning techniques to address these challenges [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. Machine learning-driven predictive analytics, widely used in many fields [29], can analyze complex data patterns to identity potential anomalies. This proactive approach enhances system reliability and resource allocation, ensuring scalable and responsive services. Integrating machine learning with cloud computing represents a significant advancement in improving system resource management.

B. RELATED WORK

Notably, many methodologies frequently favor singular predictive models [12], [13], [14], [15], [16], [17], [18], using universal models irrespective of specific system nuances, a practice often known as the "one-size-fits-all" approach. However, as emphasized by Kim et al. [22], relying on a single predictor model does not adequately address the cloud workload dynamics and short-term volatility. Singh et al. [19] proposed an alternative solution using a Support Vector Machine to categorize workloads into broad categories like "very low," "low," "medium," and "high." Although the method offers a simplified overview of the status, the reliance on categorical variables is insufficient for delivering precise values in applications that require detailed nuances. Gao et al. [20] and Caron et al. [21] proposed forecasting methods based on historical pattern data. Due to their heavy reliance on historical trends, these methods are limited in rapidly evolving environments, particularly when encountering novel patterns that emerge without precedent. In contrast, cSysGuard employs an ensemble machine-learning approach that leverages the unique strengths of various models to achieve robust predictive capabilities. This strategy effectively adapts to the rapidly evolving cloud environments.

Several studies have proposed ensemble model approaches that incorporate numerous predictors. Kim et al. [22] achieved balanced predictions by employing multiple models using a weighted averaging strategy. This method maximizes the strength of each model, calibrated by their predictive

FIGURE 1. RMSE-based performance of meta models. This figure illustrates the fluctuating performances of different meta-models for CPU requests (unit: percent) over 20 minutes in a dynamic environment.

performance. Nonetheless, as Section VI-C indicates, relying solely on weighted averaging yields a lower accuracy than cSysGuard, where high predictive accuracy is essential for anticipating potential failures. Mehmood et al. [23] proposed a stacking ensemble model using Decision Trees as a meta-model to finalize the predictions from the base models. Our findings indicate that a single meta-model may not always provide robust performance, especially in unpredictable environments. Fig. 1 illustrates the varying performances of meta-models for Central Processing Unit (CPU) requests in a dynamic cloud environment over time. The data indicates that the best-performing meta-model changes periodically, presenting a challenge to select the best meta-model for the specific usage. Rather than building upon a single meta-model or weight averaging, cSysGuard employs a multi-layered stacking ensemble approach with a weighted averaging strategy, effectively amalgamating the contributions of all models in the output generation.

On the other hand, deep learning techniques have been extensively utilized in several studies [24], [25], [26], [27], [28]. Although deep learning techniques are potent instruments for predictive analysis, they inevitably require substantial computational resources, even when applied to simpler metric patterns. Meanwhile, cSysGuard offers a balanced strategy for computational resource utilization and predictive accuracy by judiciously selecting well-suited models with specific data characteristics. Even the best accuracy is considered without regard to computational complexity; cSysGuard still performs better than standalone deep learning models, owing to its incorporation of a diverse array of predictive models.

Our prior research focused on system health assessments. Although the inputs remain consistent with our current work, the approach in this study diverges by employing forecasted values instead of actual historical data. This methodology integrates our regression analytics to predict these metrics and feeds the results into our classification health assessment process. This innovative approach enhances our capability to identify potential future system failures by leveraging insights to inform proactive measures for resource management.

C. TECHNIQUE

This subsection provides a comprehensive overview of the essential techniques and a foundation for understanding the advancements in our study area.

1) LOW-PASS BUTTERWORTH FILTER

The low-pass Butterworth filter, a key concept in signal processing [30], attenuates high-frequency noise while maintaining signal integrity. Characterized by its maximally flat magnitude response and ripple-free passband, the filter ensures a smooth frequency response up to the cut-off frequency, making it ideal for applications requiring undistorted outputs. Its transfer function, defined by the filter order, influences the steepness of the roll-off at the cut-off point [31]. Higher-order filters offer a faster passband-to-stopband transition but increase the complexity and risk of phase distortion. In the data analysis, the filter actively smooths out noise and data variations, bolstering the reliability of the analytical results.

2) SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE

The synthetic minority over-sampling technique (SMOTE) addresses imbalanced data by generating synthetic samples for the minority class, thereby improving the data distribution for machine learning applications. It creates new instances through interpolation between minority samples and their neighbors [32], enhancing diversity and reducing overfitting risks. However, its effectiveness depends on careful parameter tuning, including the number of neighbors and the amount of synthetic data generated. The versatility of SMOTE makes it applicable across various domains, from fraud detection to medical diagnoses, where it significantly contributes to more equitable and accurate predictive modeling.

3) ENSEMBLE MODEL: STACKING

The stacking approach in ensemble models features a multi-layered structure, each recognized as a Level-N layer, for enhanced prediction. Initially, the base models process the data independently, offering diverse analytical perspectives. Higher-level models then collectively analyze their outputs to enhance predictive accuracy. Each subsequent level integrates and improves upon the previous ones [33], reducing biases and variances, thus enhancing the overall predictive accuracy. Fig. 2 visually represents the multi-layer stacking approach, presenting the flow from Level-0 to Level-N layers.

4) REGRESSION ANALYTICS

The Autoregressive Integrated Moving Average (ARIMA) model is essential for time-series analysis, effectively capturing temporality with autoregressive features and moving averages for non-seasonal forecasting [34]. The Seasonal Autoregressive Integrated Moving Average (SARIMA) builds on ARIMA by incorporating seasonality, making it adept at predicting seasonal fluctuations in time series data.

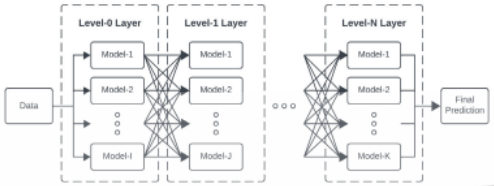


FIGURE 2. Multi-layer stacking in ensemble modeling. This figure depicts the layered architecture of our ensemble model, from initial Level-0 models to advanced aggregation layers, highlighting the sequential prediction and refinement process.

Exponential Smoothing (ETS) is a time-honored technique for forecasting that assigns exponentially decreasing weights over time. It captures trends and seasonalities in time-series data [35], and its simplicity makes it particularly effective for quick and short-term forecasting.

Linear Regression is a fundamental model in statistical analysis [36] and is ideal for predicting a continuous dependent variable using one or more independent variables. It assumes a linear relationship between the variables. A simple LR involves a single independent variable, whereas multiple LRs extend to multiple variables. This model is particularly effective in scenarios where a linear correlation exists, with coefficients indicating the influence of each variable.

Random Forest creates multiple decision trees to improve the prediction accuracy and robustness. It leverages bagging and feature randomness in tree construction [37], reducing overfitting and enhancing diversity. This method effectively handles complex datasets, offering a reliable performance in various applications.

Feedforward Neural Networks feature a straightforward input, hidden, and output layer structure [38]. Using sequential neuron connections, they excel in complex pattern recognition and prediction. Highly versatile and efficient, they are ideal for handling large datasets. Their widespread use in applications such as speech recognition in predictive analytics stems from their proficient nonlinear modeling of intricate relationships between inputs and outputs.

Convolutional Neural Networks (CNN), renowned for their efficacy in image processing, also excel in time-series and regression tasks owing to their ability to prioritize patterns through convolutional layers [39]. It enables practical applications in audio processing and natural language tasks, extending their use beyond visual data analysis.

Temporal Convolutional Networks (TCN) combine the strengths of CNN with the temporal sensitivity suited for sequence modeling [40]. They effectively handle long input sequences in complex time series forecasting tasks.

Recurrent Neural Networks (RNN) are fundamental models for sequential data analysis. It has the unique ability to retain information across sequences [41], making it essential for text processing, speech recognition, and time-series analysis. Nevertheless, RNN encounters challenges with long-term dependencies, leading to the development of advanced versions, such as Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM). The GRU incorporates a simplified gating mechanism to address the vanishing gradient problem [42], balancing computational efficiency with the ability to learn long-term dependencies. This advantage makes the GRU suitable for complex tasks such as language modeling and nuanced time-series analysis. Alternatively, LSTM selectively retains or forgets information [43], making it practical for extended sequences. Its capability to handle long-term dependencies makes LSTM ideal for challenging tasks, such as predictive text generation, advanced time series forecasting, and language translation.

5) CLASSIFICATION ANALYTICS

Decision Trees are known to simplify complex decision-making. They created an intuitive, tree-like structure by segmenting datasets into branches based on feature values [44]. This method divides data into smaller, more manageable subsets, effectively handling nonlinear relationships among variables and facilitating hierarchical decision processes.

A Support Vector Machine (SVM) is used in classification and regression tasks. It offers robust performance in high-dimensional spaces owing to its margin maximization principle, which forms a hyperplane for data classification [45].

