การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 48
The 48th Electrical Engineering Conference (EECON-48)
วันที่ 19-21 พฤศจิกายน 2568 ณ โรงแรมฟูราม่า จังหวัดเชียงใหม่

การประยุกต์ใช้และปรับแต่งโมเดลภาษาขนาดใหญ่ สำหรับการสร้างกรณีทดสอบอัตโนมัติในงานทดสอบซอฟต์แวร์

Application and Fine-Tuning of Large Language Models with LangChain for Test Case Generation in Software Testing

หทัยรัตน์ เจนวิทยา และ รัฐศิลป์ รานอกภานุวัชร์

หลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ วิทยาลัยวิศวกรรมศาสตร์และเทคโนโลยี มหาวิทยาลัยธุรกิจบัณฑิตย์
66130662@dpu.ac.th ratthaslip.ran@dpu.ac.th

บทคัดย่อ

งานวิจัยนี้การประยุกต์ใช้และปรับแต่งโมเดลLLM ได้แก่ LLaMA3.2-3B, Typhoon2-8b และ Gemma2-9b สำหรับสร้างกรณีทดสอบจากข้อกำหนดซอฟต์แวร์ โดยใช้ Fine-tuning กับชุดข้อมูลและออกแบบระบบ RAG ด้วย LangChain เพื่อดึงบริบทผ่าน Similarity Search ช่วยให้โมเดลสร้างกรณีทดสอบที่แม่นยำและตอบสนองแบบอัตโนมัติได้อย่างมีประสิทธิภาพและการประเมินประสิทธิภาพของโมเดล Typhoon2-8b โดยมีค่า Loss ต่ำสุดที่ 0.1543 และค่า Cosine Similarity สูงสุดที่ 0.8770 พร้อมค่าระยะ Euclidean และ Manhattan ต่ำสุด รองลงมา Gemma2-9b มีค่า Loss สูงสุด 0.1971 Cosine Similarity 0.8768 Euclidean 0.4511 และ Manhattan 9.7364 ในขณะที่ LLaMA3.2-3B มีค่า Loss 0.1773 Cosine Similarity 0.8762 Euclidean 0.4516 และ Manhattan 9.7489 ค่าตัวชี้วัดอยู่ในระดับกลางทุกด้าน การประเมินความครอบคลุมโมเดล Gemma2-9b มีความครอบคลุมในทุกมิติสูงที่สุดโดยสามารถสร้าง Test Case ได้หลากหลายทั้ง Positive และ Negative Cases และเชื่อมโยงกับ Requirement ได้ครบถ้วนที่สุดรองลงมา LLaMA3.2-3B และ Typhoon2-8b ผลการประเมินจากผู้เชี่ยวชาญโดยโมเดล Gemma2-9b ได้รับคะแนนมากที่สุด ความพึงพอใจสูงสุดในด้านคุณภาพของ Test Case รองลงมา LLaMA3.2-3B และ Typhoon2-8b

คำสำคัญ: โมเดลภาษาขนาดใหญ่, การสร้างกรณีทดสอบ, การทดสอบซอฟต์แวร์, ตัวชี้วัดความครอบคลุมในการทดสอบ, การฝังเวกเตอร์

Abstract

This research applies and fine-tunes ModelsLLMs namely LLaMA3.2-3B, Typhoon2-8b, and Gemma2-9b for automated test case generation based on software requirements. Fine-tuning was applied to the models using a dedicated dataset and a RAG system was designed using the LangChain framework to retrieve contextual information via similarity search This integration enables the models to generate accurate and responsive test cases efficiently Performance evaluation shows that Typhoon2-8b achieves the lowest loss 0.1543 and the highest cosine similarity 0.877 along with the lowest Euclidean 0.4501 and Manhattan 9.7200 Gemma2-9b records the highest loss 0.1971 cosine similarity of 0.8768 Euclidean distance of 0.4511 and Manhattan distance 9.7364 LLaMA3.2-3B shows moderate performance across all metrics with a loss 0.1773 cosine similarity 0.8762 Euclidean distance 0.4516 and Manhattan distance 9.7489 Regarding coverage evaluation including test coverage, functional coverage, and requirement coverage Gemma2-9b demonstrated the highest comprehensiveness across all dimensions. It generated a wide variety of test cases both positive and negative and maintained the most complete linkage to the original requirements. LLaMA3.2-3B and Typhoon2-8b ranked second and third, respectively Expert evaluations revealed that Gemma2-9b received the highest scores in terms of test case quality and usability followed by LLaMA3.2-3B and Typhoon2-8b

Keywords: Large Language Models, Test case generation, Software Testing, Coverage Metrics, Vector Embedding

