JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 11, pp. 211-221, November 2026
https://doi.org/10.9708/jksci.2026.31.11.211

JIT Defect Prediction using LLM Semantic Features and Explanation Stability Analysis

Kyoungmin Yoon*, Nammee Moon**

*Ph.D. Candidate, Dept. of Convergence Engineering, Graduate School of Venture, Hoseo University, Seoul, Korea
**Professor, Dept. of Convergence Engineering, Graduate School of Venture, Hoseo University, Seoul, Korea

[Abstract]

Just-In-Time (JIT) defect prediction has relied on quantitative features that do not capture the semantics of code changes. This study uses a Large Language Model (LLM) to extract five interpretable semantic features from commit diffs and messages and combines them with the 14 Kamei features in Logistic Regression (LR) and Random Forest (RF). In stratified 10-fold cross-validation on 4,000 OpenStack commits, the LLM features improved AUC, the primary metric, for both models; the gain remained significant after correction over the two AUC tests (but not over all eight tests) and was consistent across 10 repeated splits. In a time-ordered evaluation, the gains remained positive (LR +0.0088, RF +0.0166), but their confidence intervals included zero. SHAP analysis showed that structural risk ranked highly in RF importance although removing it did not lower performance, and bootstrap resampling kept the importance ranks stable (mean Spearman ρ of 0.80–0.94).

▸ **Key words**: Just-In-Time Defect Prediction, Large Language Model, Explainable AI, SHAP, Software Defect Prediction

[요 약]

정량적 특징 중심의 JIT 결함 예측에 변경의 의미를 반영하기 위해 LLM으로 커밋 diff와 메시지에서 5개 시맨틱 특징을 추출하여 Kamei 14개 특징과 결합하고, 로지스틱 회귀(LR)와 랜덤 포레스트(RF)의 성능을 비교하였다. OpenStack 커밋 4,000건의 층화 교차검증에서 주 지표인 AUC는 2개 검정에 대한 다중비교 보정 후 두 모델 모두 유의하게 향상되었고(8개 검정 전체 보정 시 비유의) 10회 반복에서도 일관되었으나, 시간순 검증에서는 향상 방향만 유지되고 신뢰구간이 0을 포함하였다. SHAP에서 RF의 구조적 위험도는 중요도가 높았으나 제거해도 성능이 낮아지지 않았고, 부트스트랩 재추출 시 순위의 Spearman 상관은 평균 0.80~0.94였다.

▸ **주제어**: Just-In-Time 결함 예측, 대규모 언어모델, 설명가능한 인공지능, SHAP, 소프트웨어 결함 예측

• First Author: Kyoungmin Yoon, Corresponding Author: Nammee Moon
*Kyoungmin Yoon (kmin3390@gmail.com), Dept. of Convergence Engineering, Graduate School of Venture, Hoseo University
**Nammee Moon (nammee.moon@hoseo.ac.kr), Dept. of Convergence Engineering, Graduate School of Venture, Hoseo University
• Received: 2026. 10. 12, Revised: 2026. 11. 08, Accepted: 2026. 11. 12.

Copyright © 2026 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

I. Introduction

소프트웨어 결함은 품질과 비용에 직접 영향을 미치므로 결함 가능성이 높은 부분을 사전에 식별하여 품질 보증 자원을 배분하는 것이 중요하며, 최근에는 딥러닝을 결합한 결함 예측이 제안되고 있다[1].

전통적인 결함 예측은 파일이나 모듈 단위로 이루어져 릴리즈 시점에야 평가할 수 있고, 어떤 변경이 문제였는지 특정하지 못한다. 이에 대한 대안인 Just-In-Time 결함 예측(JIT-SDP)은 커밋(코드 변경) 단위로 결함 유발 가능성을 예측하여 변경 직후에 즉각적인 피드백을 제공할 수 있으며[2], Kamei 등의 14개 커밋 특징이 표준 baseline으로 사용된다.

그러나 이러한 전통적 특징은 변경의 규모나 이력 같은 정량적 정보에 국한되어 변경의 의미는 반영하지 못한다. 이를 보완하기 위해 Hoang 등[3]과 Pornprasit 등[4]은 commit message와 diff에서 딥러닝으로 시맨틱 특징을 추출하였고, 최근에는 대규모 언어모델(LLM)을 파인튜닝된 분류기[5]나 프롬프트 기반 직접 예측기[6]로 사용하는 연구가 이어지고 있다. 이들은 결함 라벨 기반 파인튜닝과 GPU 학습이 필요하거나 전통적 특징과의 결합 방식이 명확하지 않다. 반면 LLM을 특징 추출기로 사용하면 결함 라벨 없이 점수를 산출할 수 있어 추론 비용 외에는 경량 분류기의 재학습만 필요하며, 점수가 사람이 읽을 수 있는 형태로 남아 Kamei 특징과 함께 SHAP(SHapley Additive exPlanations)으로 해석할 수 있다. LLM을 점수 산출에만 사용하므로 같은 LLM으로 결함 위험을 직접 예측하는 zero-shot 방식과 달리 프로젝트의 결함 이력을 분류기 학습에 반영할 수 있다.

또한 결함 예측 모델의 실무 적용을 위해서는 예측의 근거를 설명할 수 있어야 한다. 설명가능한 AI(XAI)는 다른 도메인에서도 적용 사례가 보고되었고[7], LLM 기반 위험도 예측에서는 attention weight를 이용한 설명이 시도되었으나[8], 그 설명이 예측 성능과 일관되는지에 대한 검증은 부족하다.

이에 본 연구는 다음 두 가지 연구 질문(Research Question, 이하 RQ)을 설정한다.

RQ1: 전통적 JIT 특징(Kamei 14개)에 LLM 기반 시맨틱 특징을 추가하면 결함 예측 성능이 유의하게 향상되는가?

RQ2: SHAP 분석에서 LLM 시맨틱 특징이 예측에 실질적으로 기여하는가, 그리고 그 설명은 안정적인가?

이를 위해 본 연구는 LLM을 경량 특징 추출기로 한정하여, 구조화된 출력으로 해석 가능한 5개의 시맨틱 특징을 추출하고 전통적 특징과 결합한 로지스틱 회귀(Logistic Regression, LR) 및 랜덤 포레스트(Random Forest, RF) 모델의 성능과 SHAP 기반 설명의 안정성을 분석한다. 또한 결합 효과가 특정 LLM에 국한되는지 확인하기 위해 같은 제공사의 크기가 다른 두 모델(Claude Sonnet 5, Claude Haiku 5.5)로 같은 절차를 반복한다.

본 연구의 주요 기여는 다음과 같다. 첫째, LLM을 경량 특징 추출기로 활용하여 파인튜닝 없이 전통적 특징과 결합하는 구성을 제시하고, 그 결합 효과를 무작위 교차검증과 시간순 검증으로 함께 검토하였으며, 크기가 다른 두 LLM으로 반복하고 각 LLM의 zero-shot 직접 예측과도 비교하였다. 둘째, 결합의 효과가 모델 유형(선형 대 트리 기반)에 따라 다르게 나타남을 보였다. 셋째, SHAP 중요도와 개별 특징 제거(ablation) 기여도의 불일치를 확인하고 상관된 특징 간 기여 분배 관점에서 그 원인을 분석하였으며, 설명 순위의 안정성과 그 해석 범위를 함께 검토하였다. 다만 기존 시맨틱 기반 JIT 결함 예측 방법과의 동일 조건 비교는 포함하지 않으므로, 본 연구의 기여는 기존 특징과 LLM 시맨틱 특징의 결합 효과와 설명 분석에 한정된다.