K-Nearest Neighbors (KNN) is an instance-based algorithm that classifies data by analyzing the majority class among its "K" nearest neighbors, relying on a similarity principle [46]. The algorithm's accuracy is sensitive to the chosen number of neighbors and the specific distance metric applied, thereby impacting its adaptability to various data.

Logistic Regression, essential for binary classification in machine learning [47], uses a logistic function to convert predictor variables into probabilities, thereby accommodating nonlinear variable relationships. In various fields, like medical diagnosis and spam detection, employing maximum likelihood estimation for coefficient calculation is pivotal.

Neural Networks, inspired by the structure of biological neurons, are composed of complex interconnected layers of artificial neurons or nodes [48]. It begins with an input layer, includes one or more hidden layers, and ends with an output layer. Each node simulates a biological neuron, calculates a weighted sum of inputs, and applies a nonlinear transformation commonly termed the activation function.

6) EVALUATION METHOD

The Root Mean Square Error (RMSE) is a critical metric for assessing regression tasks, quantifying the square root of the mean of the squared discrepancies between the predicted and actual values. By squaring the errors before averaging, the RMSE heavily penalizes the larger discrepancies, making it particularly sensitive to significant prediction errors. This characteristic enhances its utility in offering a precise measure of model accuracy. The formula is detailed in (1)

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y}_k)^2} \tag{1}$$

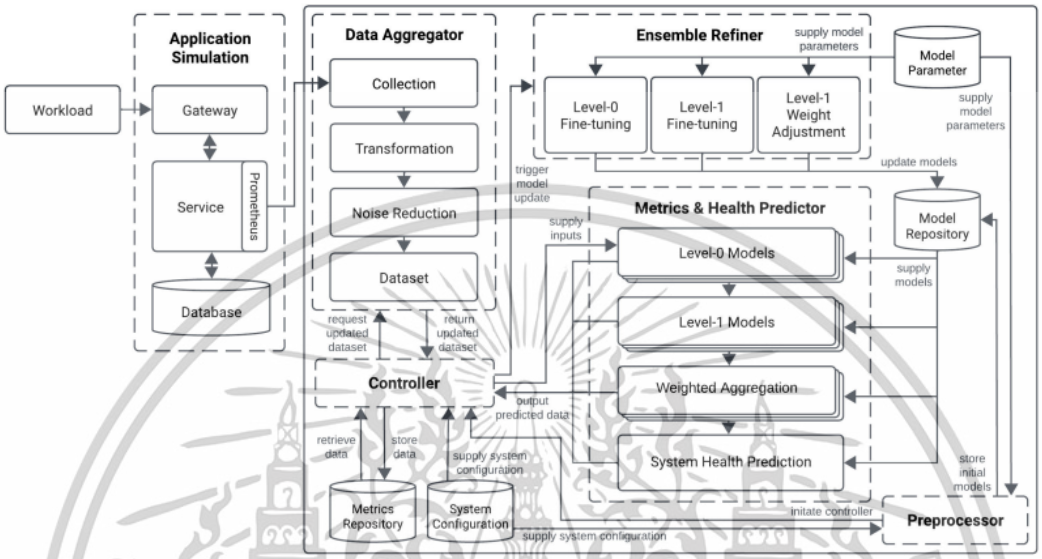


FIGURE 3. cSysGuard architecture. This figure demonstrates the process of system preprocessing, data collection, performance metrics forecasting, health assessment, and continuous model refinement.

where y_k represents the actual value, $\hat{y}_k$ denotes the predicted value, and n signifies the number of observations. A lower RMSE indicates better accuracy compared to a higher one.

The F1 score is a key metric for evaluating classification tasks. It is computed by harmonically averaging precision, which is the ratio of true positives to all positive predictions, indicating the accuracy of positive classifications, and recall, the ratio of true positives to all actual positives, reflecting the ability to identify all relevant instances. These calculations are elaborated in (2), (3), and (4) for class n.

$$\text{Precision}_n = \frac{TP_n}{TP_n + FP_n} \tag{2}$$

$$\text{Recall}_n = \frac{TP_n}{TP_n + FN_n} \tag{3}$$

$$\text{F1-Score}_n = 2 \times \frac{\text{Precision}_n \times \text{Recall}_n}{\text{Precision}_n + \text{Recall}_n} \tag{4}$$

In addition to these equations, TP represents true positives, FP signifies false positives, and FN indicates false negatives. An F1 score of 1 signifies perfect precision and recall, indicating that all predictions are accurate and complete. Conversely, a score of 0 represents the lowest accuracy, where the model fails to identify any true positive values.

III. METHODOLOGY OVERVIEW

This section provides a detailed overview of the methodology used in our study, focusing on the design and data handling of cSysGuard.

A. CSYSGUARD ARCHITECTURE

Fig. 3 comprehensively depicts the architecture of cSysGuard, comprising six core components: application simulation, data aggregator, controller, metrics and health predictor, ensemble refiner, and preprocessor. Each is integral to the prediction process. The following sections briefly describe the system workflow and each element, elucidating their functions and collaborative interactions.

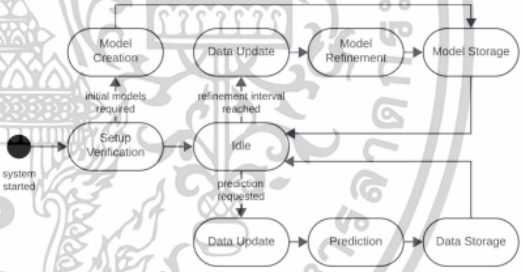


FIGURE 4. cSysGuard execution workflow. The figure illustrates cSysGuard's operational sequence, which mainly encompasses the stages of preprocessing, data prediction, and model refinement.

1) EXECUTION WORKFLOW

Fig. 4 reveals the execution workflow of cSysGuard. Initially, cSysGuard determines whether the configuration necessitates the creation of models. In such cases, it initiates and stores these models for future analysis; otherwise, it bypasses the preprocessing step and proceeds directly to the primary process using the pre-existing models stored in the repository. In the main phase, the system remains idle until a task emerges. Upon reaching the prediction intervals, the algorithm collects data from the target application to access the most recent inputs and then activates the

predictors to generate and store the forecast. In addition to preserving model performance, the refinement process is activated at predefined intervals to utilize the latest dataset to align models with current data trends. Post-refinement, updated models are stored. This cycle of prediction and refinement recurs over time, ensuring cSysGuard's processes are perpetually optimized and up-to-date.

2) APPLICATION SIMULATION

This component emulates an application cloud environment, providing a means to acquire the dataset. The study leveraged Digital Ocean's cloud service [49], deploying three Linux virtual machines [50], each with 1vCPU, 1GB of memory, and a 10GB disk. The API gateway is responsible for routing requests to the stateless application service, which utilizes a database for data storage. Our target service incorporates a monitoring tool called Prometheus [51] to collect and store metrics over time methodically. In addition, the simulator employs a custom script to simulate dynamic workloads, manipulating the intensity of various metrics in unique combinations for each request. For instance, one scenario may involve high CPU and memory usage with low usage in other metrics, whereas another could stress high bandwidth usage while keeping the rest low. This approach enables the model to evaluate the system's performance under fluctuating conditions, focusing on randomness and swift changes. These assumptions mirror real-world cloud environments, enhancing the generalizability of cSysGuard by ensuring it can adapt to unpredictable and rapidly changing workloads.

3) DATA AGGREGATOR

This component is responsible for updating and structuring the dataset for modeling and analysis. Initially, the process aggregates the system data from the application service and then transforms the metrics into a suitable format for predictive modeling. To enhance the accuracy of the forecasts, it also performs a noise reduction process to minimize data inconsistencies. Finally, the component forwards the processed dataset to the controller for further processing.

4) CONTROLLER

This component is crucial for orchestrating cSysGuard's logic flow. It begins by fetching the essential setting parameters, such as the model refinement and prediction schedules, from the configuration repository. The controller initiates the models to generate system metrics and health forecasts. Afterward, it archives the predictions in the metrics repository. Beyond these tasks, it manages the model maintenance by periodically signaling the refiner to tune models as per the schedule. Additionally, it consistently acquires the latest data from the aggregator for system predictions and refinements. This adept coordination by the controller is vital to cSysGuard's seamless operation and overall dependability and efficiency.

5) METRICS AND HEALTH PREDICTOR

This component mainly oversees the prediction process, including forecasting system performance metrics and health assessments. Once activated by the controller, the prediction process is initiated with the most recent dataset as the input. Represented as an ensemble regression model, the first three layers—Level 0, Level 1, and Weight Aggregation—mainly focus on forecasting performance metrics, as detailed in Section IV. Besides, we incorporate several sets of predictors within the forecasting layers, with each set specializing in specific metrics. Upon completion, it consolidates and forwards the results to the final layer for system health assessment, requiring inputs from all metric sets for a thorough evaluation, as detailed in Section V. Moreover, each layer sends its predictions to the controller for storage, serving as valuable data for subsequent model evaluation and refinement. This component ensures precise and timely predictions through a comprehensive and systematic ensemble architecture.