1. บทนำ

1.1 ที่มาและความสำคัญ

งาน Software Tester หรือ Software QA มีบทบาทสำคัญในการตรวจสอบคุณภาพของซอฟต์แวร์ให้พร้อมใช้งานได้อย่างถูกต้องและตรงตามวัตถุประสงค์โดยเฉพาะในกระบวนการสร้าง Test Case สำหรับ Manual Testing ซึ่งเป็นงานที่ใช้เวลาและความชำนาญสูง งานวิจัยนี้พัฒนาและประยุกต์ใช้โมเดล LLM สำหรับการสร้างกรณีทดสอบแบบ Manual ในระดับ Functional Testing โดยมีเป้าหมายเพื่อประสิทธิภาพในการทดสอบซอฟต์แวร์

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อพัฒนาเครื่องมือที่ใช้ Large Language Model และปรับแต่ง Fine-tuning โมเดล LLaMA3.2-3B, Typhoon2-8b, Gemma2-

*ผู้ประพันธ์บรรณกิจ

9b สำหรับการสร้างกรณีทดสอบ Functional Testing จาก Requirements

1.2.2 เพื่อพัฒนาและออกแบบระบบโดยใช้แนวคิด Retriever - Augmented Generation (RAG) โดยเชื่อมต่อโมเดล LLM เข้ากับฐานข้อมูล LangChain Framework เพื่อสร้าง Test Case ที่มีความสอดคล้องกับ Requirements

1.3 ขอบเขตการวิจัย

1.3.1 พัฒนาโมเดล LLMs ได้แก่ LLaMA3.2-3B, Typhoon2-8b และ Gemma2-9b ปรับแต่งด้วย Fine-tuning และประเมินความสอดคล้องค่า Loss โดยใช้ Vector Embedding ประยุกต์ใช้เทคนิค Retrieval-Augmented Generation เพื่อดึงข้อมูลบริบทที่เกี่ยวข้องจากฐานข้อมูลเชื่อมต่อผ่าน LangChain Framework จัดการ Prompt และควบคุมการทำงานของ Chain ระหว่างโมเดลและ Agent Tool

1.3.2 ชุดข้อมูลที่ใช้ในการวิจัยออกแบบมาเพื่อการสร้าง Test Case ที่เกี่ยวข้องกับระบบธนาคาร โดยมุ่งเน้นทั้งด้านธุรกรรมการเงินโดยรูปแบบ Manual Testing ครอบคลุมทั้ง Positive และ Negative Cases จำนวน 17,000 Test Cases พร้อมทั้งประเมินความพึงพอใจผู้เชี่ยวชาญด้าน Software Testing

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 พัฒนาขึ้นเพื่อช่วยลดภาระงานของนักทดสอบ Tester, QA ในการสร้าง Test Cases แบบ Manual Testing ทำให้ประหยัดเวลาในกระบวนการทดสอบซอฟต์แวร์และช่วยลดโอกาสที่ข้อผิดพลาด

1.4.2 เพิ่มความครอบคลุมของการทดสอบ Test Coverage, Functional Coverage และ Requirement Coverage โมเดลที่พัฒนาสามารถสร้างกรณีทดสอบที่ครอบคลุมทั้ง Positive และ Negative Cases รองรับการโต้ตอบแบบ ChatBot สามารถป้อนความต้องการและรับผลลัพธ์ได้ทันที โดยไม่จำเป็นต้องมีความรู้ด้านการเขียน Test Case

2. แนวคิดทฤษฎี และงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎีที่เกี่ยวข้อง

Large Language Models ใช้โครงสร้างแบบ Transformer ผ่านการฝึกเบื้องต้น Pre-training โดยเรียนรู้จากบริบทของข้อความเพื่อทำนายปรับแต่ง Fine-tuning ด้วยชุดข้อมูลเฉพาะด้าน Loss Function สำหรับ LLMs อาศัย Loss Function เพื่อวัดว่าผลลัพธ์ของโมเดล

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \quad (1)$$

เทคนิค LoRA (Low-Rank Adaptation) เป็นการปรับแต่งโมเดล Fine-tuning ที่เน้นลดการใช้ทรัพยากรคำนวณและหน่วยความจำ โดยทำงานผ่านการตรึง freeze พารามิเตอร์หลักของโมเดล และ low-rank matrices เพื่อเรียนรู้การเปลี่ยนแปลงสำหรับงานช่วยให้สามารถปรับโมเดลขนาดใหญ่ได้อย่างมีประสิทธิภาพโดยไม่ต้องปรับพารามิเตอร์ทั้งหมด

$$\Delta W = A \times B \quad (2)$$

โดยที่ A และ B จะต้องเป็น Low-Rank Matrix ซึ่งจะเป็น matrix ที่ขนาดเล็กกว่าเมื่อเทียบกับ weight ของโมเดล LLMs ที่จะทำการปรับแต่ง โดยจะทำให้จำนวนพารามิเตอร์ที่ใช้ในการฝึกมีขนาดลดลงและ ΔW จะเป็นค่า weight ที่เปลี่ยนแปลงไปจากโมเดลเดิม