II. Related Work

1. Traditional JIT Defect Prediction

Just-In-Time 결함 예측(JIT-SDP)은 커밋 단위로 결함 유발 가능성을 예측하는 접근으로, Kamei 등[2]이 제안한 14개의 커밋 단위 특징(변경 규모, 개발자 경험, 코드 이력 등)이 표준 baseline으로 사용되어 왔다. McIntosh와 Kamei[9]는 결함 유발 변경의 특성이 시간에 따라 달라질 수 있음을 보였다. 이후 Hoang 등[3]은 commit message와 diff에서 특징을 자동 추출하는 DeepJIT를 제안하였고, Pornprasit 등[4]은 CC2Vec의 테스트셋 유출 문제를 지적하며 이를 배제한 JITLine을 제안하였다. Zeng 등[10]은 단순한 특징 기반 모델이 딥러닝 기반 JIT 모델과 비슷하거나 더 나은 성능을 보일 수 있음을 보고하였다.

2. LLM-based JIT Defect Prediction

최근 LLM을 JIT-SDP에 적용하는 연구가 활발히 이루어지고 있다. DRS-OSS[5]는 Llama 3.1 8B 분류기로 변경 위험 점수를 산출하는 오픈소스 도구이며, PROPER-SDP[6]는 파인튜닝 없이 프로젝트 진화 정보를 프롬프트에 포함시켜 cold-start 문제를 다룬다.

www.kci.go.kr

Monteiro 등[11]은 파인튜닝된 소·중형 모델(CodeT5+, UniXCoder)이 closed LLM의 zero-shot 프롬프팅보다 우수함을 보였고, 이를 점진적(incremental) JIT 예측으로 확장한 연구도 있다[12].

ReDef[13]은 SZZ 기반 라벨링의 노이즈를 지적하며 리버트 커밋 기반 벤치마크를 제안하였다. 본 연구가 사용한 데이터셋도 SZZ 기반이므로 이 한계를 결론에서 언급한다.

3. Combining Semantic Features and Explainability

CommitGuru의 전문가 특징과 CodeBERT 기반 시맨틱 특징을 결합한 JIT-Fine[14]는 결함 예측과 위치 예측을 동시에 수행하였으나, CodeBERT 파인튜닝과 별도의 라인 단위 라벨링 데이터셋이 필요하다는 점에서 본 연구와 다르다. LLM이 생성한 정보를 기존 머신러닝 모델의 입력 특징으로 활용하는 접근은 테스트 우선순위화에서도 개선 가능성이 보고되었다[15]. 설명가능성 측면에서 Liu 등[8]은 LLM 기반 위험도 예측 모델의 attention weight로 diff 내 위험 부분을 강조하여 제시하였으나, 본 연구는 SHAP으로 전통 특징과 LLM 특징을 함께 설명한다는 점에서 다르다.

4. Defect Prediction Research

국외의 LLM 활용 연구[5][6][8][11]는 LLM을 분류기나 예측기로 직접 사용하거나 attention으로 설명하는 방식이었으며, 파인튜닝 없는 LLM 시맨틱 특징을 전통 특징과 결합하고 SHAP 설명의 안정성까지 검증한 연구는 확인되지 않았다.

국내에서는 홍의석[1]이 MLP 기반 결함 예측 모델로 AUC 향상을 보였으나 파일·모듈 단위의 예측이며 커밋 단위(JIT) 예측이나 LLM 시맨틱 정보는 다루지 않는다. 최지원 등[16]은 GAN 변형 모델을 비교하여 CTGAN이 결함 예측 데이터 생성에 가장 적합함을 보였으며, 본 연구는 데이터 증강 대신 class_weight='balanced' 옵션으로 불균형에 대응하였다.

XAI 적용 사례로는 남은수 등[7]이 직물 결함 검출에 Grad-CAM을 적용하여 모델이 결함으로 인식한 영역이 실제 결함 위치와 일치함을 확인하였다. 이는 도메인은 다르나 설명가능성의 실증 검증이라는 점에서 본 연구와 문제의식을 공유한다.

5. Distinction of This Study

본 연구는 LLM을 분류기[5]나 직접 예측기[6]가 아닌 경량 특징 추출기로 한정하여 GPU 자원이나 파인튜닝 없이 추론 API만으로 전통 특징과 결합하고, 그 효과의 모델 유형별 차이와 SHAP 중요도-ablation 불일치의 원인을 분석한다는 점에서 기존 연구와 접근 방식이 다르다.

III. Method

1. Dataset

본 연구는 Zeng 등[10]이 재현 패키지로 공개한 OpenStack 프로젝트의 JIT 결함 예측용 데이터셋을 사용하였다. 원본은 22,757건의 커밋으로 구성되며 이 중 결함 유발 커밋은 5,927건(26.04%)이다. 계산 비용과 연구 기간을 고려하여 결함 비율을 유지하는 층화추출(seed=42)로 4,000건을 추출하였고, 최종 표본의 결함 비율은 26.05%로 원본과 거의 같았다. 추출된 데이터셋의 diff는 파일 경로와 변경 방향(+/-) 정보 없이 변경 라인 목록으로만 제공되었다. 이에 커밋 해시로 해당 커밋이 속한 OpenStack 하위 프로젝트(nova, neutron, cinder, swift, glance)를 특정하고 git show(--unified=3)로 파일 경로와 변경 방향을 포함한 unified diff를 4,000건 전체에서 재구성하였다. 재구성한 diff에서 계산한 코드 지표는 데이터셋의 Kamei 값과 일부 일치하지 않았다(변경 파일 수 92.5% 일치, Ⅳ장 3절).

2. Traditional Commit Features

Baseline 특징으로는 Kamei 등[2]이 제안한 14개의 커밋 단위 특징을 사용하였다: 수정된 서브시스템·디렉터리·파일 수(NS, ND, NF), 변경 분산도(Entropy), 추가·삭제된 라인 수(LA, LD), 커밋 이전 파일의 평균 라인 수(LT), 버그 수정 여부(FIX), 수정 파일들의 개발자 수(NDEV), 최종 수정 이후 경과 시간(AGE), 수정 파일들의 과거 변경 횟수(NUC), 개발자의 누적·최근·서브시스템 경험치(EXP, REXP, SEXP). FIX를 제외한 13개 수치형 특징에는 값의 왜도를 완화하기 위해 log1p 변환을 적용하였다. Kamei 특징은 이 데이터셋의 값을 그대로 사용하였으며, 경험(EXP, REXP, SEXP)과 이력(NDEV, AGE, NUC) 특징은 정의상 해당 커밋 이전의 이력으로 산출되나[2], 그 산출 과정을 직접 재현하지는 않았다.