6) ENSEMBLE REFINER

This component maintains the regression model's proficiency in identifying the trends and data characteristics. Upon activation by the controller, the refiner fine-tunes the Level-0 and Level-1 models by utilizing the latest dataset. Similarly, it adjusts the Level-1 weights based on the range of historical predictions, dynamically aligning the models' strengths with their predictive efficacy. Once completed, the system promptly updates the repository with the newly refined model and weights. Through this iterative refinement, the component enhances the adaptability and precision of the models, ensuring they remain effective in a dynamic environment.

7) PREPROCESSOR

This component is essential for setting up the models cSysGuard requires to initiate the primary process. The preprocessor commences by retrieving the list of models from the system configuration. Then, it actively compiles the models with predefined parameters and training datasets. Upon completion, the calibrated models are stored in the repository. However, the preprocessing step is optional if the repository already contains the models.

B. DATA PREPROCESSING

In the data collection, we implemented a custom script to access Prometheus' local storage to receive resource utilization and performance indicators. The dataset [57] comprises approximately 8,000 data points, each accumulated every 5 seconds. The metrics include CPU and memory requests (expressed as percentages), network inbound and outbound rates (measured in gigabytes per second, GB/s, or megabytes per second, MB/s), transactions per second (TPS), and response time (in seconds, s, or milliseconds, ms), as illustrated in Fig. 5. Moreover, we collected the system's

TABLE 1. Transformed system performance metrics dataset with server health in 40 seconds.

CPU Request[1]	Memory Request[1]	Inbound Bandwidth (MB/s)	Outbound Bandwidth (MB/s)	TPS	Response Time (ms)	System Health (binary)
0.12	0.505	3160	3430	3.6	2910	1
0.124	0.504	3590	3120	2.2	1150	0
0.186	0.542	5920	5290	2.2	2560	1
0.126	0.505	3530	3900	4	4060	1
0.184	0.526	4890	4310	2	2320	0
0.150	0.534	4610	4090	3	1880	0
0.174	0.533	6250	5490	2.4	1370	0
0.164	0.535	3490	3150	2.8	1950	0

[1] Values are normalized to a range from 0 to 1.

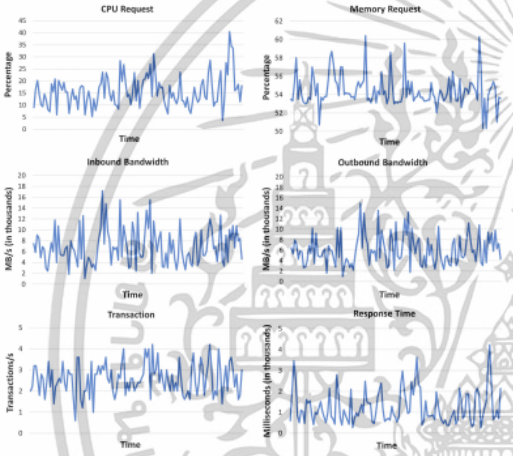


FIGURE 5. System performance metrics across different parameters. This figure displays a series of graphs that showcase CPU and memory requests, inbound and outbound bandwidth, transaction rates, and average response times, all measured over a 10-minute period.

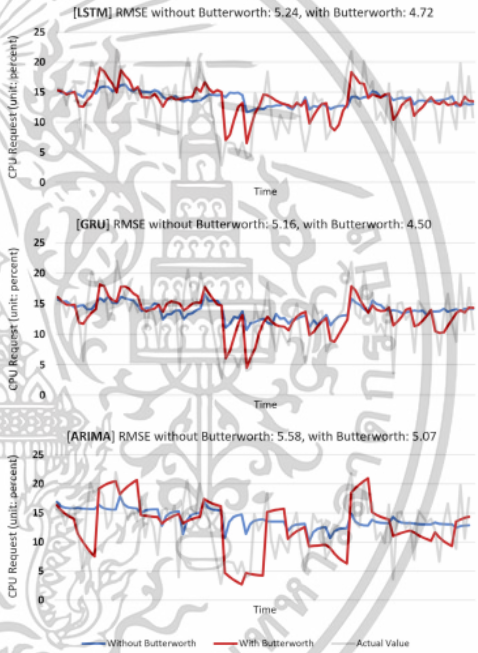


FIGURE 6. Comparative predictions and RMSE with/without low-pass butterworth filter. This figure displays the 10-minute predictions for LSTM, GRU, and ARIMA models in CPU requests, illustrating the comparison between scenarios with filtered and unfiltered data.

real-time health status, which was determined by scrutinizing HTTP response codes [53]. When the script detects predefined error response codes, it classifies the system health as "unhealthy"; in all other cases, it designates the status as "healthy." Although stateless applications rarely leverage disk storage, which is not part of the training input, we still monitor it to ensure its proper stateless behavior.

Subsequently, the system performs a series of transformations to format the data appropriately for subsequent modeling and analysis. Table 1 shows the modification results, outlining how various metrics were adjusted to meet the analytical requirements. Specifically, we normalized CPU and memory requests to a scale from 0 to 1, transformed inbound and outbound bandwidth in megabytes per second (MB/s), and quantified response time in milliseconds (ms). We retained the TPS feature in its original form as collected. The mechanism also transformed system status from the original textual designations into a numerical format, encoding "healthy" as 0 and "unhealthy" as 1, facilitating binary analysis.

Eventually, the system performs a noise reduction process utilizing a low-pass Butterworth filter to minimize data inconsistencies. To demonstrate an enhanced forecasting accuracy, Fig. 6 showcases the impact of noise reduction by contrasting the predictions and RMSE of LSTM, GRU, and ARIMA models using both noise-filtered and raw data inputs with actual values. It highlights the enhancement achieved by applying a filter to reduce noise. The process then yields and stores a well-refined dataset for further analysis.

IV. SYSTEM METRICS FORECASTING

This section describes the detailed process of forecasting performance metrics. This comprehensive examination

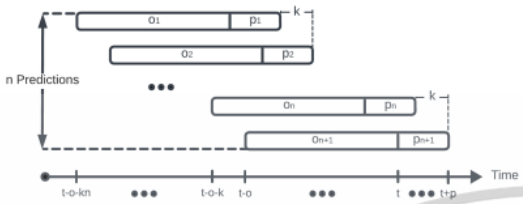


FIGURE 7. Regression prediction sequence. This figure displays the prediction cycle, integrating *o* observations and *p* forecast steps within a set interval *k* over *n* predictive cycles.

covers the intricacies of Level-0 and Level-1 learners, Weight Aggregation, and the approaches used for comparing models and tuning hyperparameters in regression models.

A. LEVEL-0 LEARNER

Level-0 learners comprise base models designed for time-series prediction that utilize datasets aggregated directly from the system for their training. At each predefined interval, these models use the most recent dataset to predict forthcoming values, considering two key elements: the predetermined count of past data points (observation steps) and the quantity of future data points (prediction steps) to be predicted. To illustrate, Fig. 7 visualizes the sequence of time series predictions, showcasing observation steps "o" and prediction steps "p" for each interval "k" across "n" predictions. cSysGuard allows the adjustment of these parameters to meet user-specific requirements. In this study, we configured the settings with 30 observation steps and five forecast steps, aligning the interval with the forecasting steps.

Our setup has operated various models, including the ARIMA, SARIMA, ETS, CNN, TCN, RNN, GRU, and LSTM. Each brings a unique forecasting approach, enabling us to capture various aspects and patterns within the data. In addition to deep learning, we incorporated two hidden layers into the architecture: the first equipped with 64 neurons and the second with 32 neurons, both employing ReLU activation. We used the default settings provided by the Statsmodels [54] and TensorFlow [55] libraries. Despite this, we observed that the library's default parameter values for deep learning led to suboptimal predictions. Therefore, parameter adjustments are required. Table 2 outlines the model-fitting configuration, ensuring the retention of initial learning proficiency of deep learning models. Upon prediction, the models forward the results to the subsequent layer, Level 1.

B. LEVEL-1 LEARNER

Level-1 learners serve as meta-models designed for refining the regression outputs and constitute the second layer in our predictive framework. Their main objective is to improve the forecast precision by leveraging the outcomes from Level-0 models as inputs to produce refined outputs. This integration enables Level-1 learners to fine-tune the initial forecasts, enhancing the overall predictive performance of cSysGuard.

TABLE 2. Parameters influencing performance in deep learning model training.

Adjusted Parameter	Values	Default
Epochs	50	1
Batch Sizes	32	none
Default Parameter	**Values**	
Shuffle	true	
Validation Split	0.0	
Validation Steps	none	
Validation Batch Size	none	
Validation Frequency	1	
Initial Epoch	0	
Steps Per Epoch	none	
Callbacks	none	
Class Weight	none	
Sample Weight	none	

This layer comprises three distinct models: Linear Regression, Random Forest, and Feedforward Neural Networks. Linear Regression provides simplicity and interoperability to effectively capture linear relationships within the data. Random Forest contributes to reducing overfitting and improving generalization. Feedforward Neural Networks excel at capturing complex, nonlinear patterns and interactions in the data. This combination leverages the strengths of each model, ensuring that the stacking approach can effectively generalize across different metrics and system behaviors to enhance the predictive accuracy and robustness of cSysGuard.