Cosine Similarity การวัด "ค่าความคล้ายกันเชิงมุม" ระหว่างเวกเตอร์สองตัวในปริภูมิ n มิติ โดยไม่คำนึงถึงความยาว Magnitude ของเวกเตอร์ พิจารณาที่ทิศทางของเวกเตอร์เป็นหลัก หากเวกเตอร์มีทิศทางเดียวกันจะได้ค่าใกล้ 1 ซึ่งแปลว่ามีความคล้ายกันสูง ในทางกลับกัน หากทิศทางต่างกันโดยสิ้นเชิงจะได้ค่าใกล้ -1 เหมาะสำหรับการวัดความคล้ายกันของข้อความหรือประโยคที่ถูกแปลงเป็นเวกเตอร์

$$Cosine\ Similarity(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (3)$$

Euclidean Distance การวัดระยะทางจริงระหว่างจุดสองจุดในพื้นที่ n มิติวัดระยะทางระหว่างสองจุด ใช้วัดว่าเวกเตอร์ A และ B อยู่ห่างกันในเชิงพิกัด z นำค่าของแต่ละมิติมาหา “ผลต่างยกกำลังสอง” ถอดราก

$$Euclidean(A,B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \quad (4)$$

Manhattan Distance เป็นการวัดระยะทางระหว่างเวกเตอร์สองตัวโดยรวมระยะทางของแต่ละแกนโดยไม่ยกกำลังหรือถอดรากวัดตามแนวแกนเหมาะสำหรับข้อมูลที่มีลักษณะไม่ต่อเนื่อง หรือกรณีที่ต้องการลดผลกระทบจากค่าที่สูงเกินไปในบางมิติ

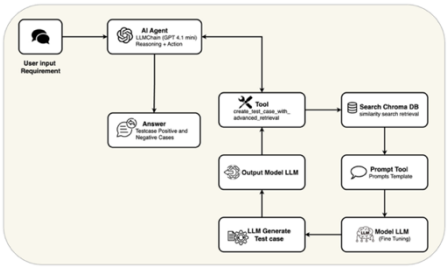
$$Manhattan(A,B) = \sum_{i=1}^{n} |A_i - B_i| \quad (5)$$

LangChain เป็นเฟรมเวิร์กที่ออกแบบมาเพื่อจัดการการทำงานร่วมกันระหว่างระบบที่ใช้โมเดล LLM ได้แก่ Prompt, Chain, Agent, Tool, Memory และ Retrieval ระบบสามารถดึงข้อมูลภายนอกโดย Retrieval Chain แบบเวกเตอร์ Embedding

Embedding เป็นเทคนิคพื้นฐานที่ใช้แปลงข้อมูลที่อยู่ในรูปแบบของข้อความให้กลายเป็นเวกเตอร์ตัวเลขในเชิงคณิตศาสตร์ เพื่อให้เวกเตอร์ที่ได้สะท้อนความหมายของคำหรือประโยคอย่างถูกต้อง

3. วิธีดำเนินการวิจัย

3.1 ภาพรวมของระบบโดยใช้ Langcain



รูปที่ 1 สถาปัตยกรรมของระบบ System Architecture

3.2 การออกแบบการวิจัย Research Design

3.2.1 คำถาม Input Stage ระบบจะรับข้อกำหนดซอฟต์แวร์ Requirements จากผู้ใช้ผ่านทาง Web Interface ที่พัฒนาด้วย Streamlit

3.2.2 วิเคราะห์ถึงบริบท Context Analysis with RAG ระบบนำ Requirements ที่ได้รับมาเข้าสู่กระบวนการวิเคราะห์เชิงบริบทโดยใช้โมเดล LLM และเชื่อมต่อกับระบบ RAG การดึงบริบทโดยใช้ Vector Search ผ่านฐานข้อมูลเวกเตอร์ ChromaDB ใช้การแปลงข้อความเป็น Vector embedding ค้นหาจากฐานข้อมูลด้วย semantic similarity

3.3.3 ประมวลผล Processing & Test Case Generation นำบริบทและ Requirement ที่สรุปได้เข้าสู่โมเดล GPT-4.1 mini และโมเดล LLM ที่ทำ Fine-Tuned เพื่อให้ระบบวิเคราะห์สร้าง Test Case [1]

3.3.4 ตอบคำถาม Answer Generation & Memory Storage การประมวลผลจะถูกส่งกลับให้ผู้ใช้งานในรูปแบบคำตอบ Positive และ Negative Test Cases เก็บประวัติการใช้งานเพื่อใช้ต่อยอดการสนทนา

3.3 การเก็บรวบรวมข้อมูล Data Collection

ชุดข้อมูล Test Cases Positive และ Negative Cases ออกแบบระบบธนาคาร เช่น การฝากถอนเงิน, แสดงรายการเดินบัญชี และการสแกนใบหน้า โดยจัดเก็บในไฟล์ CSV ได้แก่ Test Case ID, Requirement Text, Test Case Description, Test Steps, Expected Result, Test Case Type, Label, Automation Status, Priority ข้อมูล 17,000 Test Case มี Positive Cases จำนวน 13,492 และ Negative Cases จำนวน 3,508 รายการ จัดลำดับความสำคัญ Medium 15,655 Test Case, High 825 Test Case, Critical 271 Test Case, Low 249 Test Case สำหรับการแบ่งชุดข้อมูลได้ถูกแยกออกเป็น 2 ส่วน Training set จำนวน 13,600 Test Case (80%) และ Validation set จำนวน 3,400 Test Case (20%)