3. LLM-based Semantic Feature Extraction

본 연구는 Anthropic Claude Sonnet 5(API 모델 식별자 claude-sonnet-5)로 각 커밋의 diff와 commit message에서 5개의 시맨틱 특징을 추출하였다.

www.kci.go.kr

Anthropic Message Batches API로 호출하고 구조화 출력 기능(output_config.format, type json_schema)에 Table 1의 JSON Schema를 지정하여 해석 가능한 점수·범주를 산출하였다. 생성 설정은 최대 출력 토큰 512, 확장 사고(extended thinking) 비활성화이며, temperature 등 샘플링 매개변수는 지정하지 않아 API 기본값(temperature 1.0)으로 생성하였다. 추출은 2026년 9월에 수행하였으며, 4,000건에 Batches API 비용 약 17.32달러(커밋 1건당 약 0.004달러)와 약 6시간의 배치 처리 시간이 소요되었다. 모델 규모에 따른 차이를 확인하기 위해 같은 제공사의 소형 모델인 Claude Haiku 5.5(claude-haiku-5-5)로 프롬프트, JSON Schema, 입력, 생성 설정을 모두 같게 두고 4,000건을 다시 추출하였다(2026년 11월, Batches API 비용 약 0.85달러, 배치 처리 약 3.5시간, 4,000건 모두 성공). Anthropic은 모델의 매개변수 수를 공개하지 않으므로 모델 규모는 같은 제공사의 모델 등급으로 구분하였다. 입력은 데이터셋의 commit message(소문자화·토큰화된 전처리 형태)와 각 커밋의 git show 출력(커밋 해시, 작성자, 작성 시각, 원본 커밋 메시지로 이루어진 헤더와 unified diff; Table 1)이며, 앞에서부터 최대 20,000자까지 사용하였다. 따라서 커밋 메시지는 두 형태로 중복 입력되었고 20,000자를 초과하는 369건(9.2%)은 diff의 뒷부분이 절단되었다. 파일럿, sanity check, 본 추출, 재현성 검증에는 모두 동일한 입력 구성을 사용하였다. Table 1에는 실제 사용한 시스템 프롬프트 전문(줄바꿈만 생략)과 출력 JSON Schema 원문, 사용자 입력 형식을 그대로 제시하였다.

추출된 5개 특징은 전통적 수치 특징이 직접 포착하지 못하는 측면을 보완하도록 다음과 같이 설계하였다: (1) 버그 수정 의도(bug_fix_intent_score, BFIS; 키워드 매칭을 넘어선 메시지의 의도), (2) 구조적 위험도(structural_risk_score, SRS; 변경 규모가 아닌 변경의 질), (3) 커밋 메시지-diff 일치도(message_diff_alignment_score, MDAS; 기술된 의도와 실제 변경의 정합성), (4) 변경 범위 범주(change_scope_category, CSC; 변경의 유형, 8개 범주의 원-핫 인코딩), (5) 복잡도 자기평가(complexity_self_assessment_score, CSAS; LT·LA·LD가 간접적으로만 암시하는 복잡도). (4)를 제외한 특징은 1~5점이며, 4,000건 전체에서 추출에 성공하였다.

프롬프트의 타당성은 30건의 파일럿 테스트로 출력 형식과 점수 분포를 확인하고, 무관한 커밋의 commit message와 diff를 뒤섞은 가상 사례 3건(sanity check)에서 메시지-diff 일치도가 1~2점으로 낮게 산출됨을 확인하는 방식으로 검토하였다. 이는 명백한 불일치에 대한 반응을 확인한 것이며, 실제 데이터에서 관측된 4점대 후반의 점수가 OpenStack의 커밋 메시지 품질을 반영하는지는 3건의 sanity check로 확인할 수 없다.

LLM 응답의 재현성(test-retest reliability)은 파일럿 표본 30건을 동일한 프롬프트와 설정으로 재추출하여, 특징별 완전 일치율, 평균 절대 차이, Cohen's kappa(점수형은 선형 가중, 범주형은 비가중)로 측정하였다.

Table 1. Prompt design for LLM semantic feature extraction.

Component	Content (verbatim system prompt, JSON Schema, and user input format; line breaks in the prompt omitted)
Role / Constraint	You are a senior software engineer analyzing a single git commit (diff + commit message) for a just-in-time software defect prediction study. Score the commit on 5 structured dimensions. Base every judgment ONLY on the diff and message given - do not assume anything about code you cannot see. Scoring rubrics (1-5 integer scales unless noted):
BFIS (1-5)	bug_fix_intent_score: How clearly does this commit intend to fix a bug? 1 = clearly unrelated to bug fixing (new feature, pure refactor, docs), 5 = explicit bug/crash/exception/incorrect-behavior fix
SRS (1-5)	structural_risk_score: How much does this change touch core control flow, error handling, concurrency, or public API contracts (vs. cosmetic/isolated changes)? 1 = cosmetic or fully isolated (comments, logging, constants, docs, tests only), 5 = directly rewrites core control flow / error handling / concurrency / a public interface, with broad blast radius
MDAS (1-5)	message_diff_alignment_score: How accurately and specifically does the commit message describe what the diff actually does? 1 = message is missing, generic, or contradicts the diff, 5 = message precisely and specifically describes the diff's intent and scope
CSC (8 categories)	change_scope_category: the single best-fitting category for this change.
CSAS (1-5)	complexity_self_assessment_score: how complex is this change to review and understand? 1 = trivial (one or two obvious lines), 5 = highly complex (intertwined logic across multiple files/functions)
Output	Output only the JSON object matching the schema. No extra commentary. JSON Schema (output_config.format, type json_schema): {"type": "object", "properties": {"bug_fix_intent_score": {"type": "integer", "enum": [1, 2, 3, 4, 5]}, "structural_risk_score": {"type": "integer", "enum": [1, 2, 3, 4, 5]}, "message_diff_alignment_score": {"type": "integer", "enum": [1, 2, 3, 4, 5]}, "change_scope_category": {"type": "string", "enum": ["bug_fix", "feature_addition", "refactoring", "hotfix", "test", "documentation", "configuration", "other"]}, "complexity_self_assessment_score": {"type": "integer", "enum": [1, 2, 3, 4, 5]}}, "required": ["bug_fix_intent_score", "structural_risk_score", "message_diff_alignment_score", "change_scope_category", "complexity_self_assessment_score"], "additionalProperties": false}
User input	"Commit message:\n{message}\n\nDiff:\n{diff}", where {diff} is the full git show output (commit hash, author, date, original commit message, and unified diff), truncated after the first 20,000 characters with "\n... [diff truncated for length] ..." appended

www.kci.go.kr

4. Prediction Models and Evaluation

Baseline(Kamei 14개 특징)과 proposed(Kamei 14개 + LLM 시맨틱 5개, CSC의 원-핫 인코딩을 포함한 입력 26열) 두 특징 집합 각각에 대해 LR과 RF(n_estimators=300)를 적용하였다. 클래스 불균형을 고려하여 두 모델 모두 class_weight='balanced' 옵션을 적용하였다.

성능 평가는 층화(stratified) 10-fold 교차검증(seed=42)으로 수행하였으며, baseline과 proposed 간 대응비교를 위해 동일한 fold 분할을 사용하였다. 평가지표는 AUC, F1-score, Recall(threshold=0.5)과 effort-aware 지표인 PofB20(변경 라인 수(LA+LD) 기준 상위 20% effort로 회수 가능한 결함 비율)이며, 성능 차이의 유의성은 폴드별 대응표본 Wilcoxon signed-rank test로 검정하였다(α=0.05). AUC를 주 지표(primary metric), F1·Recall·PofB20을 보조 지표(secondary metric)로 두고, 주 지표에는 Bonferroni 및 Benjamini-Hochberg(BH) 절차로 다중비교를 보정하였다.

검증 방법의 선택은 결함 예측 모델의 성능 추정에 영향을 줄 수 있으므로[17], 다음 두 가지 검증을 추가로 수행하였다. 첫째, 단일 fold 분할에 따른 우연을 확인하기 위해 분할 시드를 바꾸어 층화 10-fold 교차검증을 10회 반복하였다. 각 반복에서 baseline과 proposed는 동일한 fold 분할을 사용하였으며, 반복별 fold 평균 차이 10개로부터 t 분포 기반 95% 신뢰구간(CI)을 산출하였다.

둘째, 무작위 교차검증은 미래 커밋으로 학습한 모델이 과거 커밋을 예측하는 경우를 포함하므로, 커밋 작성 시각 기준으로 정렬한 시간순 검증을 수행하였다. 가장 이른 80%(3,200건)로 학습하고 가장 늦은 20%(800건)로 평가하는 시간순 holdout과, 과거 구간 전체로 학습하고 바로 다음 구간을 평가하는 확장 윈도우(expanding window) 방식(4개 구간)을 적용하였다. 테스트 구간에서 baseline과 proposed에 동일한 bootstrap 재표본(1,000회)을 적용하는 paired bootstrap으로 지표(AUC, PR-AUC(정밀도-재현율 곡선 아래 면적), F1, PofB20) 차이의 95% CI(percentile)를 산출하였으며, 이 구간은 모델 재학습에 따른 변동은 포함하지 않고 테스트 표본의 변동만을 반영한다.

LLM을 예측기로 직접 사용하는 방식과 비교하기 위해, 특징 추출과 같은 모델·설정·사용자 입력으로 zero-shot 직접 예측을 수행하였다. 시스템 프롬프트는 주어진 commit message와 diff만을 근거로 결함 유발 위험을 0부터 100까지의 정수 하나로 출력하도록 하였으며 결함 라벨 정보는 포함하지 않았다. 최대 출력 토큰 16으로 점수가 잘린 15건(1.9%)은 최대 출력 토큰만 1,024로 늘려 다시 호출하였다. 평가는 시간순 holdout의 테스트 800건에서 수행하였으며, 지도학습 모델과의 비교는 최초 호출에서 점수를 얻은 785건을 공통 집합으로 하여 동일한 bootstrap 재표본(1,000회)으로 95% 신뢰구간을 산출하였다. Haiku 5.5에도 같은 평가 절차와 zero-shot 직접 예측을 적용하였다.

5. SHAP-based Explainability Analysis

전체 표본을 층화 방식으로 80:20 비율(seed=42)로 분할하여 학습 데이터(3,200건)로 모델을 재학습하고 테스트 데이터(800건)에 대해 SHAP[18] 분석을 수행하였다. RF에는 TreeExplainer를, LR에는 LinearExplainer(표준화된 입력 기준)를 적용하여 평균 절대 SHAP 값(mean |SHAP|) 기준으로 특징 중요도 순위를 산출하였다. 변경 범위 범주(CSC)는 원-핫 인코딩된 8개 열의 SHAP 값을 샘플별로 합산하여 하나의 특징으로 집계하였으므로, 순위는 Kamei 14개와 LLM 시맨틱 5개, 총 19개 특징 단위로 산출하였다. 단일 분할의 영향은 80:20 분할을 10개 시드(42~51)로 반복하여 순위의 평균·범위와 분할 간 Spearman 상관으로 확인하였다.

설명의 안정성은 학습 데이터에 대한 부트스트랩 재추출 10회로 모델을 재학습하고, 매 회 산출된 테스트셋 기준 특징 중요도 순위 간 Spearman 순위상관계수(총 45개 쌍)를 계산하여 검증하였다. 이는 학습 데이터가 달라져도 중요도 순위가 유지되는지를 확인하는 안정성(stability) 검증이며, 설명의 충실도(fidelity)나 결함 원인으로서의 타당성은 본 분석의 검증 범위에 포함되지 않는다. 모든 분석은 Python 3.9.6 환경에서 scikit-learn 1.6.1, scipy 1.13.1, shap 0.49.1, anthropic Python SDK 0.125.0을 사용하여 수행하였다.

IV. Results

1. RQ1: Performance Improvement from Adding LLM Semantic Features

1.1 Performance Comparison

Baseline과 proposed의 10-fold 교차검증 결과는 Table 2와 같다(Sig.: Y = 유의($p<0.05$), N = 비유의).

www.kci.go.kr

Table 2. Performance comparison between Baseline and Proposed.

Model	Metric	Baseline	Proposed	Diff	p-value	Sig.
LR	AUC	0.7784	0.7887	+0.0103	0.0137	Y
LR	F1	0.5518	0.5677	+0.0159	0.0488	Y
LR	Recall	0.7274	0.7265	-0.0010	0.8880	N
LR	PofB20	0.1612	0.1975	+0.0364	0.0371	Y
RF	AUC	0.7827	0.7899	+0.0072	0.0137	Y
RF	F1	0.4481	0.4463	-0.0018	1.0000	N
RF	Recall	0.3455	0.3407	-0.0048	0.7247	N
RF	PofB20	0.2715	0.2523	-0.0192	0.0625	N

LR에서는 AUC, F1, PofB20 3개 지표에서 proposed가 baseline 대비 유의한 향상을 보였다(모두 $p<0.05$). RF에서는 AUC만 유의한 향상을 보였고(+0.0072, $p=0.0137$), F1과 Recall은 유의한 차이가 없었으며 PofB20은 오히려 감소하였다(-0.0192, $p=0.0625$). 주 지표인 AUC는 Bonferroni 및 BH 보정 후에도 두 모델 모두 유의성이 유지되었다(Bonferroni $p=0.0274$, BH $q=0.0137$). 보조 지표의 p-value는 보정하지 않은 값이며, 8개 검정 전체를 하나의 family로 보정하면 AUC의 보정 p-value는 두 모델 모두 Bonferroni 0.1096, BH 0.0548로 유의수준(0.05)을 초과한다. 향상 폭(0.007~0.010)도 작으므로 통계적 유의성이 실무적 개선을 의미하지는 않는다.

1.2 Repeated Cross-Validation and Time-Ordered Evaluation

분할을 10회 반복한 층화 10-fold 교차검증에서 AUC는 LR(0.7787→0.7892)과 RF(0.7809→0.7916) 모두 10회 전부에서 proposed가 높았고, 평균 차이는 LR +0.0105(95% CI [+0.0101, +0.0109]), RF +0.0107(95% CI [+0.0092, +0.0122])로 Table 2의 단일 시드 결과와 같은 방향이어서 AUC 향상이 특정 분할의 우연이라고 보기는 어렵다.

보조 지표는 모델에 따라 상이하였다. LR의 F1(+0.0175)과 PofB20(0.1432→0.1828, +0.0396, 95% CI [+0.0349, +0.0443])은 10회 모두 향상되었으나, RF는 F1의 차이가 없었고(-0.0029) PofB20은 10회 모두 감소하였다(0.2584→0.2346, -0.0238, 95% CI [-0.0322, -0.0154]). 따라서 effort 기반으로 검토 우선순위를 정하는 목적에서는 RF에 LLM 특징을 추가하는 이점이 확인되지 않았다.

커밋 시각 순서를 유지한 시간순 holdout에서 proposed의 AUC는 LR 0.7757→0.7845(+0.0088, 95% CI [-0.0056, +0.0242]), RF 0.7385→0.7551(+0.0166, 95% CI [-0.0015, +0.0339])로 무작위 교차검증과 같은 방향이었으나 두 모델 모두 신뢰구간이 0을 포함하여 유의한 차이로 확인되지는 않았다. 확장 윈도우 4개 구간에서도 AUC 차이는 -0.0005~+0.0166으로 대부분 양의 방향이었으나 모든 구간에서 신뢰구간이 0을 포함하였다. PR-AUC와 F1도 신뢰구간이 0을 포함하였고, RF의 PofB20은 0.2237→0.1776(-0.046, 95% CI [-0.099, +0.022])으로 감소 방향을 보였다.

baseline의 AUC는 무작위 교차검증 대비 시간순 holdout에서 LR은 0.778에서 0.776으로 거의 변하지 않았으나 RF는 0.783에서 0.739로 낮아져, 무작위 교차검증이 트리 기반 모델의 성능을 과대평가할 수 있음을 시사한다. 시간순 테스트 구간의 결함 비율(19%)이 전체(26.05%)보다 낮아 F1, Recall, PofB20은 무작위 교차검증과 직접 비교하기 어렵다. 반복 교차검증의 구간은 분할 안정성을, 시간순 검증의 구간은 이후 커밋에 대한 일반화를 나타내므로 두 결과는 모순되지 않는다.

1.3 Post-hoc Analysis: Feature Relationships and Single-Feature Ablation

RQ1 결과가 모델에 따라 다른 이유를 탐색하기 위해 세 가지 사후 분석을 수행하였다. 일부 LLM 특징이 Kamei 특징의 변경 규모·확산 정보를 재제공하여, RF는 이를 트리 분기로 자체 포착하고 LR은 LLM의 요약 정보에서 더 큰 이득을 얻었을 가능성을 고려하였다. 이들은 탐색적 분석이며 다중비교 보정 대상에 포함하지 않았고, 최종 proposed 집합은 사전에 설계한 5개 특징을 모두 사용하였다.

(1)·(2) 상관관계 및 개별 특징 제거(ablation) 분석. LLM 시맨틱 특징 중 구조적 위험도와 복잡도 자기평가는 크기·확산 관련 Kamei 특징과 중간~강한 상관($r=0.3$~0.7)을, 버그 수정 의도는 FIX와 강한 상관($r=0.691$)을 보인 반면 메시지-diff 일치도는 모든 Kamei 특징과 $|r|<0.15$로 낮았다. 또한 RF에서 LLM 특징을 하나씩 제거하면 대부분 F1·Recall이 오히려 상승하였고, 특히 구조적 위험도를 제거했을 때 AUC(0.7929)가 5개 특징을 모두 포함한 proposed 모델(0.7899)보다 높았다.

(3) 정제된 특징 집합(pruned feature set) 재검증. (1)·(2)를 근거로 상관이 높은 3개 특징(버그 수정 의도, 구조적 위험도, 복잡도 자기평가)을 제외하고 2개 특징(메시지-diff 일치도, 변경 범위)만 유지하여 재검증한 결과, LR에서는 향상 폭이 줄고(AUC +0.0103→+0.0057, F1 +0.0159→+0.0109) PofB20이 유의성을 잃었다($p=0.037$→0.058). RF에서는 AUC 향상이 유의성을 잃고($p=0.014$→0.193) PofB20은 오히려 유의하게 악화되었다($p=0.037$). 이 정제 집합은 같은 데이터에서 사후에 설정한 것이므로 채택하지 않았다. 이 결과는 특징 선택을 상관도나 단일

www.kci.go.kr

제거 효과만으로 판단하기 어려움을 시사한다.

1.4 Comparison with Direct LLM Prediction

시간순 holdout 테스트 800건에서 Sonnet 5의 zero-shot 직접 예측 AUC는 0.7231(95% CI [0.681, 0.767]), PR-AUC는 0.3463이었다. 지도학습 모델과 같은 785건에서 비교하면 zero-shot의 AUC(0.7242)는 LLM 특징을 추가한 LR(0.7866)뿐 아니라 Kamei 특징만 사용한 LR(0.7783)보다도 약 0.05~0.06 낮았고, RF(baseline 0.7391, proposed 0.7577)와는 신뢰구간이 크게 겹쳐 차이를 구분할 수 없었다. 이는 본 설정에서 LLM의 판단을 점수형 특징으로 결합하는 구성이 직접 예측보다 성능이 낮지 않음을 시사한다. 다만 이는 단일 프롬프트의 zero-shot과 비교한 결과이며 기존 JIT 결함 예측 기법과의 비교가 아니다.

1.5 Comparison by Extraction Model Size

Haiku 5.5로 추출한 특징을 추가하면 층화 10-fold 교차검증에서 AUC가 LR 0.7784→0.7903(+0.0119, p=0.027), RF 0.7827→0.7949(+0.0122, p=0.002)로 향상되어 Sonnet 5(LR +0.0103, RF +0.0072)와 같은 방향이었다. AUC 2개 검정의 BH 보정 후에는 두 모델 모두 유의하였으나 Bonferroni 보정 후에는 RF만 유의하였다(LR p=0.054). 10회 반복 교차검증에서는 두 모델 모두 10회 전부 proposed의 AUC가 높았다(LR +0.0132, RF +0.0117). PofB20은 Sonnet 5와 같이 LR에서 향상되고(10-fold +0.0623, p=0.012) RF에서 감소하였으며(-0.0202, p=0.065), 반복 교차검증에서도 LR은 10회 모두 높고 RF는 10회 모두 낮았다. 시간순 holdout의 AUC 차이는 LR +0.0183(95% CI [+0.0031, +0.0329]), RF +0.0197(95% CI [+0.0046, +0.0354])로 신뢰구간이 0을 포함하지 않았으나, 확장 윈도우 4개 구간 중에서는 셋째 구간의 LR과 넷째 구간(시간순 holdout과 같은 분할)에서만 0을 포함하지 않았다. 두 추출 모델 간의 성능 차이는 별도로 검정하지 않았다. 두 모델이 산출한 점수의 일치도(4,000건, kappa)는 BFIS 0.838, SRS 0.788, CSAS 0.813(이상 선형 가중), CSC 0.873(비가중)이었으나 MDAS는 0.335로 낮았다. 같은 모델의 zero-shot 직접 예측에서는 800건 중 66건(8.3%)을 다시 호출하였고 1건은 점수를 얻지 못하였으며, AUC는 0.7339(799건)였다. 최초 호출에서 점수를 얻은 734건에서 비교하면 zero-shot의 AUC(0.7387)는 baseline LR(0.7876)과 Haiku 특징을 추가한 LR(0.7992)보다 낮았고, RF(baseline 0.7410, proposed 0.7592)와는 0.002~0.021 차이였다.

2. RQ2: SHAP-based Feature Importance and Explanation Stability

2.1 Feature Importance and Explanation Stability

RQ1에서 LLM 시맨틱 특징의 효과가 모델에 따라 달랐으므로, 모델이 이 특징들을 실제로 얼마나 참조하는지 SHAP으로 분석하였다. 산출 절차는 Ⅲ장 5절, 특징 약어는 Ⅲ장 2, 3절과 같다(Fig. 1, Fig. 2).

19개 특징 기준으로 상위 3개 특징은 RF에서 LA, CSAS, AGE, LR에서 NUC, LA, NDEV였다. LLM 특징 중 복잡도 자기평가(CSAS)는 RF·LR 모두에서 상위권(각각 2위, 6위)이었고, 구조적 위험도(SRS)는 RF에서 5위, LR에서 14위, 버그 수정 의도(BFIS)는 LR에서 8위, RF에서 17위, 변경 범위(CSC)는 원-핫 열을 합산하면 LR에서 10위, RF에서 13위였으며, 메시지-diff 일치도(MDAS)는 LR에서 18위, RF에서 19위로 최하위권이었다. 분할을 10회 바꾸어도 이 경향은 유지되어, CSAS는 RF에서 10회 모두 2위, MDAS는 RF에서 10회 모두 19위였다.

한편 RF에서 SRS의 SHAP dependence plot은 점수가 1점에서 5점으로 높아질수록 SHAP 값이 단계마다 증가하였으며(Fig. 3), LR에서 BFIS는 점수가 높을수록 SHAP 값이 감소하였다(Fig. 4).

부트스트랩 10회 재학습에 따른 순위 간 Spearman 상관(45쌍)은 LR 평균 0.799(최소 0.623, 최대 0.939), RF 평균 0.939(최소 0.893, 최대 0.982)로 두 모델 모두 순위가 일관되게 유지되었고 RF가 더 안정적이었다. 분할 시드를 10개로 바꾼 순위 간 Spearman 상관도 두 모델 모두 평균 0.932(LR 0.883~0.974, RF 0.860~0.986)로 단일 분할에 따른 변동은 크지 않았으나, 산출 방식이 달라 부트스트랩 값과 같은 척도로 비교할 수는 없다.

LLM 특징 추출의 재현성은 파일럿 표본 30건의 재추출 결과로 확인하였다. 점수형 4개 특징의 완전 일치율은 0.90~0.93, 평균 절대 차이는 0.07~0.10이었고, 5개 특징 모두 Cohen's kappa 0.79~0.95(최저 MDAS 0.793, 최고 SRS 0.947)로 substantial~almost perfect 수준(Landis & Koch 기준[19])이었다.

2.2 Interpretation: Comparing SHAP Importance and Ablation Results

앞선 ablation 분석과 SHAP 분석은 각각 특징 없이 재학습한 별도 모델과의 성능 차이와 학습된 하나의 모델이

www.kci.go.kr

각 특징을 사용하는 정도를 측정하므로 두 결과는 다를 수 있다. 구조적 위험도는 크기·확산 관련 Kamei 특징과 중간~강한 상관(r=0.3~0.7)을 보였으므로(Ⅳ장 1.3) 제거 후 재학습하면 그 기여가 상관된 다른 특징으로 이전될 수 있으며, RF에서 이를 제거했을 때의 AUC 차이(+0.0030)는 반복 교차검증의 분할 간 변동과 비슷한 크기이므로 제거로 성능이 높아졌다기보다 낮아지지 않은 것으로 해석된다. 상관된 특징은 결함 모델의 해석을 바꿀 수 있고[20] 중요도 산출 방법에 따라서도 해석이 달라지므로[21], 이 불일치만으로 특정 특징의 정보 손실 여부를 판단할 수는 없다. 반면 복잡도 자기평가는 두 모델 모두 SHAP과 ablation 순위가 일치하였다. 다만 ablation은 LLM 5개 특징에 대한 10-fold 평균이고 SHAP은 단일 분할 결과이므로 두 순위의 비교는 서술적 비교로 한정된다.

LR에서 BFIS의 SHAP 값이 점수가 높을수록 감소한 것(Fig. 4)은 버그 수정 의도가 뚜렷한 커밋일수록 결함 확률을 낮게 예측하는 경향과의 연관성일 뿐 인과적 해석의 근거로 보기는 어렵다. BFIS는 FIX와 강한 상관(r=0.691)을 보이므로 이 방향은 다중공선성의 결과일 수 있으며 변경 크기를 통제해도 추가 기여는 작았다(Ⅳ장 3절). 메시지-diff 일치도는 두 모델 모두 최하위권이었고 크기 통제 후에도 기여하지 않았으며(Ⅳ장 3절), 점수가 4점대 후반에 집중된 점(Ⅲ장 3절)에 비추어 변별력이 낮았던 것이 한 요인으로 추정된다. LR에서 순위가 덜 안정적이었던 것은 상관된 특징 간 계수 배분이 재표본마다 달라지기 때문으로 해석된다.

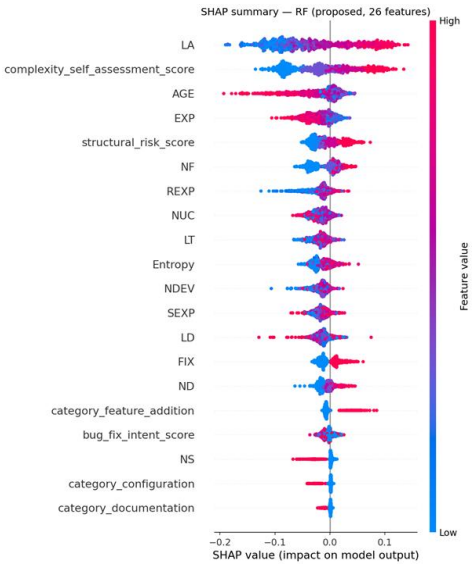


Fig. 1. SHAP summary plot for Random Forest (input columns; CSC is shown as one-hot columns).

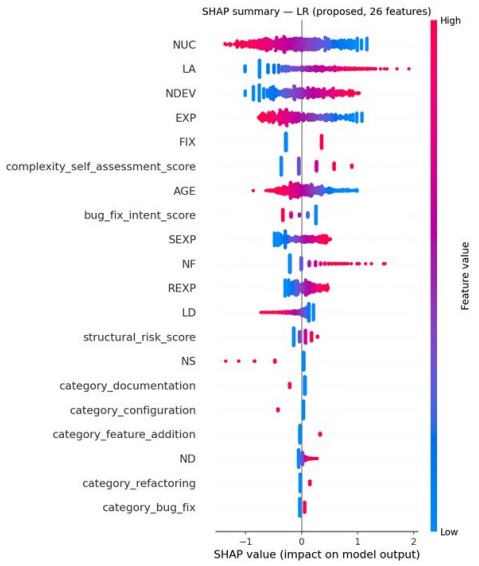


Fig. 2. SHAP summary plot for Logistic Regression (input columns; CSC is shown as one-hot columns).

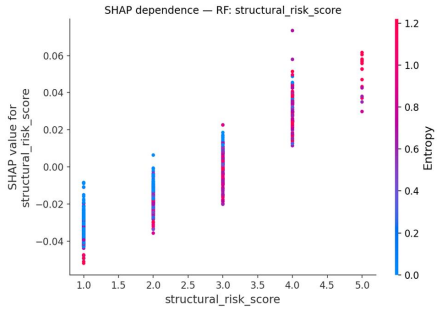


Fig. 3. SHAP dependence plot for structural risk in RF.

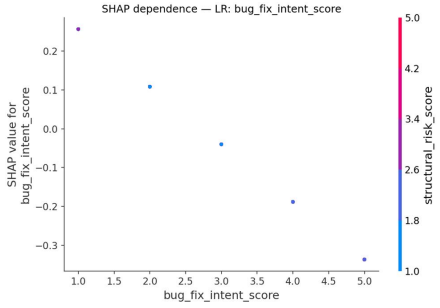


Fig. 4. SHAP dependence plot for bug-fix intent in LR.

3. Measurement Validity of LLM Semantic Features

Ⅳ장 2.1의 재현성은 반복 산출의 신뢰도(reliability)일 뿐이므로, 점수가 의도한 구성개념을 반영하는지(validity) 확인하기 위해 4,000건 전체에서 LLM 점수와 diff에서 직접 계산한 객관적 코드 특성의 대응 관계를 분석하였다. 객관 지표의 계산 정확성은 Kamei 특징과의 대조로 확인하였다(삭제 줄 수 100%, 추가 줄 수와 변경 엔트로피 99.98%(신규 추가 파일 제외), 변경 파일 수 92.5% 일치).

www.kci.go.kr

Spearman 상관 분석 결과(괄호 안은 bootstrap 95% 신뢰구간), 구조적 위험도는 변경량(churn) +0.46[0.43, 0.49], 변경 파일 수 +0.51[0.49, 0.54], 변경 엔트로피 +0.52[0.49, 0.54]와, 복잡도 자기평가는 각각 +0.71, +0.60, +0.57과 중간~강한 양의 상관을 보였다. 버그 수정 의도는 크기 지표와 거의 무관한 반면($|r| \leq 0.10$) 버그 수정 여부(FIX)와 강한 상관(+0.69)을 보였고, 메시지-diff 일치도는 모든 지표와 상관이 약하였다($|r| < 0.15$). 변경 범위 범주(CSC)는 범주 간 변경량과 변경 파일 수의 분포가 유의하게 달랐다(Kruskal-Wallis, $\varepsilon^2 = 0.19$, 0.22).

점수가 단순히 변경 크기를 대신하는지 확인하기 위해, 변경량과 변경 파일 수(로그 변환)만 사용한 LR 모델에 각 LLM 특징을 하나씩 추가하여 층화 10-fold 교차검증(seed=42)으로 비교하였다. 복잡도 자기평가(AUC +0.0159, 10개 폴드 모두 개선), 변경 범위(+0.0158), 구조적 위험도(+0.0095)는 크기를 통제한 뒤에도 AUC를 높여 크기의 단순 대리 변수는 아니었으나, 크기를 통제하면 복잡도 자기평가와 구조적 위험도의 결함 라벨과의 상관이 각각 0.389에서 0.165, 0.305에서 0.137로 절반 이하로 줄어 예측력의 상당 부분은 변경 크기와 공유된 것으로 나타났다. 버그 수정 의도의 추가 기여는 작았고(+0.0033, 보정 전 $p = 0.106$) FIX와의 상관(+0.69)이 강해 두 특징이 담는 정보가 상당 부분 겹치며, 메시지-diff 일치도는 기여가 나타나지 않았다(AUC -0.0003, 부분 상관 95% CI [-0.059, +0.005]).

다만 이 분석은 크기·확산·버그 수정 여부 지표에 한정되며, 상관의 크기만으로 측정 타당성이 입증되거나 부정되지는 않는다. 구조적 위험도, 복잡도 자기평가, 버그 수정 의도는 객관적 특성과 의도한 방향으로 대응하였으나 메시지-diff 일치도는 근거를 얻지 못하였다.

V. Conclusion

본 연구에서 주 지표인 AUC는 무작위 층화 10-fold 교차검증에서 AUC 2개 검정에 대한 다중비교 보정 후에도 두 모델 모두 유의하게 향상되었으나(8개 검정 전체를 보정하면 유의하지 않음) 향상 폭은 0.007~0.010으로 작았고, 10회 반복 분할에서 방향이 일관되었다. 반면 커밋 시각 순서를 유지한 시간순 검증에서는 향상 방향만 유지되고 신뢰구간이 0을 포함하여 개선을 통계적으로 확정할 수 없었다. 보조 지표에서는 LR만 F1·PofB20이 향상되었고 RF의 PofB20은 일관되게 감소하였다. 같은 LLM의 zero-shot 직접 예측은 시간순 테스트에서 AUC 0.7231로 LR보다 낮고 RF와는 구분되지 않아 특징으로 결합하는 구성이 직접 예측보다 성능이 낮지 않았으나, 이는 단일 프롬프트의 zero-shot과 비교한 결과이며 기존 JIT 결함 예측 기법 대비 우수성을 뜻하지 않는다. 소형 모델인 Haiku 5.5로 특징을 다시 추출한 경우에도 AUC 향상의 방향과 zero-shot 직접 예측과의 비교 결과는 같았으나, 시간순 검증에서 신뢰구간이 0을 포함하지 않은 경우는 holdout과 확장 윈도우 일부 구간에 그쳤고, 메시지-diff 일치도는 두 모델 간 점수의 일치 수준이 낮았다. SHAP 분석에서는 복잡도 자기평가가 두 모델에서 높은 기여도를 보이고 제거 결과와도 일치한 반면 RF의 구조적 위험도는 SHAP 중요도(19개 중 5위)에 비해 제거 시 성능 변화가 없어, 설명가능한 AI 기법 적용 시 성능 검증을 병행할 필요가 있다. 설명 안정성은 RF가 더 높았으나 이는 설명의 충실도나 결함 원인으로서의 타당성을 의미하지 않으며, 측정 타당성 분석에서 구조적 위험도와 복잡도 자기평가의 예측력은 상당 부분 변경 크기와 공유되었고 메시지-diff 일치도는 근거를 얻지 못하였다. 따라서 본 연구의 의의는 LLM 시맨틱 특징의 성능 우위보다는 전통적 JIT 특징과의 결합 가능성과 설명 분석의 유용성 및 한계를 검토한 데 있다.

다만 본 연구는 단일 오픈소스 프로젝트(OpenStack)의 표본 데이터로 수행되었고 모델 학습 시드는 고정하였으며, 메시지 품질에 의존하는 특징의 효과는 OpenStack의 커밋 메시지 작성 관행에서 얻은 결과이다. SZZ 기반 라벨링의 노이즈, 라벨 확정까지의 검증 지연(verification latency)[22]으로 시간순 검증에서도 학습 구간 끝부분의 라벨은 예측 시점에 확정되지 않았을 수 있다. 실시간 JIT 결함 예측에 적용하면 커밋마다 LLM 추론이 필요하여 추론 비용(배치 처리 기준 커밋 1건당 약 0.004달러, Ⅲ장 3절)과 커밋 직후 피드백까지의 처리 지연이 발생하고(본 연구의 처리 시간은 배치 처리 기준), 소스 코드와 커밋 정보가 외부 API로 전송되므로 비공개 저장소에서는 데이터 보호 문제가 따른다. LLM 입력에는 작성자 이름·이메일과 작성 시각(git show 헤더)이 포함되어 점수가 diff 내용 외에 작성자 정보의 영향도 받았을 수 있다. 공개 저장소의 과거 커밋이므로 LLM이 해당 커밋이나 이후의 수정 이력을 사전학습 중 접했을 가능성(데이터 오염)[23]이 있으며, 이 경우 zero-shot 직접 예측에 유리하게 작용한다. 측정 타당성은 객관 지표와의 대응으로만 확인하여 사람 평가자와의 일치도는 검증하지 못하였다. 또한 LLM 입력으로 사용한 재구성 diff의 코드 지표는 일부 커밋에서 Kamei

www.kci.go.kr

특징 값과 일치하지 않았고(Ⅲ장 1절), 입력이 절단된 369건(9.2%)은 LLM 점수가 diff의 앞부분에서, 객관 지표는 diff 전체에서 산출된 상태로 측정 타당성 분석(Ⅳ장 3절)에 포함되었다. 기존 시맨틱 기반 JIT 방법(JITLine, DeepJIT, CC2Vec, JIT-Fine)과의 동일 조건 비교, few-shot 및 파인튜닝 방식은 수행하지 못하였고 모델 규모는 같은 제공사의 두 등급만 비교하였으며 오픈소스 LLM은 포함하지 않았으므로, 본 연구의 결과는 동일 데이터에서 Kamei 특징에 LLM 시맨틱 특징을 추가한 효과에 한정되며 이들 비교는 후속 연구에서 다룰 필요가 있다.

향후에는 복수의 오픈소스 프로젝트(예: ApacheJIT[24])로 실험을 확장하여 일반화 가능성과 커밋 메시지 품질-LLM 특징 기여도의 관계를 검증하고, 의존성 그래프나 호출 그래프를 활용해 변경의 영향 범위까지 예측하는 변경 영향도 분석(Change Impact Analysis)으로 확장할 필요가 있다.

REFERENCES

[1] E. Hong, "Prediction Model of Software Fault using Deep Learning Methods," The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 22, No. 4, pp. 111-117, Aug. 2022. DOI: 10.7236/JIIBC.2022.22.4.111

[2] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A Large-Scale Empirical Study of Just-in-Time Quality Assurance," IEEE Trans. on Software Engineering, Vol. 39, No. 6, pp. 757-773, June 2013. DOI: 10.1109/TSE.2012.70

[3] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "DeepJIT: An End-to-End Deep Learning Framework for Just-in-Time Defect Prediction," Proceedings of the 16th IEEE/ACM International Conference on Mining Software Repositories, pp. 34-45, May 2019. DOI: 10.1109/MSR.2019.00016

[4] C. Pornprasit and C. Tantithamthavorn, "JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction," Proceedings of the 18th IEEE/ACM International Conference on Mining Software Repositories, pp. 369-379, May 2021. DOI: 10.1109/MSR52588.2021.00049

[5] A. Sayedsalehi, P. C. Rigby, and A. Mockus, "DRS-OSS: A Diff-Risk Scoring Tool for Continuous Integration Workflows," arXiv:2511.21964, Nov. 2025.

[6] I. Yeo, S. Lee, D. Ryu, and J. Baik, "Project Evolution-aware Prompting of LLMs for Just-in-time Defect Prediction in Edge-cloud Systems," Journal of Web Engineering, Vol. 25, No. 3, pp. 395-416, Apr. 2026. DOI: 10.13052/jwe1540-9589.2535

[7] E. S. Nam, Y. S. Choi, and C. K. Lee, "A Study on the Defect Detection of Fabrics using Deep Learning," Smart Media Journal, Vol. 11, No. 11, pp. 92-98, Dec. 2022. DOI: 10.30693/SMJ.2022.11.11.92

[8] Y. Liu, K. Jabre, R. Abreu, Z. J. Carmichael, V. Murali, A. Patel, J. Ge, W. Sun, C. Zhang, A. Mockus, D. Khavari, P. C. Rigby, and N. Nagappan, "A Preliminary Study on Explaining Risk of Code Changes using LLM-Based Prediction Models," Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware), Montreal, Canada, pp. 259-267, July 2026. DOI: 10.1145/3805760.3814916

[9] S. McIntosh and Y. Kamei, "Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction," IEEE Trans. on Software Engineering, Vol. 44, No. 5, pp. 412-428, May 2018. DOI: 10.1109/TSE.2017.2693980

[10] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep Just-in-Time Defect Prediction: How Far Are We?," Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 427-438, July 2021. DOI: 10.1145/3460319.3464819

[11] M. L. Monteiro, G. G. Cabral, and A. L. I. Oliveira, "Pre-trained Code Language Models for Just-in-Time Software Defect Prediction: An Empirical Study," Proceedings of the 35th Brazilian Conference on Intelligent Systems (BRACIS), pp. 347-361, 2025.

[12] M. L. Monteiro, G. G. Cabral, and A. L. I. Oliveira, "CodeFlowLM: Incremental Just-In-Time Defect Prediction with Pretrained Language Models and Exploratory Insights into Defect Localization," arXiv:2512.00231, Dec. 2025.

[13] D. Nam, T. Kim, D. Ryu, and J. Baik, "ReDef: Do Code Language Models Truly Understand Code Changes for Just-in-Time Software Defect Prediction?," arXiv:2509.09192, Sept. 2025.

[14] C. Ni, W. Wang, K. Yang, X. Xia, K. Liu, and D. Lo, "The Best of Both Worlds: Integrating Semantic Features with Expert Features for Defect Prediction and Localization," Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 672-683, Nov. 2022. DOI: 10.1145/3540250.3549165

[15] Y. Mahmoud, A. Azim, R. Liscano, K. Smith, Y.-K. Chang, G. Seferi, and Q. Tauseef, "Leveraging LLM Enhanced Commit Messages to Improve Machine Learning Based Test Case Prioritization," Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering, pp. 45-54, June 2025. DOI: 10.1145/3727582.3728681

[16] J. Choi, J. Lee, D. Ryu, and S. Kim, "Identification of Generative Adversarial Network Models Suitable for Software Defect Prediction," Journal of KIISE, Vol. 49, No. 1, pp. 52-59, Jan.

www.kci.go.kr

2022. DOI: 10.5626/JOK.2022.49.1.52

[17] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," IEEE Trans. on Software Engineering, Vol. 43, No. 1, pp. 1-18, Jan. 2017. DOI: 10.1109/TSE.2016.2584050

[18] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," Advances in Neural Information Processing Systems (NIPS), pp. 4765-4774, Dec. 2017.

[19] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," Biometrics, Vol. 33, No. 1, pp. 159-174, Mar. 1977. DOI: 10.2307/2529310

[20] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The Impact of Correlated Metrics on the Interpretation of Defect Models," IEEE Trans. on Software Engineering, Vol. 47, No. 2, pp. 320-331, Feb. 2021. DOI: 10.1109/TSE.2019.2891758

[21] G. K. Rajbahadur, S. Wang, G. A. Oliva, Y. Kamei, and A. E. Hassan, "The Impact of Feature Importance Methods on the Interpretation of Defect Classifiers," IEEE Trans. on Software Engineering, Vol. 48, No. 7, pp. 2245-2261, July 2022. DOI: 10.1109/TSE.2021.3056941

[22] G. G. Cabral, L. L. Minku, E. Shihab, and S. Mujahid, "Class Imbalance Evolution and Verification Latency in Just-in-Time Software Defect Prediction," Proceedings of the 41st IEEE/ACM International Conference on Software Engineering, pp. 666-676, May 2019. DOI: 10.1109/ICSE.2019.00076

[23] O. Sainz, J. Campos, I. García-Ferrero, J. Etxaniz, O. Lopez de Lacalle, and E. Agirre, "NLP Evaluation in trouble: On the Need to Measure LLM Data Contamination for each Benchmark," Findings of the Association for Computational Linguistics: EMNLP 2023, pp. 10776-10787, Dec. 2023. DOI: 10.18653/v1/2023.findings-emnlp.722

[24] H. Keshavarz and M. Nagappan, "ApacheJIT: A Large Dataset for Just-In-Time Defect Prediction," Proceedings of the 19th International Conference on Mining Software Repositories, pp. 191-195, May 2022. DOI: 10.1145/3524842.3527996

Authors



Kyoungmin Yoon is currently a Ph.D. candidate in the Department of Convergence Engineering, Graduate School of Venture, Hoseo University, Seoul, Korea.



Nammee Moon is currently a Professor in the Department of Convergence Engineering, Graduate School of Venture, Hoseo University, Seoul, Korea.

www.kci.go.kr