For Linear Regression and Random Forest, we employed the standard parameters provided by the Scikit-learn library [56]. For Feedforward Neural Networks, we replicated the deep learning structure and model-fitting parameters of Level-0 models using the TensorFlow library.

To optimally balance cSysGuard's predictive performance and computational complexity, we adopted a selective approach to determine which Level-0 models would be fed into specific Level-1 models for predictions. The selection depends on the impact each model has on the predictive performance for particular metrics, evaluated by the Pearson correlation coefficient, r [58], which is defined as follows:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{5}$$

where x_i represents the predicted metric values from Level-0 models, $\bar{x}$ signifies the average predicted metric across Level-0 models, y_i indicates the actual metric values, and $\bar{y}$ reflects the mean of the actual metrics. Upon score calculation, we sorted the models in descending order based on their correlations to gauge their impact. For instance, Fig. 9 illustrates the Pearson correlations of each base model for predicting CPU requests. We then incrementally incorporated Level-0 models into each Level-1 learner, starting with those having the highest correlation scores, and assessed

FIGURE 8. Incremental performance improvement in level-1 models. This figure illustrates RMSE progression for Level-1 models predicting CPU requests (unit: percent), highlighting the incremental performance enhancements achieved with each addition of a Level-0 model.

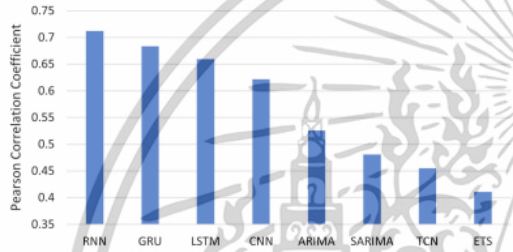


FIGURE 9. Correlation analysis of level-0 models in CPU request prediction. This figure ranks the base models based on the Pearson correlation scores.

FIGURE 10. Consolidated performance analysis of level-1 models. This figure synthesizes the RMSE achievements of Level-1 models in forecasting CPU requests (unit: percent), encapsulating the collective efficacy derived from integrating multiple Level-0 models.

the predictive performance of each Level-1 model using the RMSE. This iterative process continued until all base models were utilized and assessed. For example, Fig. 8 demonstrates the improvement in the performance of Level-1 models for predicting CPU requests by adding each model. Alternatively, Fig. 10 offers a unified visualization of the enhancements across the models, presented in a single comprehensive graph. Our analysis aimed to pinpoint an "elbow point" [59], where incorporating additional models led to minimal gains in the predictive accuracy for the specific metric. This selective approach to model integration significantly conserves computational resources while ensuring high predictive precision, demonstrating a successful trade-off between performance and computational efficiency in intricate forecasting tasks.

C. WEIGHTED AGGREGATION LAYER

The weighted aggregation layer is the final regression stage for predicting forthcoming metrics. To effectively leverage the contributions of each Level-1 model, the system receives predictive model outputs and refines the final prediction using a weight-averaging method [60] by assigning higher weights to more accurate models. In addition to the weight calculation, the layer computes the Performance Vector (PV) of Level-1 learners based on past t refinement iterations. The PV is represented as a $3 \times t$ matrix:

$$PV = \begin{bmatrix} PE_{LR,1} & PE_{LR,2} & \cdots & PE_{LR,t} \\ PE_{RF,1} & PE_{RF,2} & \cdots & PE_{RF,t} \\ PE_{FF,1} & PE_{FF,2} & \cdots & PE_{FF,t} \end{bmatrix} \quad (6)$$

where it consolidates the prediction errors, measured by RMSE, for Linear Regression (LR), Random Forest (RF), and Feedforward Neural Networks (FF) across t refinement iterations. The refinement process enables ongoing updates to the model performance, ensuring sustained accuracy in a rapidly changing cloud environment.

Nevertheless, incorporating the RMSE directly as weights presents a significant challenge, as it may not adequately reflect model correlations. A model with a slightly lower RMSE may not enhance the ensemble diversity if similar to another model. To mitigate this issue, we developed a normalized weighting system derived from the PV. The calculation scales the errors into normalized weights, identified as $NormWeight_{i,j}$, converting them to a range between 0 and 1 for comparability across different models. The weight normalization formula is outlined in (7):

$$NormWeight_{i,j} = \frac{\exp(-\alpha \times PE_{i,j})}{\sum_{k \in \{LR,RF,FF\}} \exp(-\alpha \times PE_{k,j})} \quad (7)$$

where the weights are calculated by inversely relating them to their RMSE, expressed as $\exp(-\alpha \times PE_{k,j})$, ensuring that models with lower errors are more reliable than those with higher errors. In addition, α acts as a sensitivity controller, adjusting the weight assignment based on error magnitude. α facilitates a subtle balance between emphasizing and moderating differences in model errors and acts as a controllable safeguard to prevent integer overflow [61] during exponential system computations. To maximize the weight difference, our

study aimed to maximize each weight value while ensuring that outcomes stayed within an acceptable range as controlled by α. This strategy allows for a more pronounced distinction in model weights, which is particularly beneficial when the models' performances are nearly identical.

To derive the final prediction, we integrated the outputs from our Level-1 learners through a weighted averaging approach. The final prediction for a given time point m, represented by P_m, is determined as (8):

$$P_m = \sum_{k \in \{LR, RF, FF\}} NormWeight_{k,j} \times P_{k,m} \quad (8)$$

where each model's prediction, denoted as $P_{k,m}$, is multiplied by its respective normalized weight, $NormWeight_{k,m}$. This process guarantees that each model's contribution to the final prediction is proportional to its performance. By harmonizing the models' strengths, this approach yields a more precise and reliable ensemble forecast.

D. MODEL COMPARISON

Our approach evaluated cSysGuard's efficacy in forecasting system metrics by comparing its performance with traditional models used in the Level-0 layer. We utilized RMSE to compare and benchmark the performance across different metrics. This method clearly explains the precision and dependability of cSysGuard, emphasizing its capability to manage dynamic environments effectively.

E. HYPERPARAMETER TUNING

After comparing the models, we concentrated on developing a strategy to enhance cSysGuard further. Therefore, we selected one model from each Level-0 and Level-1 layer and investigated their impacts by adjusting the hyperparameters through Bayesian optimization [62]. Subsequently, we evaluated the improvements using the RMSE as our benchmark.

Given the system configuration constraints outlined in Section VI-E, we limited our hyperparameter tuning to particular models and metrics to mainly focus on a methodology optimizing cSysGuard. With an emphasis on enhancing CPU request predictions, the study focused on refining the RNN and Random Forest models, designated as Level-0 and Level-1, respectively. These models were selected not only due to their numerous tunable parameters but also to demonstrate that our hyperparameter tuning methodology is effective across deep learning and traditional machine learning techniques in a stacking architecture.

The hyperparameters covered various factors within specified ranges, as detailed in Table 3, directly affecting model performance. For the RNN, these parameters include data structuring elements (such as observation steps), model architecture (covering learning rate, the number of neurons, and activation functions in both the first and second layers), and model training configurations (incorporating epochs and batch size). For the Random Forest, considered parameters entail the number of trees (estimators), tree depth (max depth), minimum samples for a split (min sample split), minimum samples per leaf (min sample leaf), maximum number of features considered for splitting a node (max features), maximum number of leaf nodes (max-leaf nodes), minimum impurity decrease for a split (min impurity decrease), whether bootstrap samples are used (bootstrap), and the function to measure the quality of a split (criterion).

TABLE 3. Range of hyperparameters for recurrent neural networks (RNN) and random forest (RF) tuning in CPU request analysis.

Range of Hyperparameters for RNN (Level-0)	
Parameter	**Range of Values**
Observation Steps	5 to 100
Learning Rate	0.001 to 0.1
Number of Neurons in Layer 1	20 to 80
Number of Neurons in Layer 2	20 to 80
Activation Functions in Layer 1	relu, tanh, sigmoid
Activation Functions in Layer 2	relu, tanh, sigmoid
Epochs	10 to 150
Batch Size	16 to 64
Range of Hyperparameters for RF (Level-1)	
Parameter	**Range of Values**
Number of Estimators	10 to 1000
Max Depth	3 to 30
Min Sample Split	2 to 20
Min Sample Leaf	1 to 20
Max Features	auto, sqrt, log2
Max Leaf Nodes	10 to 1000
Min Impurity Decrease	0 to 0.1
Bootstrap	true, false
Criterion	squared error, absolute error, poisson, friedman mse

V. SYSTEM HEALTH ASSESSMENT

This section describes the system health assessment approach, focusing on layer concepts, model comparisons, and hyperparameter tuning in the classification models.