3.4 เครื่องมือและเทคโนโลยีที่ใช้ Tools and Technologies

3.4.1 Large Language Model โมเดลภาษาขนาดใหญ่ LLaMA3.2-3B, Typhoon2-8b, Gemma2-9b ถูกใช้เป็นเครื่องมือนำมาใช้ในกระบวนการปรับแต่ง Fine-tuning

3.4.2 การพัฒนา Backend ภาษา Python และใช้เฟรมเวิร์ก LangChain สำหรับบริหารจัดการการเชื่อมต่อระหว่างโมเดลภาษา LLMs กับข้อมูลจากระบบ RAG (Retrieval-Augmented Generation) และฐานข้อมูลเวกเตอร์

3.4.3 Agent แบบ ReAct ใช้ Agent ที่สามารถตัดสินใจเลือกเครื่องมือ Tool ให้เหมาะสมกับ Intent ของผู้ใช้ โดยออกแบบ Agent ตามหลัก ReAct การวิเคราะห์และขั้นตอน ส่งผลให้ระบบมีความยืดหยุ่นในการตอบสนองและเลือก pipeline การประมวลผลอย่างเหมาะสม

3.4.4 Memory Conversation ระบบจัดการ BufferMemory เพื่อเก็บบริบทการสนทนาในแต่ละ session ซึ่งช่วยให้ LLM สามารถอ้างอิงถึงบทสนทนาก่อนหน้า ทำให้การตอบคำถามมีความต่อเนื่องและมีความเข้าใจบริบทที่ดีขึ้น

3.4.5 ฐานข้อมูล ChromaDB ระบบใช้ฐานข้อมูลเวกเตอร์ ChromaDB เก็บ embedding vectors ของเอกสารหรือบริบทที่เกี่ยวข้องกับงานทดสอบซอฟต์แวร์เพื่อใช้ในการสร้างคำตอบโดย LLM

3.4.6 LangChain เป็นไลบรารีหลักที่ใช้ในการสร้าง Agent, Chain, Tool, PromptTemplate และ Memory โดยเฉพาะอย่างยิ่งเหมาะกับงานที่ต้องเชื่อมต่อ LLM กับข้อมูลภายนอกเข้าด้วยกันใน Chain เดียว

3.4.7 OpenAI API ใช้สำหรับเรียกใช้โมเดล GPT-4.1-mini ที่มีประสิทธิภาพสูงในการสรุป วิเคราะห์ และประเมินผล Test Case

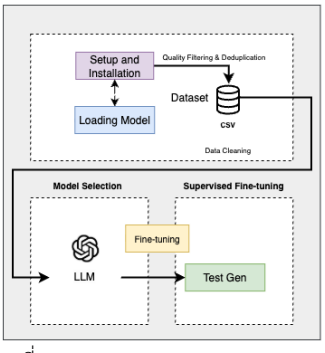
3.4.8 การพัฒนา Frontend โดยใช้ Streamlit เป็นเครื่องมือสำหรับสร้าง Chatbot โอเพนซอร์ซที่ใช้สร้างแอปพลิเคชันเว็บเชิงโต้ตอบ

3.5 การพัฒนาโมเดล

การออกแบบ Prompt เพื่อสร้าง Test Case ของโมเดล LLM ในการสร้างชุดทดสอบซอฟต์แวร์ Test Case โครงสร้างของ Prompt แบ่งออกเป็น 3 ส่วน 1. System Prompt บทบาทของโมเดลกำหนดให้โมเดลมีบทบาทเป็น Software Tester มืออาชีพ ที่มีหน้าที่สร้าง Test Case โดยพิจารณาทั้งด้าน Functional, Positive/Negative, Security และ Performance Testing 2. User Prompt Requirement ที่ผู้ใช้ป้อนข้อความที่ผู้ใช้ระบุ "ความต้องการของระบบ" เพื่อให้โมเดลสร้าง Test Case ตามบริบท 3. Expected Output Format คำตอบจากโมเดลเพื่อให้การสร้าง Test Case โดยอัตโนมัติจากโมเดลภาษา LLM

3.6 การปรับแต่งโมเดล Fine-tuning

การตั้งค่าพารามิเตอร์สำหรับ Supervised Fine-Tuning ด้วย SFTTrainer และ Hugging Face Transformers การฝึกโมเดล LLM ในการ fine-tune [2]