A. HEALTH PREDICTION LAYER

The health predictor represents the last layer in our predictive framework. It has a classification model designed to determine the system's health. Based on inputs from the forecasting layer that precedes it, this model categorizes the system's condition as either "healthy" or "unhealthy."

Moreover, we utilized the sliding window technique to incorporate data trends into the prediction process to enhance predictive accuracy. To forecast the target point, the model concentrates on the latest data within a specified range, defined by the window size. As illustrated in Fig. 11, cSysGuard adopts this technique for system health assessment, concentrating on the ten latest forecast points from the regression layer to ascertain the current system status. This method effectively captures the temporal patterns of

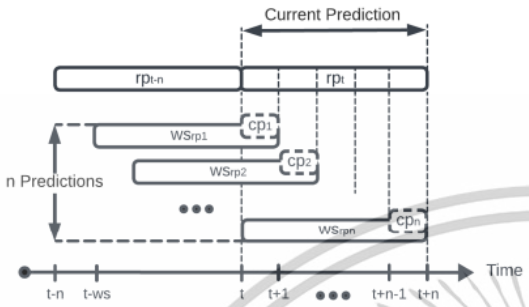


FIGURE 11. Sliding window technique in classification prediction. This figure demonstrates the sliding window technique for system health assessment, featuring the window size *ws* of regression predictions *rp* over *n* cycles, resulting in classification predictions *cp*.

data, leading to more accurate forecasts of future system conditions.

In addition to the imbalanced nature of our dataset [63], characterized by a higher prevalence of "healthy" statuses, there is a risk of models favoring the majority class and underperforming on the crucial but rarer "unhealthy" class. To address this, we implemented SMOTE to enhance the representation of the minority class, thereby enhancing the model equity by creating synthetic instances of this class to improve its detection.

B. MODEL COMPARISON

An essential aspect of our methodology is the comparative analysis of various machine learning models to assess their performance efficacy. For this analysis, we selected five prominent models: Decision Trees, Support Vector Machines, K-Nearest Neighbors, Logistic Regression, and Neural Networks. These models were initially performed with the default parameters provided by the Scikit-learn and TensorFlow libraries [55], [56]. In addition to Neural Networks, our configuration featured a two-layered structure: the first layer had six nodes using the ReLU activation function, and the second comprised a single node with a Sigmoid activation function.

To enhance the model's generalizability, the evaluation process employed 10-fold cross-validation, conducting several training rounds and validation on different data splits to derive the average scores. We utilized precision, recall, and F1 score with macro averaging [64], ensuring equal weighting of each category in the assessment throughout the cross-validation process. The model with the highest macro-averaged F1 score is the most optimal, as it demonstrates a balanced trade-off between precision and recall, with the specifics detailed in (9), (10), and (11).

$$\text{Precision}_{\text{macro}} = \frac{\text{Precision}_{\text{healthy}} + \text{Precision}_{\text{unhealthy}}}{2} \tag{9}$$

$$\text{Recall}_{\text{macro}} = \frac{\text{Recall}_{\text{healthy}} + \text{Recall}_{\text{unhealthy}}}{2} \tag{10}$$

TABLE 4. Hyperparameter ranges in decision tree tuning.

Parameter	Range of Values
Max Depth	3 to 500
Min Samples Split	2 to 100
Min Samples Leaf	1 to 100
Criterion	gini, entropy
Max Features	none, auto, sqrt, log2
Max Leaf Nodes	10 to 1000
Min Impurity Decrease	0 to 5

$$\text{F1-Score}_{\text{macro}} = 2 \times \frac{\text{Precision}_{\text{macro}} \times \text{Recall}_{\text{macro}}}{\text{Precision}_{\text{macro}} + \text{Recall}_{\text{macro}}} \tag{11}$$

C. HYPERPARAMETER TUNING

Subsequently, we carefully refined the model's hyperparameters through Bayesian optimization to enhance its predictive capabilities with our dataset. We methodically tested a range of parameter values to identify the combination that resulted in the highest macro-averaged F1 score. This process allowed us to optimize the model's accuracy and tailor the model specifically to the nuances of the dataset.

Table 4 presents the hyperparameters of the Decision Trees, identified as the top-performing model in Section VI-B. These influential hyperparameters encompass max depth, min sample split, min sample leaf, criterion, max features, max-leaf nodes, and min impurity decrease.

VI. RESULT AND DISCUSSION

This section presents our empirical findings for machine-learning models in system metrics and health forecasting. It also provides an insightful discussion of the findings.

A. SYSTEM METRICS FORECASTING EVALUATION

This subsection details the experimental results of the system metrics forecasting, including two main categories: model comparison and hyperparameter tuning.

1) MODEL COMPARISON

Table 5 provides a detailed performance evaluation of cSysGuard compared to the selected Level-0 models across various metrics. Deep learning models, such as GRU, LSTM, and RNN, show competitive predictive performance due to their ability to capture complex temporal dependencies and nonlinear patterns in the data. In contrast, traditional models like ARIMA and SARIMA generally lag in most metrics because they rely on more straightforward, linear assumptions and are less effective at handling the variability and complexity of dynamic cloud environments. Overall, cSysGuard shows superior proficiency in most areas. Notably, in response time, it achieves a significant 2.28-fold increase in performance compared to ETS. Notably, this comparison also highlights cSysGuard's capability to surpass advanced deep learning models, such as GRU and LSTM, typically known for their strong performance in predictive

TABLE 5. RMSE-based performance comparison of cSysGuard with selected level-0 models across specific metrics.

	cSysGuard	GRU	LSTM	RNN	CNN	ARIMA	SARIMA	TCN	ETS
CPU Request (percent)	**4.793**	4.917	5.008	4.796	5.254	6.816	7.724	5.825	
Memory Request (percent)	**1.494**	1.518	1.553	1.487	1.587	2.284			
Inbound Bandwidth (MB/s)	**2778**	2795	2827	2787	2897	3658		3070	
Outbound Bandwidth (MB/s)	**2368**	2388	2416	2368	2471	3122		2632	
TPS	**0.635**	0.641	0.646	0.637	0.674	0.834	0.831	0.707	
Response Time (ms)	**640.2**	649.1	665.7	642.3	742.0	1049.2		847.5	1460.0

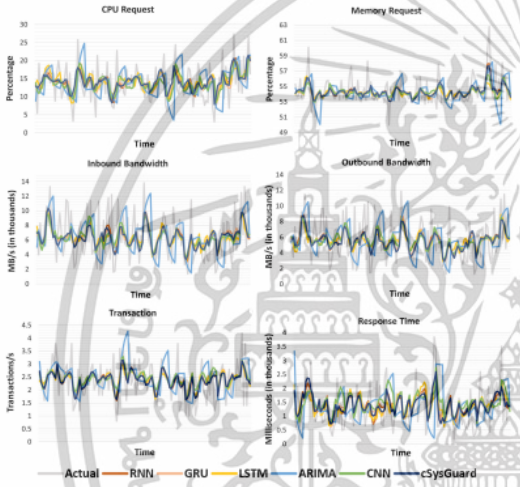


FIGURE 12. System performance metrics prediction. This figure compares cSysGuard's prediction results with those of commonly used models (GRU, RNN, LSTM, ARIMA, CNN) against actual results for CPU and memory requests, bandwidth, transaction rates, and response times.

tasks. These results demonstrate that cSysGuard effectively surpasses traditional models in certain areas, reinforcing its robustness and adaptability in diverse system-metric forecasting scenarios.

Additionally, Fig. 12 visualizes the predicted values compared to the actual values for various metrics. Each graph compares the performance of cSysGuard with commonly used models such as GRU, RNN, LSTM, ARIMA, and CNN. The visualizations clarify cSysGuard's predictive capabilities, highlighting its effectiveness in forecasting diverse system metrics. Given their alignment with performance benchmarks and the vast array of adjustable parameters, we opted for RNN and Random Forest as our Level-0 and Level-1 models in our optimization experiment.

2) HYPERPARAMETER TUNING

Table 6 lists the optimal RNN and Random Forest parameters for CPU usage prediction. Simultaneously, Table 7 presents the improved performance of each model and cSysGuard. Interestingly, the tuning process not only augmented the individual performances of the RNN and Random Forest but also

TABLE 6. Optimal parameters for recurrent neural networks (RNN) and random forest (RF) tuning in CPU request analysis.

Optimal RNN (Level-0) Parameters	
Parameter	**Value**
Observation Steps	64
Learning Rate	0.001
Number of Neurons in Layer 1	49
Number of Neurons in Layer 2	55
Activation Functions in Layer 1	relu
Activation Functions in Layer 2	relu
Epochs	100
Batch Size	16
Optimal RF (Level-1) Parameters	
Parameter	**Value**
Number of Estimators	545
Max Depth	27
Min Sample Split	5
Min Sample Leaf	10
Max Features	auto
Max Leaf Nodes	13
Min Impurity Decrease	0.00035269496460264014
Bootstrap	true
Criterion	friedman mse

TABLE 7. RMSE-based performance of recurrent neural networks (RNN), random forest (RF), and cSysGuard in CPU request (unit: percent).

	RNN	RF	cSysGuard
Base Performance	4.796	4.856	4.793
Level-0: Tuned RNN	**4.695**	**4.736**	**4.691**
Level-1: Tuned RF	4.695	**4.706**	**4.684**

enhanced the predictive capabilities of the subsequent layers in the architecture. This finding suggests a synergistic effect, where refining one system component positively influences overall efficacy, thereby underscoring the interconnected nature of the models within the system.

B. SYSTEM HEALTH ASSESSMENT EVALUATION

This subsection specifically delves into the experimental results of the system health assessment, dividing them into model comparison and hyperparameter tuning.

TABLE 8. Performance of base models and tuned decision trees in system health assessment.

	Macro-avg Precision	Macro-avg Recall	Macro-avg F1 Score
Base Performance			
Decision Trees	0.8856	0.8802	0.8812
Support Vector Machine	0.8255	0.8231	0.8223
K-Nearest Neighbors	0.8372	0.7819	0.7736
Neural Networks	0.8515	0.8159	0.8060
Logistic Regression	0.8380	0.8380	0.8368
Optimal Performance			
Tuned Decision Trees	0.9012	0.8968	0.8979

TABLE 9. Optimal parameters for tuning decision trees.

Parameter	Values
Max Depth	129
Min Samples Split	2
Min Samples Leaf	1
Criterion	gini
Max Features	none
Max Leaf Nodes	637
Min Impurity Decrease	0

1) MODEL COMPARISON

Table 8 presents a comparative analysis of various base models' performance, focusing on the macro-averaged precision, recall, and F1 score. The analysis reveals that the Decision Trees outperform the other base models by achieving the highest F1 score. This performance highlights its efficiency in accurately identifying positive instances while minimizing false positives, as evidenced by its high recall and precision. Such a balanced effectiveness profile underscores the suitability of Decision Trees for reliable system health status prediction, establishing it as a robust choice for such assessments. Based on the evaluation, we focused on Decision Trees for in-depth analysis during hyperparameter tuning.

2) HYPERPARAMETER TUNING

With Decision Trees identified as the top-performing base model, Table 8 offers a comparative analysis showcasing the impact of hyperparameter tuning on model performance. In addition, Table 9 presents the optimal parameter settings of our model. The results indicate that the optimized Decision Trees outperform the original model, showing enhancements of approximately 1.56% in precision, 1.66% in recall, and 1.67% in the F1 score. It demonstrates the improved performance of its original version and other base models.

Following the hyperparameter tuning process, we embarked on a detailed analysis of the model's classification accuracy. We constructed an averaged confusion matrix, denoted as Fig. 13. It synthesized the outcomes from each iteration of cross-validation conducted with our sample

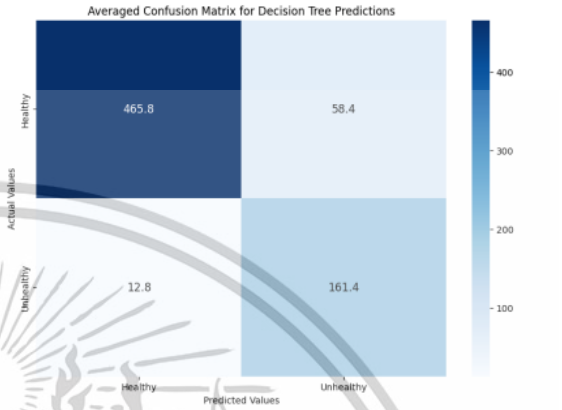


FIGURE 13. Confusion matrix visualization for system health assessment. This figure illustrates the averaged confusion matrix, capturing the predictive accuracy of server status as healthy or unhealthy.

data, offering a transparent and quantifiable measure of cSysGuard's predictive accuracy. The matrix delineates two principal classes indicative of server status: "healthy" and "unhealthy." In this matrix, columns represent predicted labels, and rows correspond to actual labels. This graphical representation is crucial for evaluating the precision and reliability of our model in distinguishing the server's operational states.

C. COMPARATIVE ANALYSIS OF DIFFERENT ENSEMBLE METHODOLOGY

Our study introduces an innovative ensemble stacking model to enhance predictive accuracy. In detail, the regression framework is a three-tiered architecture, comprising base models in the first layer, meta-models in the second, and employing a weighted averaging approach in the third. Then, we conducted a comparative analysis incorporating existing research studies alongside a novel experiment we devised, all utilizing similar ensemble modeling approaches.

Specifically, we have selected two research studies. Both employ two-tiered architectural frameworks. The first literature, by Mehmood et al. [23], proposed an ensemble model with a single meta-model (Decision Trees) in the second layer, which leverages predictions from the base models to determine the outcome. The second publication, by Kim et al. [22], introduced an ensemble model framework named CloudInsight. It incorporates a weighted average strategy in its second layer to aggregate and summarize predictions from the base models. In addition, we conducted an experiment that involves simple averaging in the third layer of cSysGuard. The study aimed to determine if a more streamlined model structure or a more straightforward aggregation method could achieve superior results without adding extra complexity. Therefore, we constructed these frameworks and applied them to our dataset to determine

TABLE 10. RMSE performance comparison: Single Meta-Model, CloudInsight, and cSysGuard (simple averaging: cSG-SA & weighted averaging: cSG-WA) across metrics.

	1 Meta-Model (Mehmood *et al.*)	Cloud Insight (Kim *et al.*)	cSG-SA	cSG-WA
CPU Request (percent)	6.6708	4.9131	4.6914	4.6845
Memory Request (percent)	2.1594	1.5442	1.4943	1.4941
Inbound Bandwidth (MB/s)	4052.5	2887.4	2778.2	2777.6
Outbound Bandwidth (MB/s)	3536.6	2462.9	2369.0	2368.7
TPS	0.9117	0.6377	0.6346	0.6348
Response Time (ms)	978.21	780.79	640.41	640.20

their effectiveness and compare them to the cSysGuard architecture.

Table 10 presents the comparative effectiveness of each methodology. Employing CloudInsight or a single meta-model does not match the performance efficacy of cSysGuard's configuration (cSG-WA). Alternatively, while using a simple averaging strategy in cSysGuard (cSG-SA) aligns closely with the current version, the critical advantage of weighted averaging is its flexibility; it enables users to tailor weight calculations based on specific needs, such as Level-1 model error sensitivity or collected error range. This flexibility enhances overall performance to favor better-performing models by fine-tuning their weights, emphasizing the importance of cSysGuard's nuanced three-tier layering strategy.

D. cSysGuard PERFORMANCE OVERHEAD

Subsequently, our analysis shifted towards assessing the impact on system performance, specifically examining the computational overhead. We experimented with cSysGuard on a host machine with a 2.9 GHz Intel Core i7-7820HK Quad-Core processor and 16GB DDR4 RAM. Table 11 showcases a comprehensive comparison, highlighting differences in key performance indicators such as average prediction and refinement times across three machine learning models (RNN, GRU, and LSTM) chosen for their substantial operational times and the cSysGuard (cSG) framework. Based on the result, cSysGuard exhibits marginally higher processing times than the selected high-performing models. However, it is essential to note that cSysGuard's processing duration is contingent upon the complexity of the underlying model it incorporates. Employing a more streamlined model within the system can reduce overall processing times for cSysGuard. Conversely, based on our findings, enhancing the host's resources can also reduce overall processing time. cSysGuard is fundamentally influenced by the characteristics

TABLE 11. Comparative analysis of prediction and refinement times (unit: seconds) for Selected ML models and cSysGuard (cSG).

	RNN	GRU	LSTM	cSG
Prediction Process	0.382	0.362	1.061	2.366
Refinement Process (Parallel Training)	74.29	132.40	141.84	152.55

of the employed models and the capabilities of its operating machine.

E. SYSTEM SPECIFICATION BIAS

As Section III-A2 outlined, our methodology deployed the system with specific configurations, including CPU type, memory capacity, and the operating system used. It is essential to acknowledge that differences in these specifications can influence the overall modeling process and its predictive outcomes. Such variations, in turn, can potentially affect the predictive accuracy and generalizability of cSysGuard.

Therefore, the study recommends that users consider adjusting cSysGuard's setup to suit their server specifications. Users can select different models and quantities, optimize parameters, fine-tune data configuration and prediction interval, or adjust Level-1 model error sensitivity. cSysGuard offers flexibility to align with users' operational goals and requirements, maximizing the advantages of ensemble learning.

VII. CONCLUSION AND FUTURE WORK

This paper introduces cSysGuard, a novel and resilient monitoring framework designed to enhance the prediction of system performance metrics and health assessments in dynamic cloud environments. To forecast performance metrics effectively, cSysGuard implements an advanced ensemble model using a stacking approach that integrates various predictors, such as the Level-0 and Level-1 models. Subsequently, the framework dynamically yields an accurate final metric prediction through a weighted averaging strategy. Furthermore, cSysGuard examines these forecasts to determine system failover through a fine-tuned classification model.