รูปที่ 2 กระบวนการ Fine-tuning Data

ตารางที่ 1 การตั้งค่าพารามิเตอร์ระหว่างการฝึกโมเดล

พารามิเตอร์	ค่า	คำอธิบาย
per_device_train_batch_size	1	จำนวนตัวอย่างที่ฝึกต่อ GPU
gradient_accumulation_steps	2	จำนวนรอบการสะสม gradient
warmup_steps	5	จำนวน step ที่เพิ่ม learning rate
num_train_epochs	2	จำนวนรอบที่โมเดลจะฝึกกับข้อมูล
learning_rate	2e-4	อัตราการเรียนรู้ที่ใช้ในระหว่างฝึก
logging_steps	1	ความถี่ log
optim	8	ประสิทธิภาพสูงหน่วยความจำน้อยลง
weight_decay	0.01	อัตราลดน้ำหนักพารามิเตอร์ป้องกันการ overfitting

3.7 กระบวนการวัดความคล้ายคลึงระหว่างเวกเตอร์

โดยใช้ Vector Embedding โดย Cosine Similarity วัดมุมระหว่างเวกเตอร์ ค่ายิ่งใกล้ 1 ยิ่งคล้ายกัน, Euclidean Distance วัดระยะทางตรงระหว่างเวกเตอร์ ค่ายิ่งใกล้ 0 ยิ่งคล้าย และ Manhattan Distance วัดระยะทางแบบรวมผลต่างทีละแกน ค่ายิ่งใกล้ 0 ยิ่งคล้ายเช่นกัน

3.8 กระบวนการทำงานของ Lang Chain

3.8.1 การออกแบบ Agent ด้วยแนวคิด ReAct (Reasoning + Acting) ใน LangChain ช่วยให้ Agent ที่ขับเคลื่อนด้วย LLM เช่น GPT 4.1mini สามารถวิเคราะห์ปัญหาและเลือกใช้เครื่องมือ Tool ทำให้ระบบมีความยืดหยุ่นและตอบสนองได้อย่างชาญฉลาด

3.8.2 การสร้าง Chains เพื่อควบคุมลำดับขั้นตอนอัตโนมัติ LangChain ใช้โครงสร้าง Chain เพื่อเชื่อมโยงการทำงาน และเก็บบริบทการสนทนา Conversation Buffer Memory รูปแบบ Short Term Memory เป็นหน่วยความจำระยะสั้น

3.8.3 การเชื่อมต่อระบบ RAG โดยกำหนดเป็น Retrieval QAChain หรือกำหนด Custom Chain ที่เชื่อมกับฐานข้อมูลเวกเตอร์ Chroma DB ในระบบใช้การแปลง Requirement เป็น Embedding บริบทที่เกี่ยวข้องจากฐานข้อมูลเวกเตอร์

3.8.4 การใช้ Prompt Template สำหรับสร้าง Prompt ที่มีตัวแปรกำหนดได้ requirement, context กับโมเดลได้แม่นยำ

3.8.5 การจัดการ Agent Tool เลือก Tool การประมวลผล Chain

4. ผลการวิจัย

4.1 ผลการวิจัยประสิทธิภาพ

การทดสอบ 3 โมเดล LLaMA3.2-3B, Typhoon2-8b, Gemma2-9b บน Testing Set โดยให้ทำการเก็บค่าเพื่อประเมินประสิทธิภาพของเครื่องมือที่พัฒนาขึ้นโดยใช้ค่า Loss เป็นเกณฑ์ในการประเมิน

ตารางที่ 2 การวิจัยประสิทธิภาพโดยใช้ค่า Loss

Model	Loss Average
LLaMA3.2-3B	0.1773
Typhoon2-8b	0.1543
Gemma-2-9b	0.1971

4.2 ผลลัพธ์การประเมินโมเดล

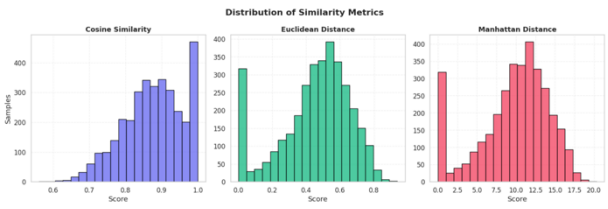
ระบบใช้เมตริกหลัก 3 รายการ ได้แก่ ค่าเฉลี่ย Cosine Similarity, Euclidean Distance และ Manhattan Distance เปรียบเทียบเวกเตอร์ Embedding ระหว่างคำตอบจากโมเดลกับคำตอบจริง โดยใช้โมเดลฝังข้อมูล Alibaba-NLP/gte-multilingual-base [3]

ตารางที่ 3 การประเมินโดยใช้ชุดข้อมูลทดสอบ

Model	Cosine Similarity	Euclidean Distance	Manhattan Distance
LLaMA3.2-3B	0.8762	0.4516	9.7489
Typhoon2-8b	0.8770	0.4501	9.7200
Gemma-2-9b	0.8768	0.4511	9.7364

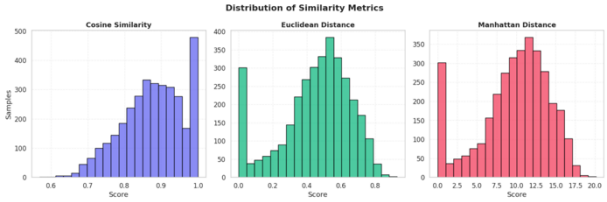
4.3 การประเมินความใกล้เคียงระหว่างคำตอบ

การประเมินความใกล้เคียงระหว่างคำตอบที่ได้โดยพิจารณาจาก 3 มาตรวัดหลัก ได้แก่ Cosine Similarity ค่า > 0.8 ใช้วัดทิศทางของเวกเตอร์เพื่อประเมินความคล้ายเชิงบริบท โดยค่าที่ใกล้ 1 คำตอบใกล้เคียงกับคำตอบจริง, Euclidean Distance < 0.5 วัดระยะห่างเชิงตำแหน่งของเวกเตอร์ ค่าน้อยแสดงถึงความใกล้กันของข้อมูลและ Manhattan Distance < 12 ใช้ค่าเฉลี่ยลบด้วยส่วนเบี่ยงเบนมาตรฐาน 30% เพื่อจำแนกคำตอบที่ดีในข้อมูลแบบเวกเตอร์ค่าที่ต่ำบ่งชี้ถึงความใกล้กันในหลายมิติ [4]



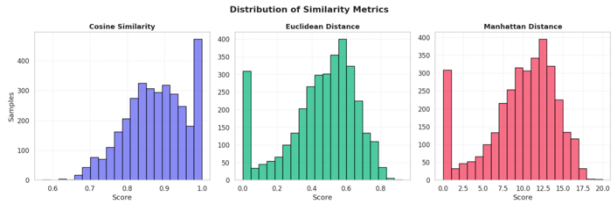
รูปที่ 3 ค่าเฉลี่ยความคล้าย Model LLaMA3.2-3B

Model LLaMA3.2-3B มีค่า Cosine Similarity ค่าเฉลี่ย 0.8762 คำตอบที่ผ่านเกณฑ์จำนวน 2761 / 3400 คิดเป็น 81.21% การกระจายอยู่ในช่วง 0.8 ถึง 1.0 Euclidean Score ค่าเฉลี่ยอยู่ที่ 0.4516 คำตอบที่ผ่านเกณฑ์จำนวน 1781 / 3400 คิดเป็น 52.38% ค่าหนาแน่นในช่วง 0.4 ถึง 0.7 Manhattan Score ค่าเฉลี่ยอยู่ที่ 9.7489 คำตอบที่ผ่านเกณฑ์จำนวน 2239 / 3400 คิดเป็น 65.85% กระจุกตัวอยู่ในช่วง 8 ถึง 14



รูปที่ 4 ค่าเฉลี่ยความคล้าย Model Typhoon2-8b

Model Typhoon2-8b มีค่า Cosine Similarity ค่าเฉลี่ยอยู่ที่ 0.8770 คำตอบที่ผ่านเกณฑ์จำนวน 2773 / 3,400 รายการ คิดเป็น 81.56% Euclidean Score ค่าเฉลี่ยอยู่ที่ 0.4501 คำตอบที่ผ่านเกณฑ์จำนวน 1,821 / 3,400 รายการ คิดเป็น 53.56% การกระจายอยู่ในช่วง 0.3 ถึง 0.7 Manhattan Score ค่าเฉลี่ยอยู่ที่ 9.7200 คำตอบที่ผ่านเกณฑ์จำนวน 2,249/ 3,400 รายการ คิดเป็น 66.15% การกระจายค่าระยะห่างส่วนใหญ่อยู่ในช่วง 7 ถึง 15

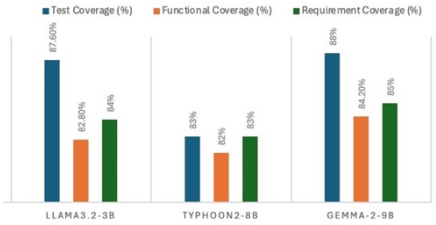


รูปที่ 5 ค่าเฉลี่ยความคล้าย Model Gemma2-9b

Model Gemma-2-9B มีค่า Cosine Similarity ค่าเฉลี่ย 0.8768 คำตอบที่ผ่านเกณฑ์จำนวน 2,813 / 3,400 รายการ คิดเป็น 82.74% การกระจาย 0.8 - 1.0 Euclidean Score ค่าเฉลี่ยอยู่ที่ 0.4511 คำตอบที่ผ่านเกณฑ์จำนวน 1,761 / 3,400 รายการ คิดเป็น 51.79% การกระจายของค่าส่วนใหญ่อยู่ในช่วง 0.4 ถึง 0.7 Manhattan Score ค่าเฉลี่ยอยู่ที่ 9.7364 จำนวน 2,176 / 3,400 รายการ คิดเป็น 63.9% กระจายตัวอยู่ในช่วง 8 ถึง 14