In our study, cSysGuard consistently outperformed the other standard models, encompassing traditional statistical and advanced deep learning approaches. It showcased significant proficiency, from nearly equivalent to an impressive increase of up to 2.28 times compared to conventional models. In addition to the system health assessment, it leveraged Decision Trees refined through hyperparameter tuning, achieving a macro-averaged F1 score of 89.79%. These outcomes highlight cSysGuard's effectiveness in delivering precise performance metrics and health analysis, showcasing it as a comprehensive system monitoring tool.

In future work, we intend to expand the capabilities of cSysGuard beyond its focus on detecting system-health failures based on resource sufficiency. Future iterations will

consider integrating other system failures, such as network-related issues, security breaches, and application-level errors. By addressing these additional dimensions, cSysGuard will become a more versatile framework adaptable to a broader range of scenarios and capable of providing comprehensive monitoring across various aspects of system health.

REFERENCES

[1] VMware by Broadcom. *What is Cloud Scalability?* Accessed: Jan. 14, 2024. [Online]. Available: https://www.vmware.com/topics/glossary/content/cloud-scalability.html

[2] Synopsys. *How Cloud Computing Flexibility Enables Design Changes*. Accessed: Jan. 14, 2024. [Online]. Available: https://www.synopsys.com/cloud/insights/cloud-computing-flexibility.html

[3] Red Hat. (Dec. 21, 2023). *Stateful vs Stateless*. Accessed: Jan. 14, 2024. [Online]. Available: https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless

[4] Q. Zhang and R. Boutaba, "Dynamic workload management in heterogeneous cloud computing environments," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, Krakow, Poland, May 2014, pp. 1–7, doi: 10.1109/NOMS.2014.6838288.

[5] L. Rosencrance, S. Louissaint, and K. Brush. (2024). *Service-Level Agreement (SLA)*. TechTarget. Accessed: Jan. 25, 2024. [Online]. Available: https://www.techtarget.com/searchitchannel/definition/service-level-agreement

[6] M. A. S. Netto, C. Cardonha, R. L. F. Cunha, and M. D. Assuncao, "Evaluating auto-scaling strategies for cloud computing environments," in *Proc. IEEE 22nd Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Paris, France, Sep. 2014, pp. 187–196, doi: 10.1109/MASCOTS.2014.32.

[7] Amazon Web Services. (Jul. 4, 2023). *EC2 Auto Scaling*. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html

[8] Google Cloud. *Autoscaling Groups of Instances*. Accessed: Jun. 14, 2023. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler

[9] P. Wenda. (Jul. 2, 2021). *AutoScaling: Introducing Compute Engine Predictive Autoscaling*. Google Cloud. Accessed: Jan. 14, 2024. [Online]. Available: https://cloud.google.com/blog/products/compute/introducing-compute-engine-predictive-autoscaling

[10] Google Cloud. *Scaling Based on Predictions*. Accessed: Jan. 14, 2024. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler/predictive-autoscaling

[11] N. Chairatana and R. Chawuthai, "Cloud stateless server failover prediction using machine learning on proactive system metrics," in *Proc. 18th Int. Joint Symp. Artif. Intell. Natural Lang. Process. (iSAI-NLP)*, Bangkok, Thailand, Nov. 2023, pp. 1–6, doi: 10.1109/isai-nlp60301.2023.10354585.

[12] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, "Cost-effective cloud HPC resource provisioning by building semi-elastic virtual clusters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Denver, CO, USA, Nov. 2013, pp. 1–12.

[13] A. A. Bankole and S. A. Ajila, "Predicting cloud resource provisioning using machine learning techniques," in *Proc. 26th IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, Regina, SK, Canada, May 2013, pp. 1–4, doi: 10.1109/CCECE.2013.6567848.

[14] N. Suksriupatham and A. Hoonlor, "Workload prediction with regression for over and under provisioning problems in multi-agent dynamic resource provisioning framework," in *Proc. 17th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Bangkok, Thailand, Nov. 2020, pp. 128–133, doi: 10.1109/JCSSE49651.2020.9268289.

[15] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015, doi: 10.1109/TCC.2014.2350475.

[16] K. Dmytro, T. Sergii, and P. Andiy, "ARIMA forecast models for scheduling usage of resources in IT-infrastructure," in *Proc. 12th Int. Sci. Tech. Conf. Comput. Sci. Inf. Technol. (CSIT)*, vol. 1, Lviv, Ukraine, Sep. 2017, pp. 356–360, doi: 10.1109/STC-CSIT.2017.8098804.

[17] P. Sekwatlakwatla, M. Mphahlele, and T. Zuva, "Traffic flow prediction in cloud computing," in *Proc. Int. Conf. Adv. Comput. Commun. Eng. (ICACCE)*, Durban, South Africa, Nov. 2016, pp. 123–128, doi: 10.1109/ICACCE.2016.8073735.

[18] D. Lee, D. Lee, M. Choi, and J. Lee, "Prediction of network throughput using ARIMA," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Fukuoka, Japan, Feb. 2020, pp. 1–5, doi: 10.1109/ICAIIC48513.2020.9065083.

[19] S. T. Singh, M. Tiwari, and A. S. Dhar, "Machine learning based workload prediction for auto-scaling cloud applications," in *Proc. OPJU Int. Technol. Conf. Emerg. Technol. Sustain. Develop. (OTCON)*, Chhattisgarh, India, Feb. 2023, pp. 1–6, doi: 10.1109/OTCON56053.2023.10114033.

[20] J. Gao, H. Wang, and H. Shen, "Machine learning based workload prediction in cloud computing," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Honolulu, HI, USA, Aug. 2020, pp. 1–9, doi: 10.1109/ICCCN49398.2020.9209730.

[21] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2010, pp. 456–463, doi: 10.1109/CLOUDCOM.2010.65.

[22] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Forecasting cloud application workloads with CloudInsight for predictive resource management," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1848–1863, Jul. 2022, doi: 10.1109/TCC.2020.2998017.

[23] T. Mehmood, S. Latif, and S. Malik, "Prediction of cloud computing resource utilization," in *Proc. 15th Int. Conf. Smart Cities, Improving Quality Life Using ICT IoT (HONET-ICT)*, Islamabad, Pakistan, Oct. 2018, pp. 38–42, doi: 10.1109/HONET.2018.8551339.

[24] S. Li, J. Bi, H. Yuan, M. Zhou, and J. Zhang, "Improved LSTM-based prediction method for highly variable workload and resources in clouds," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Toronto, ON, Canada, Oct. 2020, pp. 1206–1211, doi: 10.1109/SMC42975.2020.9283029.

[25] S. Bhagavathiperumal and M. Goyal, "Workload analysis of cloud resources using time series and machine learning prediction," in *Proc. IEEE Asia–Pacific Conf. Comput. Sci. Data Eng. (CSDE)*, Melbourne, VIC, Australia, Dec. 2019, pp. 1–8, doi: 10.1109/CSDE48274.2019.9162385.

[26] P. N. S. Chhetri, S. R. Kini, G. G. Mishra, and S. Kumar, "Resource provisioning in cloud using ARIMA and LSTM technique," in *Proc. IEEE 2nd Mysore Sub Sect. Int. Conf. (MysuruCon)*, Mysuru, India, Oct. 2022, pp. 1–7, doi: 10.1109/MysuruCon55714.2022.9972689.

[27] W. Chen, C. Lu, K. Ye, Y. Wang, and C.-Z. Xu, "RPTCN: Resource prediction for high-dynamic workloads in clouds based on deep learning," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Portland, OR, USA, Sep. 2021, pp. 59–69, doi: 10.1109/Cluster48925.2021.00038.

[28] M. Hassan, H. Chen, and Y. Liu, "DEARS: A deep learning based elastic and automatic resource scheduling framework for cloud applications," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustain. Comput. Commun. (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Melbourne, VIC, Australia, Dec. 2018, pp. 541–548, doi: 10.1109/BDCLOUD.2018.00086.

[29] A. Radwan, M. Amarneh, H. Alawneh, H. I. Ashqar, A. AlSobeh, and A. A. A. R. Magableh, "Predictive analytics in mental health leveraging LLM embeddings and machine learning models for social media analysis," *Int. J. Web Services Res.*, vol. 21, no. 1, pp. 1–22, Feb. 2024, doi: 10.4018/ijwsr.338222.

[30] J. H. Challis, "Signal processing," in *Experimental Methods in Biomechanics*. Cham, Switzerland: Springer, 2021, doi: 10.1007/978-3-030-52256-8_4.

[31] L. D. Paarmann, "Butterworth filters," in *Design and Analysis of Analog Filters* (The International Series in Engineering and Computer Science), vol. 617. Boston, MA, USA: Springer, 2003, doi: 10.1007/0-306-48012-3_3.

[32] S. Satpathy. (Nov. 17, 2023). *SMOTE for Imbalanced Classification With Python*. Analytics Vidhya. Accessed: Jan. 28, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques

[33] R. Dey and R. Mathur, "Ensemble learning method using stacking with base learner: A comparison," in *Proc. Int. Conf. Data Anal. Insights*, vol. 727. Singapore: Springer, 2023, pp. 159–169, doi: 10.1007/978-981-99-3878-0_14.

[34] A. C. Harvey, "ARIMA models," in *The New Palgrave* (Time Series and Statistics), J. Eatwell, M. Milgate, and P. Newman, Eds. London, U.K.: Palgrave Macmillan, 1990, doi: 10.1007/978-1-349-20865-4_2.

[35] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting With Exponential Smoothing: The State Space Approach*. Cham, Switzerland: Springer, 2008, doi: 10.1007/978-3-540-71918-2.

[36] G. James, D. Witten, T. Hastie, and R. Tibshirani, "Linear regression," in *An Introduction to Statistical Learning* (Springer Texts in Statistics). New York, NY, USA: Springer, 2021, doi: 10.1007/978-1-0716-1418-1_3.
[37] Y. Liu, Y. Wang, and J. Zhang, "New machine learning algorithm: Random forest," in *Information Computing and Applications* (Lecture Notes in Computer Science), vol. 7473, B. Liu, M. Ma, and J. Chang, Eds. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-642-34062-8.
[38] S. Skansi, "Feedforward neural networks," in *Introduction to Deep Learning* (Undergraduate Topics in Computer Science). Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-73004-2_4.
[39] S. Tripathy and R. Singh, "Convolutional neural network: An overview and application in image classification," in *Proc. 3rd Int. Conf. Sustain. Comput.*, vol. 1404. Singapore: Springer, 2022, doi: 10.1007/978-981-16-4538-9_15.
[40] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks: A unified approach to action segmentation," in *Proc. Comput. Vis. ECCV Workshops*, in Lecture Notes in Computer Science, vol. 9915. Cham, Switzerland: Springer, 2016, pp. 47–54.
[41] S. A. Marhon, C. J. F. Cameron, and S. C. Kremer, "Recurrent neural networks," in *Handbook on Neural Information Processing* (Intelligent Systems Reference Library), vol. 49, M. Bianchini, M. Maggini, and L. Jain, Eds. Berlin, Heidelberg: Springer, 2013, doi: 10.1007/978-3-642-36657-4_2.
[42] S. Kostadinov, (Dec. 16, 2017). *Understanding GRU Networks*. Towards Data Sci. Accessed: Jan. 17, 2024. [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
[43] K. Smagulova and A. P. James, "Overview of long short-term memory neural networks," in *Deep Learning Classifiers With Memristive Networks* (Modeling and Optimization in Science and Technologies), vol. 14, A. James, Ed. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-14524-8_11.
[44] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986, doi: 10.1007/bf00116251.
[45] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, Jul. 1995, doi: 10.1007/BF00994018.
[46] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.
[47] J. S. Cramer, "The origins and development of the logit model," in *Logit Models From Economics and Other Fields*. Cambridge, U.K.: Cambridge Univ. Press, 2003, pp. 149–157, doi: 10.1017/CBO9780511615412.010.
[48] B. Mer, J. Reinhardt, and M. T. Strickland, "Neural networks: An introduction," in *Physics of Neural Networks*, 2nd ed. Berlin, Germany: Springer, 1995, doi: 10.1007/978-3-642-57760-4.
[49] *Digital Ocean*. Accessed: Jan. 14, 2024. [Online]. Available: https://www.digitalocean.com
[50] Google Cloud. *What is a Virtual Machine?* Accessed: Jan. 14, 2024. [Online]. Available: https://cloud.google.com/learn/what-is-a-virtual-machine
[51] *Prometheus*. Accessed: Jun. 15, 2023. [Online]. Available: https://prometheus.io
[52] *Zabbix*. Accessed: May 26, 2023. [Online]. Available: https://www.zabbix.com
[53] *MDN Web Docs: HTTP Response Status Codes*. Accessed: Jun. 15, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
[54] *Statsmodels*. Accessed: Jan. 15, 2023. [Online]. Available: https://www.statsmodels.org
[55] *TensorFlow*. Accessed: Jan. 15, 2023. [Online]. Available: https://www.tensorflow.org
[56] *Scikit-learn*. Accessed: Jan. 15, 2023. [Online]. Available: https://www.scikit-learn.org
[57] N. Chairatana and R. Chawuthai, Jan. 31, 2024, "Cloud stateless system performance metrics and status," IEEE Dataport, doi: 10.21227/8wf2-2y40.
[58] S. Turney. (Jun. 22, 2023). *Pearson Correlation Coefficient*. Scribbr. Accessed: Jan. 15, 2024. [Online]. Available: https://www.scribbr.com/statistics/pearson-correlation-coefficient
[59] B. Saji. (2021). *Elbow Method for Finding the Optimal Number of Clusters in K-Means*. Analytics Vidhya. Accessed: Jan. 15, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning
[60] D. Schumacher. *Understanding Weighted Average*. SERP AI. Accessed: Jan. 15, 2024. [Online]. Available: https://serp.ai/weighted-average
[61] ScienceDirect. *Integer Overflow*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/integer-overflow
[62] W. Wang. (2020). *Bayesian Optimization Concept Explained in Layman Terms*. Towards Data Sci. Accessed: Jan. 15, 2024. [Online]. Available: https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f
[63] J. Brownlee. (2020). *A Gentle Introduction to Imbalanced Classification*. Mach. Learn. Mastery. Accessed: Jun. 15, 2023. [Online]. Available: https://machinelearningmastery.com/what-is-imbalanced-classification
[64] A. Tariq. (2023). *What is the Difference Between Micro and Macro Averaging?* Educative. Accessed: Jun. 15, 2023. [Online]. Available: https://www.educative.io/answers/what-is-the-difference-between-micro-and-macro-averaging

NUTT CHAIRATANA received the B.Eng. degree in computer innovation engineering from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 2022, where he is currently pursuing the M.Eng. degree in robotics and computational intelligence systems with the Department of Robotics and AI Engineering.

From 2022 to 2023, he was a Software Engineer with RentSpree. He is a Machine Learning Engineer with Kasikorn Business-Technology Group, Thailand.

RATHACHAI CHAWUTHAI received the B.Eng. degree in computer engineering from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 2006; the M.Eng. degree in information management from Asian Institute of Technology, Pathum Thani, Thailand, in 2012; and the Ph.D. degree in informatics from SOKENDAI and National Institute of Informatics, Kanagawa, Japan, in 2016.

From 2006 to 2010, he was a Software Engineer with Thomson Reuters. From 2012 to 2013, he was a Senior Software Engineer with Punsarn Asia. From 2013 to 2016, he was a Research Assistant with the National Institute of Informatics. He is currently an Assistant Professor with the King Mongkut's Institute of Technology Ladkrabang.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายณัฏฐ์ ชัยรัตน์
วัน เดือน ปีเกิด	13 ธันวาคม 2542
ที่อยู่	99/1 ซ.สุขสวัสดิ์ 22 ถ.สุขสวัสดิ์
	แขวงบางปะกอก เขตราษฎร์บูรณะ กรุงเทพฯ 10140 โทรศัพท์ 086-392-2658
ประวัติการศึกษา	2565 วิศวกรรมศาสตรบัณฑิต สาขาวิชานวัตกรรมคอมพิวเตอร์ (เกียรตินิยมอันดับ 1)
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ความชำนาญเฉพาะด้าน	1.) การออกแบบระบบ และพัฒนาซอฟต์แวร์
	2.) การเรียนรู้ของเครื่อง (Machine Learning)
	3.) ระบบคลาวด์ (Cloud Computing)
ประสบการณ์การทำงานและผลงานวิจัย	
พ.ศ.2565-2566	ตำแหน่งนักพัฒนาซอฟต์แวร์ที่บริษัท RentSpree
พ.ศ.2566-ปัจจุบัน	ตำแหน่งนักพัฒนาซอฟต์แวร์ (Machine Learning)
	ที่บริษัท KASIKORN Business-Technology Group
พ.ศ.2563-2564	ทุนการศึกษาเรียนดี ณ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2566	ผลงานวิจัย Cloud Stateless Server Failover Prediction Using Machine Learning on Proactive System Metrics. 18th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP), Bangkok, Thailand, 2023, pp. 1-6, doi: 10.1109/iSAI-NLP60301.2023.10354585
พ.ศ.2567	ผลงานวิจัย Stateless System Performance Prediction and Health Assessment in Cloud Environments: Introducing cSysGuard, an Ensemble Modeling Approach. IEEE Access, vol. 12, pp. 78232-78247, 2024, doi: 10.1109/ACCESS.2024.3406670