4.4 ผลการวิเคราะห์ความครอบคลุม Coverage Analysis

โดยการวิเคราะห์ Test Coverage, Functional Coverage, Requirement Coverage ทั้งหมด 20 ข้อ เนื้อหาเกี่ยวกับระบบในธนาคารและ ระบบทั่วไป Non-Banking โมเดล Gemma-2-9B มีความครอบคลุมสูงสุดในทุกมิติค่า Test Coverage 88%, Functional Coverage 84.20% และ Requirement Coverage 85% ทำให้เหมาะสมที่สุดสำหรับการใช้งานเชิงปฏิบัติ ขณะที่ LLaMA-3.2-3B มีพารามิเตอร์น้อยกว่า ประสิทธิภาพใกล้เคียงกัน ส่วน Typhoon-2-8B มีความสม่ำเสมอและเสถียรในเชิงผลลัพธ์ ข้อจำกัดในการสร้างกรณีทดสอบ



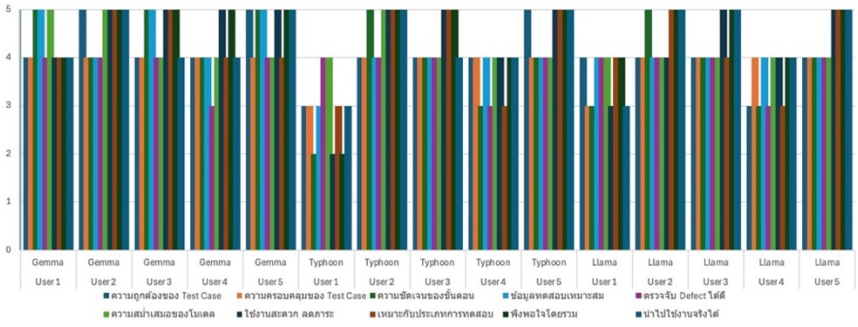
รูปที่ 6 การวิเคราะห์ความครอบคลุม

4.5 การประเมินผลโดยผู้เชี่ยวชาญ Expert Review

ประเมินผลผู้เชี่ยวชาญด้านการทดสอบซอฟต์แวร์ ได้แก่ Software Tester และ QA จำนวน 5 คน ตำแหน่ง Software Tester, QA โดยเกณฑ์ประเมินมาตราส่วน 5 ระดับ

ตารางที่ 4 เกณฑ์การให้คะแนนความพึงพอใจ 5 ระดับ

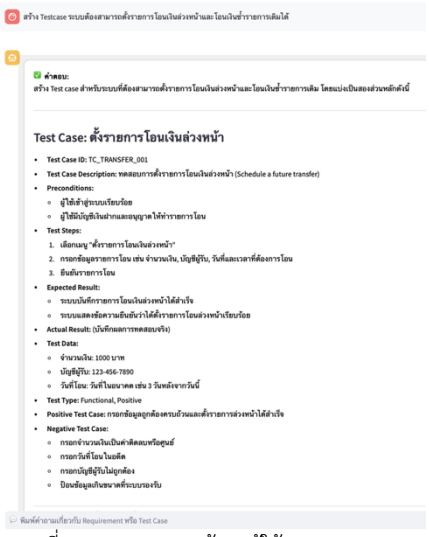
มาตราส่วน 5 ระดับ	คำอธิบาย
5 = พอใจมากที่สุด	ตรงตามเกณฑ์อย่างครบถ้วน ชัดเจน ใช้งานได้จริง
4 = พอใจมาก	ตรงตามเกณฑ์เกือบทั้งหมด มีรายละเอียดดี
3 = พอใจปานกลาง	ตรงตามเกณฑ์พอสมควร แต่มีจุดให้ปรับปรุง
2 = ค่อนข้างไม่พึงพอใจ	ตรงตามเกณฑ์บางส่วน ยังขาดความครบถ้วน
1 = ไม่พึงพอใจมาก	ไม่ตรงตามเกณฑ์ ไม่สามารถใช้งานได้โดยตรง



รูปที่ 7 กราฟแสดงผลการประเมินคุณภาพของ Test Case

4.6 การใช้งานระบบ Chatbot

รูปที่ 8 แสดงตัวอย่างการใช้งานระบบสร้าง Test Case อัตโนมัติด้วยโมเดล LLM ในบริบทของระบบโอนเงินผ่านธนาคารเพื่อจำลองกรณีการใช้งานจริงของผู้ใช้ที่เข้าสู่ระบบด้วยรหัสผ่านอย่างถูกต้อง



รูปที่ 8 การแสดงผลหน้าจอผู้ใช้งาน Chatbot

5. สรุปผลการวิจัย อภิปรายผล และข้อเสนอแนะ

การเปรียบเทียบประสิทธิภาพ โมเดล Typhoon2-8b มีค่า Loss 0.1543 ต่ำที่สุดและ Cosine Similarity 0.8770 สูงสุด Euclidean ต่ำสุด 0.4501 และ Manhattan ต่ำสุด 9.7200 แสดงถึงศักยภาพในการสร้างเวกเตอร์ที่แม่นยำทั้งเชิงความหมายและโครงสร้าง รองลงมาโมเดล Gemma2-9b มีค่า Loss 0.1971 สูงที่สุด Cosine Similarity 0.8768 Euclidean 0.4511 และ Manhattan 9.7364 สามารถสร้างเวกเตอร์ที่มีความสอดคล้องทางบริบทกับข้อกำหนดได้ดีขณะที่โมเดล LLaMA3.2-3B มีค่า Loss 0.1773 Cosine Similarity 0.8762 Euclidean 0.4516 และ Manhattan 9.7489 โมเดลเข้าใจความหมายเชิงบริบทอยู่ในระดับกลาง

การประเมินความครอบคลุม Test Coverage, Functional Coverage และ Requirement Coverage โมเดล Gemma2-9b มีค่าความครอบคลุมสูงสุดในทุกมิติ ได้แก่ Requirement Coverage 85%, Test Coverage 88% และ Functional Coverage 84.2% ศักยภาพในการเข้าใจข้อกำหนดและครอบคลุมฟังก์ชันหลักของระบบได้อย่างมีประสิทธิภาพ LLaMA3.2-3B ให้ค่า Requirement Coverage 84%, Test Coverage 87.6% และ Functional Coverage 82.8% สามารถสร้าง Test Case ที่ครอบคลุมได้หลากหลาย Typhoon2-8b อยู่ในระดับดีโดยมีค่า Requirement Coverage 83%, Test Coverage 83% และ Functional Coverage 82% มีความสม่ำเสมอและเสถียรในเชิงผลลัพธ์ แต่ยังต่ำกว่าโมเดลอื่นเล็กน้อย ข้อจำกัดในการสร้างกรณีทดสอบ

ผลการประเมินจากผู้เชี่ยวชาญโดยโมเดล Gemma2-9b ได้รับคะแนนเฉลี่ยรวมสูงที่สุด (4.42) โดยมีจุดเด่นด้าน ความชัดเจนของขั้นตอนทดสอบผู้ประเมินมองว่าโมเดลสามารถสร้าง Test Case ที่นำไปใช้งานจริงได้อย่างมีประสิทธิภาพมากที่สุด LLaMA3.2-3B ได้คะแนนเฉลี่ย (4.06) โดยมีจุดแข็งในด้าน ความพึงพอใจโดยรวม และความเหมาะสมกับประเภทการทดสอบ แต่ยังมีข้อเสนอแนะให้ปรับปรุงเรื่องการจัดลำดับความสำคัญของ Test Case และการแยกประเภท Test Case ให้ชัดเจน Typhoon2-8b ได้คะแนนเฉลี่ยต่ำสุดที่ (4.00) ด้านความสะดวกในการใช้งาน แต่ผู้ประเมินบางรายพบว่า การประมวลผลช้ากว่าโมเดลอื่น และการจัดลำดับความสำคัญของ Test Case ยังไม่ดีพอ

ข้อเสนอแนะเพื่อเพิ่มศักยด้าน Automated Test Script โดยแปลง Test Case รูปแบบ Script ที่สามารถเชื่อมต่อและใช้งานกับเครื่องมือทดสอบ Selenium, PyTest ขยายการทดลองไปยังโดเมนอื่นและปรับใช้นอกเหนือจาก Banking System ระบบสามารถทำงานแบบครบวงจรตั้งแต่การสร้าง Test Case ไปจนถึงการรันทดสอบอัตโนมัติ

6. กิตติกรรมประกาศ

งานวิจัยนี้สำเร็จลุล่วงด้วยดีจากการได้รับคำแนะนำอบรมสั่งสอนและให้คำปรึกษาจากคณาจารย์ประจำหลักสูตรวิศวกรรมศาสตร มหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ วิทยาลัยวิศวกรรมศาสตร์และเทคโนโลยี มหาวิทยาลัยธุรกิจบัณฑิตย์ ผู้วิจัยขอกราบขอบพระคุณมา ณ โอกาสนี้

เอกสารอ้างอิง

[1] Z. Wang, K. Liu, G. Li and Z. Jin, "HITS: High-coverage LLM-based Unit Test Generation via Method Slicing," 2024.

[2] J. Hoffmann and D. Frister, "Generating Software Tests for Mobile Applications Using Fine-Tuned Large Language Models," 2024.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ‘nearest neighbor’ meaningful?,” in Proc. 7th Int. Conf. Database Theory (ICDT), Jerusalem, Israel, Jan. 1999, pp. 217–235.

[4] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval, Cambridge, U.K.: Cambridge Univ. Press, 2008.


หทัยรัตน์ เจนวิทยา
วศ.บ. วิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยศรีปทุม,
วศ.ม. วิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยธุรกิจบัณฑิตย์


รัฐศิลป์ รานอกภานุวัชร์
วศ.ม.วิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, ปร.ด. วิศวกรรมไฟฟ้า สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง