



Dream

+



Think

+



Idea

+



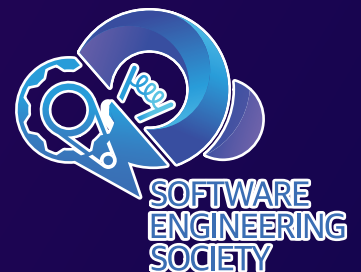
come true

COMMUNICATIONS

OF SOFTWARE ENGINEERING SOCIETY

09/2025 VOL.5 NO. 2

소프트웨어공학 소사이어티 소식





COMMUNICATIONS OF SOFTWARE ENGINEERING SOCIETY 소프트웨어공학 소사이어티 소식

- 03 **게스트 편집자 인사말** / 강종구 (성신여자대학교)
- 04 **기고문**
대규모 언어 모델 시대의 프로그램 정정 패러다임 연구 동향 / 이병정 (서울시립대), 양근석 (한경국립대학교)
- 16 **신진연구자 소개**
안가빈 (고려대학교)
- 18 **따끈한 논문**
딥러닝 라이브러리를 하얗게 테스트하기 위한 경량 Concolic Testing의 첫걸음 / 김미정 (울산과학기술원)
- 20 **국내외 학술행사 소개**
ICSE/FSE/ISSTA 2025를 둘러보고 / 유신 (한국과학기술원)
소프트웨어공학 단기 강좌 후기 / 류덕산 (전북대학교)
- 26 **기업탐방**
슈어소프트테크 : Software for Safe World / 최경화 (슈어소프트테크㈜)
- 30 **소사이어티 광장**
- 31 **2025 국내외 학술대회**
- 32 **기고문 및 소식모집**
- 33 **소사이어티 조직도**
- 33 **발행정보**

09/2025
VOL.5 NO. 2



**SOFTWARE
ENGINEERING
SOCIETY**

• 발행인 이정원 • 편집인 이주용, 정우성, 지은경, 손정주 • 발행일 2025. 9. 1



강종구
성신여대 교수

한국정보과학회 회원, 소프트웨어공학 소사이어티 소식지 독자 여러분, 안녕하세요.
이번호의 게스트 편집을 맡게 된 성신여자대학교 시융합학부 강종구 교수입니다.

SDV(Software-Defined Vehicle), 로보택시, 물류드론, AAM(Advanced Air Mobility) 등과 같이 소프트웨어 중심 미래 모빌리티와 인공지능 기술과의 융합에 대한 관심이 그 어느 때보다도 높은 시기입니다. 미래의 모빌리티는 소프트웨어 중심 플랫폼과 발생하는 데이터의 처리와 응용을 수단으로 혁신이 이루어지고 있습니다. 소프트웨어, 데이터, 인공지능에 대한 공학 - 엔지니어링은 기존 소프트웨어 엔지니어링을 과학적 원리, 도구, 기술을 통해 복잡한 컴퓨팅 시스템 소프트웨어를 데이터 수집부터 모델 훈련, 제품이나 고객이 모델을 사용할 수 있도록 하는 전 생명 주기를 포함하여 설계하고 구축하도록 확장하는 것입니다. 스마트 모빌리티 디바이스의 지능화와 데이터, 인공지능 모델로 인해 소프트웨어의 규모와 복잡도가 커짐에 따라 지속가능한 체계를 구축하는 엔지니어링의 중요성 또한 커지고 있습니다.

이러한 시기에 이번 소프트웨어공학 소사이어티 9월 소식지와 한국정보과학회 뉴스레터 805호 소프트웨어공학 소사이어티 특집호를 함께 연계하여 준비하게 되었습니다.

이번 호는 새롭고 중요한 계기에 특별히 아래의 소사이어티 내에서 중요한 분들을 초대해서 구성해 보았습니다.

- 전임 회장단 이시자 출선수범하시어 “자동 프로그램 정정 기술 동향”에 대한 메인 기고문을 투고해주신 서울시립대 이병정 교수님
- 소프트웨어공학 분야 Top Conference인 ICSE 2025에서 발표하신 따끈한 최신 대표 논문을 소개해주신 UNIST 김미정 교수님
- ICSE/FSE/ISSTA 2025 국제학술대회를 둘러보시고 소프트웨어 공학과 학술대회의 미래에 대해서 투고해주신 KAIST 유신 교수님
- 소프트웨어 시대의 변화와 함께 끊임없는 발전을 꿈꾸시는 고려대학교 컴퓨터공학과에 임용을 앞두고 계신 안가빈 신임교수님
- 소프트웨어로 만드는 안전한 세상의 비전을 AI시대로의 여정으로 계속하고 계시는 슈어소프트테크 최경화 연구소장님

이 외에도 회원 소개, 소식과 8월에 진행된 제13회 소프트웨어공학 단기전문강좌에 대한 후기를 수록하였습니다. 단기 강좌 후기는 소사이어티 교육부회장으로 수고를 아끼지 않으신 전북대학교 류덕산 교수님께서 작성해 주셨습니다.

이번 2025년 9월 소프트웨어공학 소사이어티 소식지와 한국정보과학회 뉴스레터 805호가 소프트웨어 엔지니어링 and beyond에 관심이 있으신 많은 독자분들께 도움이 되기를 바랍니다. 기꺼이 바쁜 시간을 내어 원고를 투고해주신 투고자 분들, 그리고 본 소식지 발간을 위해 지원해 주신 이정원 소사이어티 회장님, 가장 힘써 주신 편집부위원 이주용, 정우성, 지은경, 손정주 교수님께 감사드립니다.

아직 더운 어느 여름날 밤 편집자 **강종구** 올림



대규모 언어 모델 시대의 프로그램 정정 패러다임 연구 동향



서울시립대학교 이병정 교수



한경국립대학교 양근석 교수

1. 서론

현대 소프트웨어 시스템은 규모와 복잡도가 지속적으로 증가하고 있으며, 이에 따라 소프트웨어 버그의 발생 빈도 또한 증가하는 경향을 보이고 있다. 대형 산업용 소프트웨어 프로젝트에서는 다양한 기능 변종과 복잡한 제품군 구조가 도입됨에 따라 테스트 및 유지보수의 부담이 크게 확대되고 있으며, 이는 소프트웨어의 신뢰성, 안정성, 그리고 보안에 부정적인 영향을 미친다 [1][2]. 특히, 소프트웨어 유지보수 비용은 전체 개발 생명주기 비용의 50~70%를 차지하며, 이 중 다수가 버그 정정에 기인하는 것으로 보고되었다 [3]. 이러한 비용 구조는 버그 관리 및 정정의 자동화가 소프트웨어 개발 효율성에 중요한 영향을 미친다는 점을 보인다.

프로그램 정정 패러다임은 일반적으로 세 가지 주요 과정으로 구성되고 목표는 자동 프로그램 정정(Automatic Program Repair, APR)이다: 버그 분류(Bug Triage), 버그 위치 추정(Bug Localization), 버그 정정(Bug Repair). 각 과정의 정확성과 효율성은 전체 소프트웨어 품질 향상 및 유지보수 비용 절감에 직접적인 기여를 한다 [4]. 최근 들어 대규모 언어 모델(Large Language Models, LLMs)의 비약적인 발전은 자연어 처리(Natural Language Processing, NLP) 영역을 넘어 소프트웨어 공학 분야에도 적용되며, 새로운 기술적 전환점을 만들고 있다. 특히 GPT, CodeBERT, Codex와 같은 모델들은 방대한 양의 소스코드와 자연어 데이터로부터 학습하여 코드 생성, 요약, 분석 등 다양한 소프트웨어 관련 작업에서 뛰어난 성능을 입증하고 있으며 [5], 자동 프로그램 정정 분야에서도 그 응용 가능성이 활발히 탐색되고 있다 [6][7]. 기존에는 주로 휴리스틱 기반 또는 정형 규칙 기반의 정정 기법들이 사용되었으나, 최근에는 LLM의 패턴 인식 및 추론 능력을 활용한 정교한 접근법이 주목받고 있다. 예를 들어, RAG(Retrieval-Augmented Generation), 다중 에이전트 기반 프롬프트 설계, 하이브리드 학습 등의 기술이 APR 시스템에 적용되면서, 기존의 검색-기반 또는 패턴-기반 방식에 대한 대안을 제시하고 있다 [8]. 본 연구는 이러한 기술적 변화에 대응하여, 대규모 언어 모델 기반 프로그램 정정 패러다임 연구의 국내외 주요 흐름을 체계적으로 분석하고자 한다. 구체적으로, 버그 분류, 버그 위치 추정, 그리고 버그 정정 과정에 걸친 LLM 응용 연구를 비교 분석하고, 각 기술의 강점 및 한계를 논의한다. 더불어, APR 연구의 기반이 되는 벤치마크 데이터집합의 활용 현황을

정리하며, 향후 발전을 위한 학술적 시사점을 도출하는 것을 목표로 한다. 본 연구의 구성은 다음과 같다. 2장에서 버그 분류, 3장에서 버그 위치 추정, 4장에서 버그 정정, 5장에서 벤치마크 데이터집합, 그리고 마지막으로 6장에서 결론과 향후전망을 소개한다.

2. 버그 분류 (Bug Triage)

버그 분류 기법은 전통적인 버그 분류 기법과, LLM 기반 버그 분류 기법으로 크게 나눌 수 있다 (그림 1).

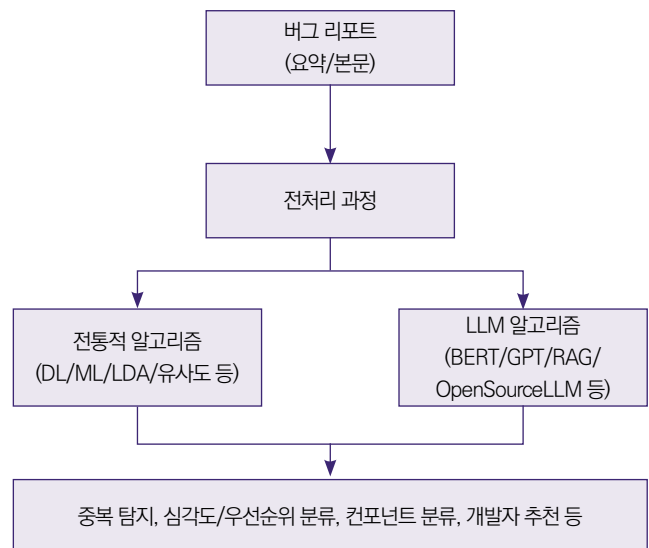


그림 1. 버그 분류 도식도

일반적으로, 소프트웨어 개발 중 발생하는 버그 리포트(요약 및 본문)를 입력으로 받아, 이를 TF-IDF와 같은 벡터화 기법을 통해 수치화한다. 이후, 해당 벡터화된 데이터를 기반으로 전통적인 알고리즘(DL, ML, LDA 등) 또는 대규모 언어 모델 기반 알고리즘(BERT, GPT, RAG, 오픈소스 LLM 등)을 활용하여 분석한다. 최종적으로 중복 버그 탐지, 심각도 및 우선순위 분류, 컴포넌트 분류, 개발자 추천 등의 다양한 분류 작업을 수행한다.

2.1 전통적인 버그 분류 기법

버그 분류는 전통적으로 문서 기반 정보 검색 및 고전적 기계학습 기법을 활용해 중복 리포트 탐지, 심각도/우선순위 분류, 컴포넌트/모듈 식별, 개발자 추천(배정)을 수행해 왔다. 대표적으로, 버그 리포트의 요약-본문을 TF-IDF 벡터로 표현하고 코사인 유사도로 개발자 또는 컴포넌트를 매칭하는 방법[11], 토픽 모델(LDA) 및 SVM/나이브 베이즈 기반 분류, 전이-증분 학습을 통한 프로젝트 변화 대응이 널리 연구되었다. 개발자 추천 분야에서는 과거 수정 이력과 기술 스택을 반영해 “누가 이 버그를 고쳐야 하는가?”를 모델링하는 접근이 표준화되었고 [9][11][12], 버그 토싱(tossing) 그래프를 통해 최종 담당자에 도달하기까지의 라우팅을 단축하는 기법이 제안되었다[10]. 평가는 주로 정확도/정밀도-재현율 F1, Top-k 추천 정확도/Recall, MRR/MAP 등의 지표로 수행되며, 데이터는 Eclipse, Mozilla, Apache 등 대형 오픈소스 이슈 저장소가 주로 활용되었다. 이 분야 종합 조사에서는 특징 공학, 데이터 불균형, 콜드스타트, 인력 가용성 제약 등을 핵심 과제로 분류한다[12].

2.2 LLM 기반 버그 분류 기법

최근에는 사전학습 언어모델(PLM)과 LLM을 활용해 텍스트 및 코드 맥락을 통합적으로 이해하는 분류기가 등장했다. BERT·CodeBERT·T5 등을 미세조정하여 중복 탐지, 심각도/우선순위, 컴포넌트 분류, 개발자 추천 성능을 개선하는 연구가 보고되었고, 대형 LLM에는 지침튜닝(instruction tuning), RAG(문서-이슈-코드 검색 결합), 에이전트 협업(agentic collaboration)을 결합해 운영환경 적합성을 높이려는 시도가 진행 중이다[8]. 보안 도메인에서는 CWE 분류와 취약점 리포트 심각도 판정 등 특화 태스크에 LLM을 적용해 의미 해석 기반의 분류 정확도를 제고하는 방향이 제시되었다[6][8].

국내 적용 측면에서는 한국어 사전학습 모델(예: KoBERT, KoELECTRA)을 기반으로 한 분류기 및 오픈프미스 LLM 운용이 논의되고, 라벨 부족을 해소하기 위해 약지도·자기지도·합성 데이터가 병행된다. SPL(Software Product Line) 맥락에서는 제품군-변종 관리와 연동한 자동 분류 실험이 보고되며, 기업 환경 요구(보안-비용-추적성)에 맞춘 설명가능성·감사 가능성 확보가 강조된다[1][3][13][8].

2.3 국내외 연구 비교 및 시사점

국외는 표준화된 데이터집합과 지표, 대형 프로젝트 멀티 도메인 실험, 에이전트형 분산 추론 구조까지 포함한 기술 성숙도가 높다. LLM을 RAG/에이전트와 결합해 운영 환경의 지식·코드·이슈 산출물을 통합하는 경향이 강하며, 정확도, Top-k, MRR/MAP-처리비용을 함께 관리한다[8]. 반면 국내는 실제 제약(보안·오픈프미스·한국어 리포트)을 반영한 현장 적용성에 강점이 있으나, 공개 벤치마크 부족, 다중언어/도메인 다양성 미흡, 평가 표준 미정립이 학술적 비교·일반화 관점의 한계로 지적된다[1][3][13].

이에 따른 시사점은 다음과 같다.

- 산출물 통합: 이슈 텍스트 외 로그, 스택트레이스, 패치, 테스트 결과를 결합해 분류 및 추천 정확도를 높여야 한다[2].

- 국내 벤치마크 구축: SPL 맥락을 반영한 국내 공개 벤치마크와 시계열(time-aware) 평가를 마련해 재현성과 전이 성능을 검증할 필요가 있다[1][3].
- 에이전트형 파이프라인: 수집 → 정제 → 분류 → 검증 → 우선순위 결정 전 과정을 에이전트 기반으로 자동화하고, 사람-반복 작업(human-in-the-loop) 설계를 통해 운영 적합성을 높인다[8].
- 신뢰성·설명가능성: 편향/드리프트 탐지, 반대사실 설명, 감사 추적성을 포함하는 신뢰성 프레임워크를 정립한다[4][8].
- 산학 협력 평가: 정확도뿐 아니라 처리 지연·운영 비용·개발자 수용성을 포함한 실증 평가 체계를 구축한다[13].

3. 버그 위치 추정 (Bug Localization)

버그 위치 추정과 관련한 방법은 크게 전통적인 알고리즘과 LLM 기반 알고리즘으로 분류한다 (그림 2). 버그 위치 추정 과정은 버그 리포트, 소스 코드, 테스트 로그, 스택 트레이스 등의 다양한 입력 정보를 기반으로 시작된다. 이러한 입력 데이터는 먼저 전처리 과정을 통해 정제되고 벡터화되며, 분석 가능한 형식으로 변환된다.

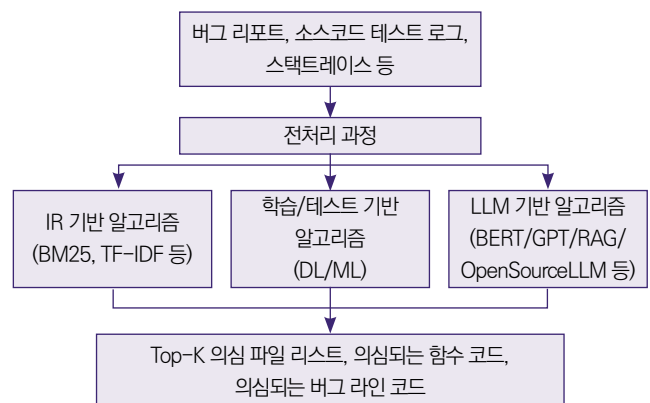


그림 2. 버그 위치 추정 도식도

그 다음, 위치 추정을 위한 다양한 접근 방식이 사용된다. 첫 번째로, 정보검색 기반 알고리즘은 BM25, TF-IDF 등과 같은 검색 기법을 활용하여 버그 리포트와 코드 간의 유사도를 계산한다. 두 번째로, 학습 기반 또는 테스트 기반 알고리즘은 기존 버그 수정 이력이나 테스트 실패 결과를 활용하여 딥러닝 혹은 머신러닝 모델을 학습시킨다. 세 번째로는 LLM 기반 알고리즘을 이용하여, 자연어와 코드 정보를 통합적으로 처리하고, 대규모 언어 모델을 통해 위치를 추정한다. 이러한 과정을 통해 최종적으로 Top-K 수준의 의심 파일 리스트, 의심되는 함수 또는 코드 블록, 의심되는 버그 라인 코드 등을 결과로 제공한다.

3.1 전통적인 버그 위치 추정 기법

버그 위치 추정은 소프트웨어 디버깅 과정에서 버그 위치를 자동으로 식별하기 위한 핵심 기술로, 전통적으로는 스펙트럼 기반 기법(Spectrum-Based Fault

Localization, SBFL)과 정보 검색 기반 방법이 널리 사용되었다.

SBFL 계열 기법은 테스트 케이스의 통과 여부와 소스 코드 라인 실행 정보를 바탕으로 각 코드 요소의 결함 가능성(suspiciousness)을 정량화하여, 높은 의심 점수를 가진 코드 라인을 우선적으로 제시하는 방식이다. 대표적인 예로는 Tarantula, Ochiai, DStar 등의 수식 기반 기법이 있으며, 이들은 테스트 커버리지 정보를 통해 Top-k 또는 EXAM 점수와 같은 지표로 평가된다 [14][15][16]. 정보 검색 기반 기법은 버그 리포트를 질의(Query)로 보고, 소스 코드 파일이나 함수 등을 문서(Document)로 간주하여, 두 사이의 유사도를 계산하는 방식이다. 초기에는 TF-IDF 기반 접근이 주를 이루었으며, 이후 소스코드 히스토리, 개발자 활동, 토픽 모델링 등을 통합한 BugLocator [17], BLUIR [18], AmaLgam+ [19] 등 모델들이 제안되었다. 이들 기법은 대부분 Eclipse, Mozilla, Apache 등의 오픈소스 저장소를 기반으로 학습 및 평가되며, MRR, MAP, Top-N 정확도 등을 주요 성능 지표로 사용한다 [20].

이와 같은 전통 기법들은 비교적 해석 가능성이 높고 구현이 용이하다는 장점이 있으나, 다중 모듈 간 상호작용, 의미적 오류, 코드 맥락의 복잡성 등을 효과적으로 반영하지 못한다는 한계를 갖고 있다 [16][20].

3.2 LLM 기반 버그 위치 추정 기법

최근 대규모 언어 모델(LLM)의 발전은 버그 위치 추정 기법에도 본질적인 변화를 가져오고 있다. 기존의 규칙 기반 접근에서 벗어나, 코드와 버그 리포트를 함께 이해하고, 의미 기반 추론을 통해 오류 위치를 정밀하게 예측하는 방식으로 진화하고 있다.

Hossain et al.이 제안한 Toggle 프레임워크[6]는 LLM을 활용해 토큰 수준에서 오류 위치를 식별하고, 이후 별도의 정정 모델을 통해 수정 코드를 생성하는 구조를 제시하였으며, Defects4J 및 CodeXGLUE에서 기존 대비 Top-N, MAP, MRR 지표 모두에서 우수한 성능을 기록하였다. Chakraborty et al.이 발표한 BLAZE 시스템[21]은 프로젝트 간, 언어 간 일반화 성능을 높이기 위해 동적 청킹(dynamic chunking)과 하드-예제(hard-example) 학습 전략을 채택하였고, 크로스-프로젝트(cross-project) 환경에서 Top-1 기준 정확도가 최대 120% 향상되었다고 보고하였다. 또한, Feng et al.이 개발한 DEVLore 프레임워크[22]는 이슈 리포트, 디버깅 로그, 스택 트레이스 등 다양한 소프트웨어 산출물을 LLM 입력으로 결합하여, 기존 단일 메서드 단위 버그(single-method bugs)뿐 아니라 다중 메서드 버그(non-single-method bugs)까지 높은 정확도로 위치 추정을 수행하였다. 해당 시스템은 Defects4J v2.0 기준으로 단일 메서드 버그에 대해 약 49.3%의 위치 추정 정확도와 56.0%의 테스트 통과 패치 생성율을 기록하였다. 최근에는 지식-향상(knowledge-enhanced) LLM을 활용하여 정적 분석 정보(예: 제어 흐름, 타입 오류 등)와 사전학습된 모델 지식을 결합한 문맥 기반 버그 위치 추정도 제안되고 있으며, 이는 버그 리포트가 불완전하거나 모호한 경우에 특히 효과적인 것으로 보고된다 [8][23].

국내 연구는 상대적으로 초기 단계에 머물러 있으며, 기존의 SBFL 또는 정적 분석 기반 접근을 중심으로 연구가 진행되고 있다. 그러나 일부 산업체에서는 정적 분석 도구의 결과를 LLM에 프롬프트 형태로 결합하여 위치 추정 정확도를

개선하는 PoC(Proof of Concept) 수준의 실험이 보고되고 있다. 특히 SPL 기반 개발 환경에서는 클론 코드 간 차이 분석을 통해 버그 위치를 유추하고, 이를 LLM 기반 자동 정정 파이프라인과 연동하려는 시도가 일부 이루어지고 있다 [1][3][13].

3.3 국내외 연구 비교 및 시사점

국외 연구는 LLM 기반 버그 위치 추정 기법에 있어 기술 성숙도와 정량적 성능에서 우위를 점하고 있다. Toggle, BLAZE, DEVLore 등의 시스템은 LLM의 코드 이해 능력을 기반으로 크기가 작은 토큰 단위 예측을 수행하며, 기존 APR 파이프라인과의 결합을 통해 패치 생성까지 연동 가능한 구조를 구현하고 있다. 특히 다양한 산출물(버그 리포트, 로그, 스택 트레이스 등)을 통합 입력으로 활용하는 구조는 실제 운영 환경과 유사한 조건에서의 성능 향상에 기여하고 있다 [6][21][22].

반면 국내 연구는 여전히 정적 분석 또는 테스트 기반 기법 중심이며, LLM의 능력을 적극적으로 활용한 공개 사례는 많지 않다. 특히 공용 벤치마크를 통한 재현 가능한 정량 평가가 부재하며, 다양한 언어와 프로젝트에 대한 일반화 성능 측정 역시 미흡하다. 그러나 실무 중심의 특화 도메인(SPL, 산업용 코드베이스 등)을 고려한 구조적 설계 실험은 잠재력을 보여주고 있다 [3][13].

이에 따라 다음과 같은 시사점을 도출할 수 있다:

- 버그 리포트뿐 아니라 스택 트레이스, 테스트 로그, 코드 문맥 등의 다양한 산출물을 활용한 통합 입력 구조가 필요하다. 이는 실제 시스템 수준 오류에서의 정확도를 높이는 핵심 요소이다 [22].
- 국내 SPL 기반 개발 환경에 특화된 Java/C++ 중심의 변종 분석 및 전이 학습 구조 설계가 요구된다 [1][13].
- Defects4J, BEETLEBOX, SWE-Bench 등 공개 벤치마크를 활용한 재현 가능한 평가 체계를 구축하고, Top-N, MRR, MAP 등의 정량 지표를 기반으로 성능을 객관화해야 한다 [16][24].
- 향후 연구는 LLM을 이용한 고정밀 위치 추정 결과를 기존 APR 시스템과 결합하여, 버그 위치 추정과 정정을 분리된 구조로 통합하는 방향으로 발전시킬 필요가 있다 [6][8].

4. 버그 정정 (Bug Repair)

버그 정정에서는 전통적인 자동 프로그램 정정 기법과 LLM 기반 프로그램 정정 기법이 있다 (그림 3). 버그 정정 자동화 과정은 먼저 버그 리포트, 소스 코드, 테스트 로그, 스택 트레이스 등 다양한 소스에서 정보를 수집하는 것으로 시작된다. 이렇게 수집된 정보는 전처리 과정을 거쳐, 기계 학습이나 언어 모델이 이해할 수 있는 형태로 정제된다. 그 다음 단계에서는 여러 접근 방식이 적용될 수 있다. 전통적인 방법으로 첫째는 수정 규칙, 의미적 유사성 또는 코드 패턴을 활용하는 접근이고, 둘째는 과거의 버그와 수정 이력을 학습하거나 테스트 실행 결과를 기반으로 하는 머신러닝/딥러닝 기반 접근이다.

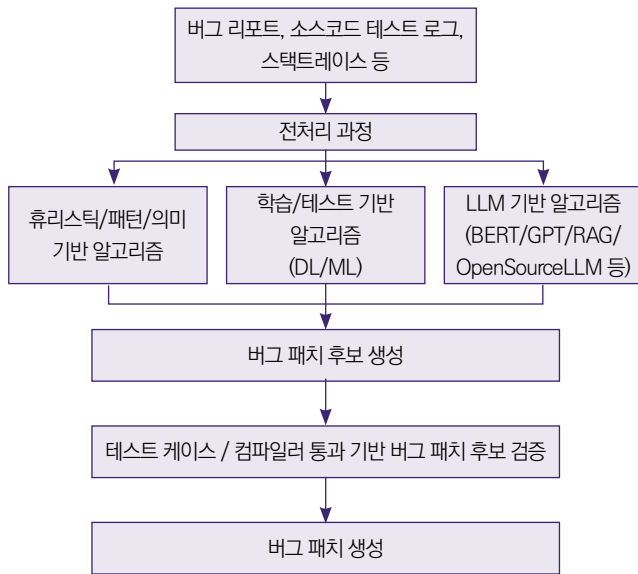


그림 3. 버그 정정 도식도

마지막으로 버그 설명과 코드 정보를 통합적으로 이해하는 대규모 언어 모델(LLM) 기반 접근이 있다. 이 방식들은 각각의 특성을 바탕으로 다양한 방식으로 패치 후보를 생성하게 된다. 이렇게 생성된 버그 패치 후보들은 다음 단계에서 테스트케이스 혹은 컴파일러 통과 여부를 활용하여 검증된다. 즉, 생성된 패치가 실제로 문제를 해결하는지를 자동화된 테스트를 통해 확인한다. 최종적으로, 테스트를 통과한 패치 후보는 실제로 적용 가능한 버그 패치로 생성되며, 이 과정을 통해 자동화된 버그 정정이 완료된다.

4.1 전통적인 자동 프로그램 정정

전통적인 APR 기법은 휴리스틱 기반 APR(Heuristic-based APR), 패턴 기반 APR(Pattern-based APR), 의미 기반 APR(Semantic-based APR), 그리고 학습 기반 APR(Learning-based APR)로 분류할 수 있다.

먼저, 휴리스틱 기반 APR은 자동화된 패치 생성을 검색 공간 표현 및 탐색 문제로 간주한다 [25]. 따라서 프로그램을 정정하는 것은 편향된 휴리스틱 또는 무작위 검색을 통해 템플릿 편집의 검색 공간을 탐색하는 것과 같다. 이러한 명시적인 검색 접근 방식은 목적 함수에 의해 안내되며, 종종 테스트케이스들 전체 또는 부분적으로 사용한다. 패치가 처음에 실패한 테스트(버그를 나타내는 테스트)를 포함한 모든 테스트를 통과할 때 바르게 정정된다고 간주한다. GenProg [26]은 버그 코드 AST를 기반으로 한 패치 표현을 제안했다. 이 표현은 세 가지 유전 연산자(즉, 돌연변이, 크로스오버, 선택)와 세 가지 동작(즉, 삽입, 배치 및 삭제)에 중점을 둔다. V.P.L. Oliveira et al. [27]은 패치 표현을 세 가지 다른 부분(즉, 편집 작업 유형, 버그가 있을 가능성이 있는 위치, 교체/삽입 코드)으로 나누어, GenProg를 개선했다. 또한 ARJA [28]는 [26]과 [27]을 개선하기 위해 저차원 패치 표현을 제안했다. 이 패치 표현은 수정된 이진수, 작업 유형 및 성분 집합에 대한 세 벡터를 대상으로 한다. 작업 유형과 성분 집합의 값은 무작위 검색을 사용하여 선택된다. ARJA는 또한 두 가지 목표(즉, 패치 크기와

가중 실패율)를 최소화하기 위해 NSGA-II를 사용했다. HECURLES[29]는 AST와 유전 연산 알고리즘을 적용하여 복잡한 멀티 체크 버그 정정을 수행했다. 이 APR기법은 부적절한 패치를 많이 생성할 수 있으므로 시간이 많이 걸릴 수 있다.

다음으로, 패턴 기반 APR은 소스 코드 데이터집합을 수집하고, 그 집합에서 패턴을 마이닝하여 패턴 템플릿을 추출하고, 그 패턴을 적용하여 패치를 생성한다. 이 방법에서는 소프트웨어 변경을 탐색하여 자주 발생하는 버그와 패치를 식별하여 정정 패턴을 정의한다. 이렇게 정의된 프로그램 정정 패턴을 버그 코드 조각에 적용하여 패치를 생성한다. FixMiner [30]는 정정된 버그의 AST 코드 변경 사항을 나타내고 변경 사항의 편집 스크립트를 추출했다. 그리고 그 스크립트를 클러스터링하여 유사한 변경 사항을 식별한 후에, 공통 정정 패턴을 추출했다. SemFix [31]는 패치 생성을 위해 두개 공간 간의 교차점을 사용했다. 첫 번째 공간은 도너(donor) 코드 조각을 사용하고 기존 패치를 마이닝하는 추상적인 검색 공간이다. 반면, 두 번째 공간은 유사한 코드 조각과의 차이를 통해 수정 사항을 추출하는 구체적인 검색 공간이다. TBar [32]는 15개의 카테고리를 결정하고 그 안에서 35개의 패턴을 추출했다. 다음으로, TBar는 선택한 패턴을 사용하여 의심스러운 명령문들을 변이 시켜서 후보 패치를 생성하였다. 그 당시 TBar는 공개 평가를 통하여 패턴 기반 기법에서 SOTA를 달성했다. 패턴 기반 APR의 장점은 다양한 유형의 버그를 정정할 수 있으며, 좀더 관리가능한 코드를 생성할 수 있다. 하지만, 정정을 위한 수정 패턴으로 추출되지 못한 패턴은 처리하지 못하고 패턴 추출은 주로 수작업으로 이루어진다. 또한 버그 정정 패턴 생성을 위하여 많은 시도가 필요하므로 시간이 많이 소요된다.

그리고 의미 기반 APR은 버그 정정 문제를 제약 조건 해결 및 프로그램 합성으로 전환하여 패치를 구성한다. 이러한 유형의 작업은 주어진 테스트 집합을 통과하는 것과 같이 패치된 프로그램이 주어진 정확성 사양을 충족해야 하는 정정 제약 조건을 추론한다. 일반적으로 프로그램 합성을 통해 제약 조건을 해결하면 해당 사양을 충족하는 대체 코드를 구성하고, 다음에 대체 코드를 패치로 사용한다. 제약 조건 해결은 패치 구성 과정에서 “주어진 테스트를 통과할 수 있는 프로그램의 ‘가장 작은’ 수정은 무엇인가?”와 같은 질문을 인코딩하거나, 정정 중인 코드를 기반으로 패치 코드를 제안한다. DirectFix [33]는 전통적인 버그 위치 추정을 수행하고 잠재적인 변경 사항과 관련된 의심스러운 위치를 식별한다. 다음으로 위치와 테스트케이스에 대한 추적 공식을 정의하고, 공식에 대한 정정 조건을 생성한다. 마지막으로 이 조건을 사용하여 위치별 패치를 합성한다. S3 [34]에서는 버그 정정이 두 단계로 구성된다. 첫 단계는 테스트케이스에 대한 동적 기호 실행을 사용하여 예제를 추출한다. 둘째 단계는 도메인별 언어, 검색 절차 및 순위 특징을 사용하여 패치를 합성한다. Angelix [36]은 버그 위치 추정 기법을 실행하여 잠재적인 변경 사항의 상위 k 개의 위치 찾는다. 다음으로 해당 위치에 대해 심볼릭 오라클을 사용하여 패치를 합성한다. Angelix는 C 언어를 대상으로 의심스러운 몇 가지 표현으로 제한하여 확장성 문제를 해결하고자 했다. 그러나 Angelix는 물리적으로 가까운 버그 위치가 필요하지만, 복잡한 멀티 체크 버그 정정을 지원한다. Nopol [35]은 버그 if 조건과 누락된 사전 조건이 있는 버그를 목표로 한다. 먼저, 조건의 예상 값을 식별하기 위해 “Angelix 수정 위치”이라는

버그 위치 추정 방법을 실행한다. 다음으로, 객체 지향 타입과 기초 데이터 타입을 모두 고려한 변수와 그 실제 값을 수집한다. 마지막으로, Nopol은 이러한 수집된 데이터를 인코딩하여 패치를 합성한다.

마지막으로, 학습 기반 APR은 딥러닝 기술의 발전과 함께 신경망을 활용하여 오픈소스 코드에서 버그 수정 패턴을 학습하는 APR 기술이다. 이러한 학습 기반 기술은 일반적으로 APR을 신경망 기계 번역(NMT) 작업으로 처리하며, 소스 버그 코드 조각이 정정 코드 조각으로 자동으로 번역된다. NMT는 RNN, CNN, LSTM, GAN 등의 전형적인 딥러닝 모델을 사용하여 구현된다. SequenceR [37]은 NMT를 구현하고 OOV(out-of-vocabulary) 문제를 해결하기 위해 LSTM과 어텐션 및 복사 메커니즘을 적용했다. SequenceR은 NMT를 사용하여 후보 패치를 생성했지만, 단일 위치 버그만 정정한다. Recoder [38]은 시퀀스 결과를 편집하기 위해 세 개의 인코더로 구성된다. 결과는 추상 구문 트리(AST)를 기반으로 하고 호스트 언어로 작성된다. 첫 번째 인코더와 두 번째 인코더는 각각 버그 메서드와 편집 시퀀스에 대한 AST를 학습한다. 세 번째 인코더는 루트 노드에서 비터미널 노드로 가는 AST 경로를 학습한다. 그 당시 정정 버그 개수에 대해 SOTA를 달성했다. CoCoNut [39]은 양상불 학습을 통해 최적의 모델을 선택하고 NMT를 구현하기 위한 전략을 적용한다. CoCoNut은 20개의 모델을 학습하고 각 모델에 대해 1,000개의 후보 패치를 생성한다. 또한 CoCoNut은 두 개의 인코더(버그 라인을 학습하는 인코더와 버그 문맥을 학습하는 인코더)로 구성되어 입력을 병합하는 문맥 인식 전략을 적용하는데, 두 인코더 모두 CNN을 사용하였다. DLFix[40]는 NMT의 문제점을 해결하기 위해 2 계층의 트리 기반 인코더-디코더(Encoder-Decoder) 모델을 사용한 APR 연구이다. 첫 번째 계층은 버그 정정의 문맥을 학습하며, 두 번째 계층은 이전 버그 정정의 결과를 추가 가중치로 사용하여 학습을 진행한다. DeepFix [41]는 어텐션 인코더-디코더 기법을 사용하여 버그 위치 식별 및 정정 알고리즘을 제안한다. 이 알고리즘은 리눅스 GCC 컴파일러를 사용하여 C-언어 기반 프로그램을 정정하고 컴파일러 오류 메시지 수를 줄인다. DEAR [42]은 멀티 체크 버그 정정에 두 가지 알고리즘을 적용했다. 하나는 데이터 흐름 분석과 스펙트럼 기반 버그 위치 추정을 결합한 체크 탐지 알고리즘이고, 다른 하나는 버그와 의심스러운 문장 사이의 고정 인스턴스를 감지하는 확장 알고리즘이다. 또한 DEAR는 LSTM 기반 사이클 학습 메커니즘을 적용했다. TENURE[43] 접근법은 전통적인 템플릿과 신경망 기반 시퀀스-투-시퀀스 모델을 결합한다. 먼저 코드 정정 템플릿 대규모 데이터집합을 구축하고, 인코더-디코더 NMT 모델을 학습시켜 버그 있는 코드를 수정된 코드로 번역한다. TENURE는 OpenNMT/Transformers를 활용한 공개 모델을 사용하며, 실제 버그-정정 쌍으로 미세조정한다. 생성 과정은 고정 단계이고 모델은 템플릿 단위로 패치를 생성한다.

4.2 LLM 기반 자동 프로그램 정정

대형언어 모델이 자동 프로그램 정정을 새롭게 재구성하고 있다 [8]. LLM 기반의 APR은 2022년부터 급속히 발전하여 많은 관심을 받게 되었고, 최근 코드 중심의 LLM, 예를 들어 OpenAI의 Codex, CodeLlama, DeepSeek-Coder는 코드를 이해하고 생성하는 능력을 입증하여 최소한의 인간 입력으로 프로그램

버그를 수정한다. 초기 연구들에서 LLM 기반의 APR 기술이 정정 벤치마크 자료를 사용한 실험에서 전통적인 정정 방법을 능가하여 보다 우수한 프로그램 정정 성공률을 달성한 것을 보였다. 실제로 새로운 LLM 기반 정정 기법이 놀라운 속도로 등장하고 있으며, 이 기법이 실제로 프로그램 디버깅과 유지보수를 크게 개선할 것이라고 기대한다.



그림 4. LLM 기반 APR

LLM 기반 APR 연구동향은 [8]에서 제시한 기준을 확장하여 기본 LLM모델, 문맥, 미세조정, 프롬프트, 절차적 파이프라인, 에이전트 프레임워크의 6가지 기준으로 소개한다 (그림 4). 기본 모델은 사용한 학습모델, 문맥은 버그 코드에 대한 관련정보이다. 미세조정은 모델과 정정 작업을 긴밀하게 연결하여 훈련 데이터와 유사한 버그에 대해 매우 정확한 패치 생성을 지원한다. 그러나 대규모 버그 정정 데이터와 상당한 GPU 시간이 요구된다. 프롬프트는 추가적인 훈련이 필요하지 않고 신속한 배포를 지원하지만, 그 효과는 세심한 프롬프트 엔지니어링에 의존하며 모델 지식과 문맥 정보 크기에 제한받는다. 절차 파이프라인은 고정된 단계 작업에서 하나 이상의 LLM을 호출한다. 에이전트 프레임워크는 계획과 도구 호출을 LLM 자체에 위임함으로써 복잡성, 지연시간, 그리고 리소스 소비가 증가되지만 멀티 체크 버그 또는 다른 파일에 있는 버그 정정을 지원한다. 이와 같이 한번의 생성 작업에서 반복적인 작업으로 전환하게 되면 복잡한 결함을 정정할 수 있는 가능성을 높인다. 문맥, 미세조정, 프롬프트, 그리고 절차 파이프라인 패러다임은 외부 지식이나 프로그램 분석 피드백을 활용하는 RAG 또는 AAG(Analysis-Augmented Generation) 프롬프트를 통해 강화되고, 이를 통해 의미적 정확성을 개선하고 유효하지 않은 패치를 줄인다. ITER [44]는 버그 위치 확인, 패치 생성, 패치 검증을 여러 번 반복하여 여러 위치에 분포한 버그를 정정한다. 프롬프트에는 추가적인 문맥 정보없이 버그 메서드 정보만을 사용하며, 대량의 데이터집합을 통해 미세조정된 모델(T5)을 통해 후보 패치를 생성한다. SelfAPR[45]는 버그 코드와 실패한 테스트 로그를 함께 입력해 동적 실행 정보를 활용하는 접근 방식이다. 자체 생성한 대규모 버그-정정 데이터를 기반으로 미세조정된 모델(T5)을 활용하여 후보패치를 생성한다. RepairAgent[7]는 GPT-3.5 모델 기반 LLM을 에이전트로 활용하여, 버그 탐색부터 정정, 검증까지 자율적으로 수행한다. 동적 프롬프트 업데이트를 통해 다양한 도구(ANTLR, 테스트 실행 등)를 필요에 따라 호출하여 버그를 정정한다. 모델은 별도의 미세조정 없이 사용되며, zero-shot 프롬프트 방식으로 작동한다.

ChatRepair[46]는 GPT-3.5모델과 대화형 프롬프트를 통해 패치 생성과 테스트 피드백을 반복하여 점진적으로 버그를 수정한다. 테스트 실패 정보를 활용해 같은 오류를 반복하지 않고, 버그를 정정할 수 있도록 유도한다. 시스템 메시지로 LLM에 자동 프로그램 정정이라는 역할을 부여하고, 사용자 메시지를 통해 코드 및 피드백을 주고받는 zero-shot 방식이다. D4C[47]는 GPT-4 모델에 버그 메서드 외에 관련된 테스트케이스, 오류 로그, 주석 등 다양한 정보(AAG)를 함께 제공하여 버그를 정정한다. 버그 위치를 별도로 지정하지 않고 메서드 단위의 버그를 one-shot 프롬프트로 처리하여, LLM이 정정된 메서드를 직접 생성하도록 한다. NTR[48]은 먼저 미세조정된 CodeT5모델을 통해 버그가 입력되면 버그 유형에 맞는 수정 템플릿을 선택한다. 이후 선택된 템플릿은 버그 코드와 함께 프롬프트를 구성하고, 미세조정된 대형 코드 모델인 Codellama-70B를 통해 후보 패치를 생성한다. 템플릿 기반 입력 형식을 통해 패턴에 기반한 정밀한 정정을 유도하고, 두 단계는 순차적으로 진행된다. RAP-Gen[49]은 대규모 버그-정정 쌍을 데이터베이스로 구축하고, 입력된 버그와 어휘적 그리고 의미적으로 유사한 정정 쌍을 검색한다. 검색된 버그-정정 쌍은 프롬프트에 문맥 정보(RAG)로 추가되고 이를 기반으로 미세조정된 모델(CodeT5)을 통해 패치를 생성한다. FitRepair[50]는 CodeT5 기반으로 프로젝트 전체 코드에서 50%를 무작위로 마스킹하여 코드 스타일과 토큰 학습을 강화한 지식강화 미세조정(Knowledge-Intensified Fine-tuning), 정적 분석을 통해 파일 내에서 가장 관련도가 높은 식별자를 선별하여 프롬프트에 포함하는 관련 식별자 프롬프팅(Relevant-Identifier Prompting), 그리고 한 번에 하나의 연속된 코드 조각만 마스킹하여 정적작업에 특화된 정정 지향 미세조정(Repair-Oriented Fine-tuning)을 통해 다양한 문맥관련 정보를 효과적으로 활용하여, 후보 패치를 생성한다. AlphaRepair[51]는 버그가 있는 라인을 토큰으로 대체한 후 메서드 내 주변 문맥 정보를 통해 사전 학습된 모델(CodeBERT)이 적절한 코드를 채워 넣는 방식으로 후보 패치를 생성한다. 모델은 추가적인 미세조정 없이 zero-shot 방식으로 작동하며, 다양한 후보 패치를 탐색하기 위해 Complete Mask, Partial Mask, Template Mask의 세 가지 마스킹 전략을 사용한다. 생성된 후보 패치는 CodeBERT의 확률 기반 점수로 재정렬하여 우선순위가 높은 패치부터 검증한다. Gamma[52]는 기존 템플릿 기반 방법에서 도나(donor) 코드를 검색하던 과정 대신, LLM (UniXcoder, CodeBERT, 또는 ChatGPT가 직접 마스킹된 위치를 채우도록 한다. 사전에 설계된 템플릿 마스크 방식에 따라 버그 코드 일부를 처리하고, 모델은 주변 문맥을 기반으로 빈칸을 채우는 방식으로 후보 패치를 생성한다. 기존 템플릿 기법보다 유연하고 다양한 정정을 할 수 있으며, 미세조정 없이도 높은 성능을 보인다. Toggle[6]은 버그 위치를 정밀하게 예측한 뒤, 해당 위치 정보를 프롬프트에 반영하여 모델(CodeGen, CodeGPT등)이 올바르게 버그를 정정하도록 유도한다. 미세조정된 위치 추정 모델이 토큰 수준에서 버그 위치를 식별하면, 버그 정정 모델은 프롬프트를 기반으로 후보 패치를 생성한다. ThinkRepair[53]는 폐쇄형 소스 LLM(ChatGPT/gpt-3.5-turbo API, CodeLlama, DeepSeek, StarCoder)을 활용하는 2 단계 LLM 기반 자동 프로그램 정정 기법이다. 첫 단계 수집(collecting)에서는 CoT(Chain-of-Thought) 프롬프트를 사용하여 LLM으로부터 추론 과정을 생성하며, 두

번째 단계 정정(fixing)에서는 품질이 높은 예시(few-shot)를 선택하여 LLM과 반복적으로 상호작용하며 패치를 제안하고 개선하며, 필요시 테스트 실패 피드백을 통합한다. 이 방법은 프롬프트 기반의 다중 생성 파이프라인으로, LLM과 외부 테스트 도구가 반복 루프를 돌면서 올바른 패치를 찾는다. 외부 지식을 명시적으로 검색하지 않지만, 테스트 피드백을 프롬프트에 반영해 LLM 추론을 유도하는 분석 보강 생성(AAG) 방식을 활용한다. GiantRepair[54]는 먼저 LLM(폐쇄형 또는 오픈형)으로 패치를 생성하고, 이후 프로그램 분석을 통해 이를 정제하는 하이브리드 방식이다. 실제로는 버그가 있는 함수를 입력받아 GPT-3.5, Llama-2, CodeLlama, StarCoder 등 다양한 LLM으로부터 후보 패치를 생성하고, 각 LLM 출력에서 불필요한 세부 정보를 제거하고 구조만 남긴 스켈레톤으로 변환하여 탐색 공간을 대폭 축소한다. 이후 두 번째 LLM 단계에서 문맥 인식 생성을 수행하여 프로그램 맥락에 맞는 코드를 채워 넣는다. 즉, LLM과 프로그램 분석을 순차적으로 활용하는 고정된 2단계 절차적 파이프라인이며, 사전 학습된 모델을 추가 미세조정 없이 사용하므로 프롬프트 기반이고, 프로그램 분석을 활용해 생성을 유도하는 AAG 방식을 활용한다. Yi. et.al.[55]은 폐쇄형 LLM (GPT-3.5-Turbo 및 GPT-4, OpenAI API 사용) 을 활용하며 정교한 프롬프트 엔지니어링을 수행한다. 이 시스템은 문맥 인식 CoT 프롬프트 루프를 구현하는데, 먼저 실패한 테스트 메시지를 분석해 정정 방향을 추천하고, 해당 방향에 가장 관련 있는 코드 조각, 오류 정보 등 문맥을 자동으로 선택한 뒤, 이를 기반으로 LLM이 수정을 생성하도록 유도한다. 실제로는 LLM을 여러 회차(최대 10회) 실행하며 문맥을 갱신하고 대화형 CoT 전략을 사용하므로, 고정 파라미터 기반(프롬프트 기반) 절차적 방법이다. 각 버그에 맞는 최적의 문맥을 적응적으로 검색하는 RAG를 활용한다. Mulpor[56]는 문장(statement), 수식(expression), 토큰(token) 단위의 세 가지 CodeT5 기반 생성기를 사용하며, 각 생성기는 해당 단위에 맞춰 자기지도 학습으로 사전 학습된 다음에 버그 정정 데이터로 미세조정된다. CodeT5 모델을 활용하며, 마스킹된 코드 구간을 효과적으로 채우는 방식으로 동작한다. 각 단위별로 다중 생성을 수행하고, 이후 유사도 및 테스트 통과 여부를 기반으로 후보 패치를 휴리스틱하게 재순위화하는 절차적 고정 파이프라인이다. Repilot[57]은 공개형 LLM(CodeT5, InCoder)을 코드 의미 기반 자동 완성 엔진(Eclipse JDT)과 결합하여 동작한다. LLM이 다음 토큰을 자동 회귀 방식으로 생성하면, 완성 엔진이 불가능한 토큰을 제거하고, 가능한 토큰이 하나만 남으면 자동으로 완성한다. 이 반복 생성 과정은 프롬프트 기반으로만 이루어지며, 미세조정은 수행되지 않는다. 각 토큰 생성이 LLM과 완성 엔진의 고정된 교차 알고리즘으로 진행되는 다단계 절차적 방식이다. 코드 의미 기반 제약을 실시간으로 적용하는 AAG 방식을 활용한다.

4.3 국내 연구 동향 및 시사점

국내 APR 연구(제1저자 또는 교신저자 소속기관 기준) 동향은 전통적인 버그 정정 연구를 일부 수행하였으나 최신의 LLM 기반 버그 정정 연구는 초기 단계로서 극히 일부이다. 또한 특정 오픈소스 LLM에 국한된다.

SmartFix[58]는 안전이 중요하나 결함이 취약한 스마트계약 정정기술이다. SmartFix는 후보패치를 반복적으로 열거하고 안전성 검증기를 호출하여

정확성을 검증하는 생성 및 검증 방식을 사용하는데, 검증기의 피드백에서 도출된 통계적모델을 사용하여 방대한 검색 공간과 고비용 패치 검증 문제를 해결한다. NPEx[59]는 테스트 없이 Java 널포인트 예외(NPE)를 정정하는 기술이다. NPEx는 기존 코드베이스에서 널 처리 패턴을 마이닝하여 개발자가 NPE 처리 방식을 예측하는 통계적 모델을 학습하고, 이 모델을 사용하여 버그가 있는 프로그램에서 정정 명세를 추론하는 기호 실행을 사용한다. BAPP[60]은 최대 결합 위치 추정 공식을 도출하고 정정작업을 분석하는 베이저안 패치 우선순위 기법이다. BAPP 프레임워크는 핵심 수식을 유도하고 프로그램 값을 통합하여 올바른 패치를 정확하게 식별한다. RepCoder[61]는 입출력 예제에서 프로그램 합성하는 문제를 프로그램 정정 문제로 재구성하는 정정 프레임워크이다. 일련의 입출력 예제와 버그 프로그램이 주어지면 RepCoder는 사용자가 작성한 문장이 신경망 프로그램 합성방법으로 예측된 상위 k 후보문장에 포함되면 정확할 가능성이 높을 것으로 간주하여 검색 공간을 효과적으로 줄인다. Arachne[62]는 심층 신경망(DNN) 프로그램 정정 기법으로서 효과적인 패치 생성을 위한 신경망 가중치를 지역화하고, 차등 진화를 사용하여 지역화된 가중치를 최적화하고 신경망의 오토작을 정정한다. Yang et.al [42]은 딥러닝 알고리즘을 적용하여 버그 위치를 찾아 정정하는 기법을 소개했다. 버그 리포트와 프로그램 소스 코드에서 특징을 추출하여 오토인코더와 합성곱 신경망에 입력하고 순위 점수를 계산하여 버그 위치를 파악한다. 순위 점수가 높은 버그 라인을 토큰들로 구분해 변환하여 Seq-GAN 알고리즘에 입력해서 버그를 정정한다. MCRepair[63], MCRepair++[64]는 멀티 체크 버그를 정정하기 위한 기법으로, 버그가 있는 메서드 내 버그 코드 조각을 버그 블록(buggy block)으로 구성하고, 관련 있는 문맥 메서드를 함께 프롬프트로 구성한다. 미세조정된 모델(CodeBERT)을 통해 후보 패치를 생성하고, 이들을 조합하여 전체 버그를 수정하는 패치 세트를 구성하기 위해 패치 최적화 알고리즘을 적용한다. 또한 검색 효율성을 위해 생성된 패치를 스케줄링하는 연구들이 있는데 [65],[66], Casino[65]는 퍼징 관점에서 테스트 결과를 활용하여 패치된 프로그램을 블랙박스로 보고 스케줄링 한다. Gresino [66]는 프로그램의 내부 상태를 활용하여 효율성을 향상시키는 패치 스케줄링 알고리즘이다. Gresino는 프로그램 실행 중 관찰된 분기의 적중 횟수를 모니터링하고 이를 사용하여 유효한 패치를 검색한다. 그리고 LeakPair[67]는 패턴 기반 APR로서 웹의 SPA(Single Page Application)에서 메모리 누수를 사전에 정정하는 패치를 생성한다. 이러한 사전 예방 기법은 메모리 누수 정정에 필요한 시간과 노력을 크게 줄일 수 있다.

지금까지 여러 APR 기법을 살펴보았는데, 지속적인 도전 과제로서 테스트 스위트를 넘어 의미론적 정확성 검증, 보다 큰 규모인 저장소 수준의 버그 정정, LLM 비용감소 등을 예로 들 수 있다[25]. 특히, APR기법에 관계없이 모든 정정기술은 오류의 실제 근본 원인이 아닌 실패한 테스트로 나타난 증상만을 해결하는 패치인 “오버피팅 패치” 우려가 있다. 간단히 말해, 이전에 실패한 테스트가 이제 통과되었다고 해서 문제의 버그가 항상 정정되는 것은 아니며, 그 차이를 자동으로 알려주는 것은 아직 해결되지 않은 문제이다. 프로그래머들도 때때로 실수로 불완전하거나 부정확한 버그 수정을 하기도 한다. 자동으로 정정된 프로그램은 테스트 집합과 같은 불완전한 프로그램 명세에 대해서는 정확할 수

있지만, 여전히 전역적으로 부정확하거나 개발자가 잘못 판단할 수 있다. 자동 정정 연구의 대부분은 프로그램 동작의 극히 일부 명세에 대해 작동하면 올바른 패치라고 간주한다. 그러나 일부 실제 배포 시나리오에서는 배포전에 개발자가 검토하여 이러한 문제점을 보완할 수 있다. 앞으로, 패치 품질 향상을 위한 수동 또는 자동화된 기술이 자동 프로그램 정정의 핵심이 될 것이다.

5. 벤치마크 데이터집합

5.1 대표적인 공개 데이터집합

자동 프로그램 정정의 성능을 공정하게 비교 평가하기 위해, 다양한 공개 벤치마크 데이터집합이 개발되어 왔다. 가장 널리 사용되는 벤치마크는 Defects4J로, Java 기반의 오픈소스 프로젝트에서 수집된 수백 건의 실제 버그와 그에 대응하는 테스트케이스를 제공하며, 재현 가능한 실험 환경을 구축할 수 있는 특징을 갖는다 [24]. 최근에는 GitHub 저장소에서 최신 결함을 수집하여 만든 GitHub-Java가 제안되었으며, 이 데이터집합은 2023년 이후 발생한 199건의 현대적인 Java 결함을 포함하고 있어, LLM 기반 APR모델의 최신 코드 적용 가능성을 평가하는 데 적합하다 [68]. 한편, C 언어에 대한 대표적인 벤치마크로는 ManyBugs와 IntroClass가 있으며, ManyBugs는 대규모 오픈소스 C 프로그램에서 수집된 실제 버그와 대응 패치를 제공하고, IntroClass는 998개의 학생 과제 프로그램과 각 프로그램의 다수의 버그 버전 및 테스트케이스를 포함한다. 이들은 APR 시스템 성능 평가 뿐만 아니라 소프트웨어 결함 분석 및 교육/연구에도 폭넓게 활용된다 [69]. 또한, BEARS 벤치마크는 CI(Continuous Integration) 환경에서 버그와 패치가 함께 수집된 구조로, Java 프로젝트에서 발생한 251개의 버그-패치 쌍을 포함한다. 이 데이터집합은 코드 변경 이력을 기반으로 수집되었기 때문에 현실적인 버그 수정 시나리오를 제공한다 [70].

5.2 국내외 연구 활용 사례

국외 연구에서는 LLM 기반 APR 성능 검증을 위해 Defects4J, Defects4C, BugsCpp, BEARS 등 다양한 벤치마크가 적극 활용되고 있으며, 특히 일반화 성능 평가와 프롬프트 최적화에 대한 실증 연구가 활발히 수행되고 있다 [6][7][31][71]. Liu et al.[71]은 Defects4J와 BugsCpp 데이터집합을 기반으로 다양한 LLM 모델의 버그 정정 성능을 비교하고, 프롬프트 길이, 버그 유형, 수정 복잡도 등이 성능에 미치는 영향을 정량적으로 분석하였다. 또 다른 사례로, SWE-Synth[72]는 합성 데이터 생성 체계를 통해 버그-정정 쌍, 테스트케이스, 에이전트 추론 트레이스 등을 생성함으로써, 학습 성능 향상 및 모델 추론 단계의 투명성을 제고하는 접근을 제시하였다. SWE-Synth 기반 평가에서는 실제 벤치마크 대비 미세조정 용이성과 패치 생성 다양성이 우수한 것으로 보고되었다. 반면 국내에서는 APR 벤치마크 활용 사례가 상대적으로 제한적이며, 대부분 Defects4J 일부 프로젝트를 사용하여 APR 알고리즘의 PoC 수준의 성능을 측정하는 방식에 머무르고 있다. LLM 기반 연구도 공개 데이터집합보다는 내부 산업 데이터 또는 합성 예제 실험에 치중되고 있으며, 재현성 관점에서 학술적 확장성이 낮다는 지적이 있다 [3][13].

5.3 벤치마크 평가 방식 및 주요 이슈

APR 벤치마크의 평가는 일반적으로 두 가지 관점에서 이루어진다. 하나는 테스트-스위트 적절성 기준으로, 생성된 패치가 테스트 통과하는지를 기준으로 하며, 다른 하나는 패치 정확성 관점으로, 실제 의미적 정정 여부를 판단하는 방식이다. 이 두 기준 사이의 차이는 종종 오버피팅 문제를 유발하는 원인이 된다. 예를 들어, 테스트는 통과하지만 실제로는 버그를 해결하지 못하는 패치가 존재할 수 있으며, 이는 Defects4J 기반 실험에서 자주 지적되어 왔다 [4][73].

LLM 기반 APR 연구에서는 패치의 테스트 통과성뿐 아니라, Top-N 순위 정확도, 의미적 정확성, 저장소 수준 규모 적용 가능성, 추론 단계 수 등을 종합적으로 고려한 평가 방식이 적용되고 있다 [7][8][31]. 특히, LLM이 특정 코드베이스에 과적합되지 않도록 하기 위해, Defects4J-Trans와 같은 코드 변환 기반 벤치마크가 도입되어, LLM의 기억 편향을 제거하고 일반화 성능을 정량화하려는 시도가 보고되었다 [71].

또한, SWE-Synth와 같이 합성 데이터집합 기반 평가는 추론 과정의 단계별 추적성, 에이전트 추론 로그 기록, 다양한 버그 시나리오 포함 등 현실성과 확장성을 모두 고려한 실험 설계를 가능하게 한다. 향후 APR 벤치마크의 신뢰성을 확보하기 위해서는 언어-도메인 다양성 확보, 구성 요소 간 비교 가능성 정립, 재현 가능한 테스트 스크립트 제공 등이 필수적으로 수반되어야 한다 [72].

6. 결론 및 향후 전망

대규모 언어 모델의 등장과 발전은 프로그램 정정 패러다임 분야에 실질적인 기술적 전환점을 제공하고 있다. 본 연구에서는 버그 분류, 버그 위치 추정, 버그 자동 정정이라는 APR의 주요 세 과정을 중심으로 전통 기법과 LLM 기반 기법의 기술 흐름을 비교 분석하였으며(부록 A: 본 연구의 과정을 각각 요약하여 설명), 벤치마크 데이터집합의 활용과 평가 관점에서도 국내외 연구 동향을 소개하였다.

LLM 기반 APR 기술은 기존의 규칙 기반 또는 검색 기반 기법에서 한계를 보이던 복잡한 오류 상황에서도 정밀한 위치 추정 및 자연어 기반 의미 해석을 통해 높은 정확도의 정정을 가능케 하고 있다. 특히, Toggle, RepairAgent, InferFix 등의 사례는 작은 단위 위치 추정과 의미적 정확성 기반 패치 생성을 통합한 구조를 통해 기존 기법 대비 뛰어난 성능을 입증하였다 [6][7][31]. 또한, 다양한 산출물(버그 리포트, 테스트 로그, 디버깅 트레이스 등)을 통합한 LLM 기반 프롬프트 설계, 에이전트 기반 시스템 구조는 APR의 자동화 수준과 실용성을 높이고 있다 [8][22][72]. 그러나 아직 해결해야 할 과제 역시 뚜렷하다.

첫째, LLM은 강력한 생성 능력이 있으나, ‘블랙박스’ 형태로 동작하기 때문에 패치 생성 과정의 해석 가능성 및 신뢰성 확보가 미흡하며, 이는 개발자 수용성과 유지보수 단계 적용에 제한을 초래할 수 있다 [74].

둘째, 대부분의 LLM 기반 APR 연구는 특정 벤치마크(예: Defects4J, BugsCpp 등)에 한정되어 있으며, 특수 도메인(산업용 소프트웨어, 임베디드 시스템, SPL 등)에 대한 성능은 미검증 상태이다. 그래서 LLM의 도메인 일반화와 학습 데이터 다양성 확보가 중요한 연구 과제로 부상하고 있다 [13][69].

셋째, 폐쇄형 LLM의 사용은 반복 재현성과 실험의 장기적 유효성 측면에서

제약을 동반하며, 이에 따라 오픈소스 모델과 재현 가능한 환경 구축이 요구된다 [75]. 특히, 모델 업데이트에 따른 결과의 편차를 줄이기 위해 동적 프롬프트 설계와 버전 관리 전략이 필수적이다. 향후 연구를 위한 방향성은 다음과 같다:

- 구조화된 LLM-APR 파이프라인 설계: 버그 진단, 위치 추정, 패치 생성, 검증을 분리하여 에이전트 기반으로 구조화하고, 사람-반복 작업 설계를 통해 시스템 신뢰성을 제고해야 한다 [7][8][22].
- 산출물 활용 및 도메인 적응: 이슈 설명, 테스트 로그, 코드베이스 메타데이터 등 다양한 입력을 통합하며, SPL-CPS 등 실제 산업 환경에 특화된 학습 구조 및 평가 지표 도입이 필요하다 [13][72].
- 벤치마크 다변화 및 평가 정교화: SWE-Synth, Defects4C, GitBug-Java 등의 다양한 데이터집합을 통해 의미적 정확성, 저장소 수준 확장성 등 실용성 중심 지표를 확장해야 한다 [69][83][72].
- 설명가능성 및 감사 가능성 확보: LLM 기반 정정 과정의 패치 사유 추적성, 반대사실 설명, 에이전트 추론 로그 기반 분석 구조를 도입해야 한다 [75][76].

종합하면, 대규모 언어 모델 기반의 프로그램 정정 기술은 소프트웨어 개발과 유지보수의 생산성 향상뿐만 아니라, 시스템 안정성과 신뢰성을 높이는 핵심 기술로 자리잡을 가능성이 높다. 이를 위해서는 단순 성능 지표를 넘어, 재현성 신뢰성 실용성을 포함하는 종합적 연구와, 학계와 산업계 간의 긴밀한 연계 연구가 필수적이다.

참고문헌

- [1] H. Agh, A. Azamnouri, and S. Wagner, “Software Product Line Testing: A Systematic Literature Review,” *Empirical Software Engineering*, vol. 29, no. 6, p. 146, 2024.
- [2] R. Flores, C. Krueger, and P. Clements, “Mega-scale Product Line Engineering at General Motors,” in *Proceedings of the 16th International Software Product Line Conference-Volume 1*, Sep. 2012, pp. 259-268.
- [3] D. Benavides, S. Segura, and A. Ruiz-Cortés, “Automated Analysis of Feature Models 20 Years Later: A Literature Review,” *Information Systems*, vol. 35, no. 6, 2010, pp. 615-636.
- [4] M. J. Harrold, “Testing: A Roadmap,” in *Proceedings of the Conference on the Future of Software Engineering*, Jun. 2000, pp. 61-72.
- [5] M. Chen et al., “Evaluating Large Language Models Trained on Code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [6] S. B. Hossain et al., “A Deep Dive into Large Language Models for Automated Bug Localization and Repair,” in *Proceedings of the ACM on Software Engineering*, vol. 1, p. 1471-1493, 2024.

- [7] I. Bouzenia, P. Devanbu, and M. Pradel, "RepairAgent: An Autonomous, LLM-Based Agent for Program Repair," arXiv preprint arXiv:2403.17134, 2024.
- [8] B. Yang et al., "A Survey of LLM-based Automated Program Repair: Taxonomies, Design Paradigms, and Applications," arXiv preprint arXiv:2506.23749, 2025.
- [9] J. Anvik, L. Hiew, and G. C. Murphy, "Who Should Fix This Bug?," in Proceedings of the 28th International Conference on Software Engineering, May 2006, pp. 361–370.
- [10] G. Jeong, S. Kim, and T. Zimmermann, "Improving Bug Triage with Bug Tossing Graphs," in Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, Aug. 2009, pp. 111–120.
- [11] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning Bug Reports using a Vocabulary-based Expertise Model of Developers," in Proceedings of 6th IEEE international Working Conference on Mining Software Repositories, May 2009, pp. 131–140.
- [12] P. Bhattacharya, I. Neamtii, and C. R. Shelton, "Automated, Highly-accurate, Bug Assignment using Machine Learning and Tossing Graphs," Journal of Systems and Software, vol. 85, no. 10, 2012, pp. 2275–2292.
- [13] S. Fischer, R. Ramler, W. K. Assunção, C. Egyed, C. Gradl, and S. Auberger, "Model-based Testing for a Family of Mobile Applications: Industrial Experiences," in Proceedings of the 27th ACM International Systems and Software Product Line Conference, Aug. 2023, pp. 242–253.
- [14] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," IEEE Transactions on Software Engineering, vol. 42, no. 8, pp. 707–740, 2016.
- [15] R. Abreu, P. Zoetewij, and A. J. Van Gemund, "Spectrum-Based Multiple Fault Localization," in Proceedings of IEEE/ACM International Conference on Automated Software Engineering, Nov. 2009, pp. 88–99.
- [16] L. Naish, H. J. Lee, and K. Ramamohanarao, "A Model for Spectra-based Software Diagnosis," ACM Transactions on Software Engineering and Methodology, vol. 20, no. 3, 2011, pp. 1–32.
- [17] J. Zhou, H. Zhang, and D. Lo, "Where Should the Bugs Be Fixed? More Accurate Information Retrieval-based Bug Localization based on Bug Reports," in Proceedings of 34th International Conference on Software Engineering (ICSE), Jun. 2012, pp. 14–24.
- [18] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving Bug Localization using Structured Information Retrieval," in Proceedings of 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), Nov. 2013, pp. 345–355.
- [19] S. Wang and D. Lo, "Amalgam+: Composing Rich Information Sources for Accurate Bug Localization," Journal of Software: Evolution and Process, vol. 28, no. 10, 2016, pp. 921–942.
- [20] F. Thung, D. Lo, and L. Jiang, "Automatic Defect Categorization," in Proceedings of 19th Working Conference on Reverse Engineering, Oct. 2012, pp. 205–214.
- [21] P. Chakraborty, M. Alfadel, and M. Nagappan, "BLAZE: Cross-Language and Cross-Project Bug Localization via Dynamic Chunking and Hard Example Learning," IEEE Transactions on Software Engineering, 2025, pp. 1–14.
- [22] Q. Feng, X. Ma, J. Sheng, Z. Feng, W. Song, and P. Liang, "Integrating Various Software Artifacts for Better LLM-based Bug Localization and Program Repair," arXiv preprint arXiv:2412.03905, 2024.
- [23] Y. Li, B. Liu, T. Zhang, Z. Wang, D. Lo, L. Yang, J. Lyu, and H. Zhang, "A Knowledge Enhanced Large Language Model for Bug Localization," in Proceedings of the ACM on Software Engineering (PACMSE), vol. 2, no. FSE, pp. 1914–1936, Jun. 2025, doi: 10.1145/3729356.
- [24] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs," in Proceedings of the 2014 International Symposium on Software Testing and Analysis, Jul. 2014, pp. 437–440.
- [25] C. Le Goues, M. Pradel, A. Roychoudhury, and S. Chandra, "Automatic Program Repair," IEEE Software, vol. 38, no. 4, pp. 22–27, 2021.
- [26] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A Generic Method for Automatic Software Repair," IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 54–72, 2011.
- [27] V. P. L. Oliveira, E. F. Souza, C. Le Goues, and C. G. Camilo-Junior, "Improved Crossover Operators for Genetic Programming for Program Repair," in Proceedings of International Symposium on Search Based Software Engineering, Sep. 2016, pp. 112–127.
- [28] Y. Yuan and W. Banzhaf, "ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming," IEEE Transactions on Software Engineering, vol. 46, no. 10, 2018, pp. 1040–1067.
- [29] S. Saha, "Harnessing Evolution for Multi-Hunk Program Repair," in Proceedings of IEEE/ACM 41st International Conference on Software Engineering (ICSE), May 2019, pp. 13–24.
- [30] A. Koyuncu, K. Liu, T. F. Bissyandé, D. Kim, J. Klein, M. Monperrus, and Y. Le Traon, "FixMiner: Mining Relevant Fix Patterns for Automated Program Repair," Empirical Software Engineering, vol. 25,

- no. 3, 2020, pp. 1980–2024.
- [31] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, “Semfix: Program Repair via Semantic Analysis,” in Proceedings of 35th International Conference on Software Engineering (ICSE), May 2013, pp. 772–781.
- [32] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé, “TBar: Revisiting Template-based Automated Program Repair,” in Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, Jul. 2019, pp. 31–42.
- [33] S. Mechtaev, J. Yi, and A. Roychoudhury, “DirectFix: Looking for Simple Program Repairs,” in Proceedings of IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, May 2015, pp. 448–458.
- [34] X. B. D. Le, D. H. Chu, D. Lo, C. Le Goues, and W. Visser, “S3: Syntax-and Semantic-guided Repair Synthesis via Programming by Examples,” in Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, Aug. 2017, pp. 593–604.
- [35] J. Xuan, M. Martinez, F. DeMarco, M. Clement, S. L. Marcote, and T. Durieux, “Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs,” IEEE Transactions on Software Engineering, vol. 43, no. 1, 2016, pp. 34–55.
- [36] S. Mechtaev, J. Yi, and A. Roychoudhury, “Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis,” in Proceedings of the 38th International Conference on Software Engineering, May 2016, pp. 691–701.
- [37] Z. Chen, S. Komrusch, M. Tufano, L. Pouchet, D. Poshvanyk, and M. Monperrus, “SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair,” IEEE Transactions on Software Engineering, vol. 47, no. 9, 2019, pp. 1943–1959.
- [38] Q. Zhu et al., “A Syntax-guided Edit Decoder for Neural Program Repair,” in Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Aug. 2021, pp. 341–353.
- [39] T. Lutellier, H. V. Pham, L. Pang, Y. Li, M. Wei, and L. Tan, “CoCoNut: Combining Context-aware Neural Translation Models using Ensemble for Program Repair,” in Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Jul. 2020, pp. 101–114.
- [40] Y. Li, S. Wang, and T. N. Nguyen, “DLFix: Context-based Code Transformation Learning for Automated Program Repair,” in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Jun. 2020, pp. 602–614.
- [41] R. Gupta, S. Pal, A. Kanade, and S. Shevade, “DeepFix: Fixing Common C Language Errors by Deep Learning,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, no. 1, Feb. 2017.
- [42] G. Yang, K. Min, and B. Lee, “Applying Deep Learning Algorithm to Automatic Bug Localization and Repair,” in Proceedings of the 35th ACM Symposium on Applied Computing, Mar. 2020, pp. 1634–1641.
- [43] X. Meng et al., “Template-based Neural Program Repair,” in Proceedings of IEEE/ACM 45th International Conference on Software Engineering (ICSE), May 2023, pp. 1456–1468.
- [44] H. Ye and M. Monperrus, “ITER: Iterative Neural Repair for Multi-Location Patches,” in Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, Feb. 2024, pp. 1–13.
- [45] H. Ye, M. Martinez, X. Luo, T. Zhang, and M. Monperrus, “SelfAPR: Self-supervised Program Repair with Test Execution Diagnostics,” in Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Oct. 2022, pp. 1–13.
- [46] C. S. Xia and L. Zhang, “Automated Program Repair via Conversation: Fixing 162 out of 337 Bugs for \$0.42 Each using ChatGPT,” in Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Sep. 2024, pp. 819–831.
- [47] J. Xu, Y. Fu, S. H. Tan, and P. He, “Aligning the Objective of LLM-based Program Repair,” arXiv preprint arXiv:2404.08877, 2024.
- [48] K. Huang, J. Zhang, X. Meng, and Y. Liu, “Template-Guided Program Repair in the Era of Large Language Models,” in Proceedings of IEEE/ACM 47th International Conference on Software Engineering (ICSE), Oct. 2024, pp. 367–379.
- [49] W. Wang, Y. Wang, S. Joty, and S. C. Hoi, “Rap-Gen: Retrieval-Augmented Patch Generation with CodeT5 for Automatic Program Repair,” in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Nov. 2023, pp. 146–158.
- [50] C. S. Xia, Y. Ding, and L. Zhang, “The Plastic Surgery Hypothesis in the Era of Large Language Models,” in Proceedings of 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Sep. 2023, pp. 522–534.
- [51] C. S. Xia and L. Zhang, “Less Training, More Repairing Please: Revisiting Automated Program Repair via Zero-shot Learning,” in Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Nov. 2022, pp. 959–971.
- [52] Q. Zhang, C. Fang, T. Zhang, B. Yu, W. Sun, and Z. Chen, “Gamma:

Revisiting Template-Based Automated Program Repair via Mask Prediction,” in Proceedings of 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Sep. 2023, pp. 535–547.

- [53] X. Yin, C. Ni, S. Wang, Z. Li, L. Zeng, and X. Yang, “ThinkRepair: Self-Directed Automated Program Repair,” in Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Sep. 2024, pp. 1274–1286.
- [54] F. Li, J. Jiang, J. Sun, and H. Zhang, “Hybrid Automated Program Repair by Combining Large Language Models and Program Analysis,” *ACM Transactions on Software Engineering and Methodology*, 2025.
- [55] Y. Li, M. Cai, J. Chen, Y. Xu, L. Huang, and J. Li, “Context-aware Prompting for LLM-based Program Repair,” *Automated Software Engineering*, vol. 32, no. 2, p. 42, 2025.
- [56] B. Lin, S. Wang, M. Wen, L. Chen, and X. Mao, “One Size Does not Fit All: Multi-granularity Patch Generation for Better Automated Program Repair,” in Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Sep. 2024, pp. 1554–1566.
- [57] Y. Wei, C. S. Xia, and L. Zhang, “Copiloting the Copilots: Fusing Large Language Models with Completion Engines for Automated Program Repair,” in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Nov. 2023, pp. 172–184.
- [58] S. So and H. Oh, “SmartFix: Fixing Vulnerable Smart Contracts by Accelerating Generate-and-Verify Repair using Statistical Models,” in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Nov. 2023, pp. 185–197.
- [59] J. Lee, S. Hong, and H. Oh, “NPEx: Repairing Java Null Pointer Exceptions without Tests,” in Proceedings of the 44th International Conference on Software Engineering, May 2022, pp. 1532–1544.
- [60] S. Kang, W. Choi, and S. Yoo, “A Bayesian Framework for Automated Debugging,” in Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Jul. 2023, pp. 880–891.
- [61] S. Ji, S. M. Choi, S. K. Ko, D. Kim, and H. Im, “RepCoder: An Automated Program Repair Framework for Probability-Based Program Synthesis,” in Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, Apr. 2022, pp. 1554–1561.
- [62] J. Sohn, S. Kang, and S. Yoo, “Arachne: Search-Based Repair of Deep Neural Networks,” *ACM Transactions on Software Engineering*

and Methodology, vol. 32, no. 4, 2023, pp. 1–26.

- [63] J. Kim and B. Lee, “MCRepair: Multi-Chunk Program Repair via Patch Optimization with Buggy Block,” in Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Mar. 2023, pp. 1508–1515.
- [64] J. Kim, A. S. Abdinabiev, and B. Lee, “Improving Patch Optimization With Buggy Block for Complex Program Repair,” *IEEE Access*, vol. 13, Feb. 2025, pp. 30420–30443.
- [65] Y. Kim, S. Han, A. Y. Khamit, and J. Yi, “Automated Program Repair from Fuzzing Perspective,” in Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Jul. 2023, pp. 854–866.
- [66] Y. Kim, Y. Park, S. Han, and J. Yi, “Enhancing the Efficiency of Automated Program Repair via Greybox Analysis,” in Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, Oct. 2024, pp. 1719–1731.
- [67] A. Shahoor, S. Abdylidayev, H. Hong, J. Yi, and D. Kim, “Proactive Debugging of Memory Leakage Bugs in Single Page Web Applications,” *IEEE Transactions on Software Engineering*, vol. 51, iss. 7, 2025, pp. 1969–1995.
- [68] A. Silva, N. Saavedra, and M. Monperrus, “Gitbug-Java: A Reproducible Benchmark of Recent Java Bugs,” in Proceedings of the 21st International Conference on Mining Software Repositories, Apr. 2024, pp. 118–122.
- [69] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Weimer, “The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs,” *IEEE Transactions on Software Engineering*, vol. 41, no. 12, Dec. 2015, pp. 1236–1256.
- [70] F. Madeiral, S. Uri, M. Maia, and M. Monperrus, “BEARS: An Extensible Java Bug Benchmark for Automatic Program Repair Studies,” in 2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER), Feb. 2019, pp. 468–478.
- [71] J. Sun, F. Li, X. Qi, H. Zhang, and J. Jiang, “Empirical Evaluation of Large Language Models in Automated Program Repair,” *arXiv preprint arXiv:2506.13186*, Jun. 2025.
- [72] M. V. Pham et al., “SWE-Synth: Synthesizing Verifiable Bug-Fix Data to Enable Large Language Models in Resolving Real-World Bugs,” *arXiv preprint arXiv:2504.14757*, 2025.
- [73] X. B. D. Le, F. Thung, D. Lo, and C. Le Goues, “Overfitting in Semantics-based Automated Program Repair,” in Proceedings of the 40th International Conference on Software Engineering, May 2018, p. 163–163.

- [74] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You? Explaining the Predictions of Any Classifier," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 2016, pp. 1135–1144.
- [75] "Open LLM Leaderboard," HuggingFace, 2024. Accessed: Aug. 9, 2025. [Online]. Available: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
- [76] A. Srivastava et al., "Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models," Transactions on Machine Learning Research, 2023.

부록 A. 본 연구의 과정별 정성 비교

항목	버그 분류	버그 위치 추정	버그 정정
입력 데이터	버그 리포트 (요약, 본문)	버그 리포트, 테스트 로그, 소스 코드, 스택 트레이스	버그 리포트, 코드, 테스트 로그, 오류 로그 등
전통적 접근 방식	TF-IDF, LDA, SVM, 코사인 유사도, 토픽 모델링, Tossing Graph	SBFL (Tarantula, Ochiai), IR 기반 (BugLocator 등)	GenProg, ARJA, TBar, FixMiner, SemFix, Angelix, SequenceR, DLFix
LLM 적용 방식	CodeBERT, RAG, Agent 기반 분류, 지칭 튜닝, 한국어 PLM	Toggle, BLAZE, DEVLoRe, Knowledge-enhanced LLM	RepairAgent, ChatRepair, ThinkRepair, FitRepair, GiantRepair, D4C 등
대표 기술 예시	CodeBERT, KoBERT, Developer Recommendation 모델	Toggle, DEVLoRe, BLAZE	RepairAgent, ChatRepair, SelfAPR, FitRepair
국내 연구 특성	온프레미스 LLM, 한국어 대응, SPL 특화 분류기	정적 분석 연계, SPL clone 분석, 일부 PoC 실험	CodeBERT 기반, 멀티칭크 정정, SPL 실험, 패치 스케줄링
기술적 장점	다중 작업 가능, 개발자 추천 정확도 개선	정밀 위치 추론, 다양한 입력 지원	복잡한 버그 대응, 테스트 기반 검증 가능
기술적 한계	다언어 문제, 데이터 부족, 평가 기준 미정립	일반화 미흡, 벤치마크 부족, 실험 환경 적용 한계	Overfitting, Black-box 문제, 비용/지연 시간
시사점	한국어 벤치마크 필요, 설명가능한 설계 필요	로그/테스트 등 다양한 입력 활용 필요	설명가능성 확보, 분석 연계 하이브리드 구조



신진연구자 소개

고려대학교 안가빈 교수



• 안가빈 박사 (Dr. Gabin An)

조교수
고려대학교 컴퓨터학부
gabin_an@korea.ac.kr

주요 약력

- 2025.09~ 고려대학교 정보대학 컴퓨터학과 조교수
- 2024.06~2025.07 Roku Inc., Software Engineer in Test
- 2020.03~2024.08 KAIST 전산학부 박사 (지도교수: 유신)
- 2018.02~2020.02 KAIST 전산학부 석사 (지도교수: 유신)
- 2012.02~2018.02 KAIST 전산학부 학사

주요 연구 분야

- 자동화된 소프트웨어 테스트 및 디버깅
- 소프트웨어 공학을 위한 인공지능

대표논문

- G. An, J. Yoon, T. Bach, J. Hong, and S. Yoo, "Just-in-Time Flaky Test Detection via Abstracted Failure Symptom Matching," in Proceedings of the 40th International Conference on Software Maintenance and Evolution (ICSME'24), Oct. 2024, Industry Track.
- S. Kang *, G. An *, and S. Yoo, "A Quantitative and Qualitative Evaluation of LLM-based Explainable Fault Localization," in Proceedings of the ACM on Software Engineering and Volume 1, Number FSE (FSE'24), Jul. 2024, Research Track
- G. An, J. Hong, N. Kim, and S. Yoo, "Fonte: Finding Bug Inducing Commits from Failures," in Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE'23), May 2023, Technical Track.
- G. An and S. Yoo, "FDG: A Precise Measurement of Fault Diagnosability Gain of Test Cases," in Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'22), Jul. 2022, Technical Track
- G. An *, J. Yoon *, J. Sohn, J. Hong, D. Hwang, and S. Yoo, "Automatically Identifying Shared Root Causes of Test Breakages in SAP HANA," in Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE'22), May 2022, SEIP Track.

소개글

안녕하세요. 이번 학기부터 고려대학교 정보대학 컴퓨터학과에 조교수로 부임하게 된 안가빈입니다. 이렇게 소프트웨어 소사이어티 소식지를 통해



Figure 1. Roku에 근무할 때, SUNY Korea Campus에서 사내 자동 테스트 인프라에 대한 세미나를 진행하였습니다.

인사드릴 수 있어 무척 기쁘고 영광스럽게 생각합니다.

저는 KAIST 전산학부에서 유신 교수님의 지도하에 석사 및 박사과정을 마쳤습니다. 대학원 시절에는 자동 디버깅을 중심으로, 실용적이면서도 산업 현장에 적용 가능한 효율적인 방법론에 대해 연구해 왔으며, 특히 박사 말년 차부터는 대규모 언어 모델(LLM)을 활용한 소프트웨어 엔지니어링 분야에 관심을 가지고 연구를 진행하고 있습니다.

대학원 내내 아무래도 "소프트웨어 공학"분야를 연구하다 보니 실제 산업 현장에 근무하며 개발자들이 어떠한 고충을 겪고, 어떤 문제에 직면 하는지, 어떤 부분이 개발과 유지 보수 과정의 병목이 되는지 가까이서 관찰하고 직접 부딪혀보고 싶다는 생각이 많이 들었고, 이러한 궁금증을 해소하기 위해 졸업 후 미국 IT 기업 Roku Inc.의 서울 오피스에서 소프트웨어 엔지니어 (Software Engineer in Test) 로 1년간 근무하였습니다.

Q. Roku Inc.에서의 근무 경험에 대해 소개해주세요.

Roku Inc.는 2002년에 설립된 기업으로, 초기에는 Netflix의 Project Griffin 하드웨어 기술을 계승하여 2008년부터 독립적인 스트리밍 기기 및 스마트 TV 플랫폼 회사로 자리잡았습니다. 현재는 미국 및 북미 지역의 Connected TV 시장에서 약 40%의 점유율을 차지하고 있습니다. 많은 분들이 Roku가 한국에서 서비스를 제공하지 않는데도 왜 서울 오피스가 있는지 궁금해하시는데요, KAIST 졸업생들이 주축이 된 Enswers라는 회사가 여러 차례 인수 과정을 거쳐 Roku에 합류하면서 서울 오피스가 설립된 것으로 알고 있습니다.

저는 Roku에서 Software Engineer in Test로 근무하며, 자동화된 테스트 인프라를 구축하고 유지하는 팀에서 일했습니다. 대학원에서의 연구 주제와 밀접하게 연결된 업무를 맡을 수 있어 만족도가 높았습니다. 보안상의 이유로 구체적인 내용을 공유하기는 어렵지만, 소프트웨어 실행 환경에서 발생하는

문제를 사전에 감지하고, 근본 원인을 분석하기 위한 라이브러리 및 플랫폼을 설계·구현하는 독립 프로젝트를 주도했습니다.

Roku에서 일하면서 가장 좋았던 점은 제가 맡은 프로젝트에 대해 미국 본사 및 영국 오피스의 엔지니어, 매니저들과 자주 소통하며 피드백을 받을 수 있었다는 점입니다. 오피스가 여러 나라에 분산되어 있어 이른 아침이나 늦은 밤 회의를 해야 할 때도 있었지만, 글로벌한 환경 속에서 협업하는 경험이 즐겁고 새로웠습니다. 또한 현업 엔지니어들에게 프로젝트를 제안하고 피드백을 받는 과정을 통해, 연구 과정에서 세우곤 하는 단순한 가정들이 산업 현장에서는 반드시 통하지 않을 수 있다는 점을 실감할 수 있었습니다. 반대로, 학계에서 활발히 연구되고 실제로 적용 가능한 기술조차 산업계에는 아직 충분히 전달되지 않았거나, 그 존재조차 잘 알려지지 않은 경우도 많다는 것을 깨달으며 학계와 산업계 간의 간극을 체계화하게 되었습니다. 앞으로 산학 협력을 꾸준히 이어가며 이러한 간극을 점차 좁혀 나가고, 학계의 지식과 기술을 산업 현장에 효과적으로 전달할 수 있는 방식에 대해 지속적으로 고민하고 실천해 나갈 수 있다면 참 좋을 것 같습니다.

Q. 최근 연구 중인 분야에 대해 말씀해주세요.

최근에는 LLM을 활용한 소프트웨어 엔지니어링 연구, 특히 소프트웨어 신뢰성 향상을 위한 자동화 기법 개발에 집중하고 있습니다. 최근 몇 년간 LLM은 그 크기와 성능 면에서 비약적인 발전을 이루었고, 단순한 코드 생성 뿐 아니라 테스트 생성, 디버깅, 문서 작성 등 다양한 소프트웨어 엔지니어링 작업에 적용되며 큰 주목을 받고 있습니다. 소프트웨어는 본질적으로 언어적 구조로 이루어져 있으며, 개발자와 시스템 간의 상호작용은 자연어(NL)와 프로그래밍 언어(PL)의 이중 채널을 통해 이루어집니다. 기존에는 이 두 언어 사이의 의미를 연결하고 조율하는 역할을 전적으로 사람이 수행했지만, 이제는 LLM이 그 역할을 일부 혹은 상당 부분 대체할 수 있을 만큼 발전했습니다. 저는 이러한 PL과 NL의 경계에서 LLM이 어떤 방식으로 기여할 수 있는지에 큰 관심을 가지고 연구를 이어가고 있습니다.

예를 들어, 최근 FSE 2024 논문에서 제안된 AutoFL 기법은 예러 메시지와 같은 자연어적 증상 정보와 실행 커버리지 등의 동적 정보를 기반으로, LLM이 버그 위치를 진단하고 그 이유를 자연어로 설명하도록 하는 시스템을 개발하였습니다. 이때 단순한 단편적 추론에 그치지 않고, LLM이 다양한 함수 호출을 통해 실제 코드와 주석을 탐색하며 자율적으로 디버깅을 수행하도록 구성하였습니다. 또한, LLM의 응답 일관성을 기반으로 한 self-consistency 평가 방식으로 응답의 신뢰도를 정량적으로 측정할 수 있는 구조를 갖추었습니다.

또 다른 연구인 Metamon (LLM4Code'25)에서는 소스코드와 문서 사이의 불일치를 자동으로 탐지하는 기법을 제안했습니다. 이때, 평가하고자 하는 문서와 함께 자동 생성한 테스트 케이스를 LLM에게 제공하고, 이를 다양한 방식으로 변형하여 질문하는 metamorphic prompting을 도입하였습니다. 이렇게 반복적인 질문-응답의 흐름에서 발생하는 응답의 패턴과 변화 양상을 분석함으로써, 문서와 실제 코드 간의 불일치를 더욱 정확하게 감지할 수 있었습니다.

이처럼 저는 LLM이 소프트웨어의 언어적 이중성(NL-PL)을 이해하고 매개하는 능력을 활용하여, 디버깅이나 문서 검증과 같은 신뢰성 중심의 자동화 기법을 설계하고자 노력하고 있습니다. 저는 LLM 기반 기술이 개발자의 업무를 대체하기

보다는, 반복적이고 수고로운 작업을 줄여줌으로써 개발자가 보다 창의적이고 본질적인 문제에 집중할 수 있도록 도와주는, 인간과 코드 사이의 효율적인 가교 역할을 수행하게 되기를 기대합니다.

Q. 향후 연구 방향이나 포부에 대해 말씀해주세요.

LLM은 매우 유용하고 강력한 도구이지만, 그만큼 신중하게 다뤄야 할 요소도 많습니다. 특히, 그럴듯하지만 사실과 다른 정보를 생성하는 Hallucination 현상은 소프트웨어 개발에서 심각한 위험 요소가 될 수 있습니다. LLM의 출력을 무비판적으로 신뢰할 경우, 오히려 오류를 확대하거나 불필요한 비용을 초래할 수 있기 때문입니다.

앞으로 제가 집중하고자 하는 방향은, 첫째로 “LLM의 응답을 어떻게 자동적이고 정량적으로 검증할 수 있을 것인가”에 대한 체계적인 방법론을 다양한 소프트웨어 공학 문제에 대해 연구해보는 것입니다. 예를 들어, 앞서 소개한 Metamon 연구에서는 코드의 실제 동작을 테스트를 통해 관찰하고, 이를 문서와 다양한 방식으로 비교함으로써 LLM의 응답 일관성을 보다 정밀하게 검증하는 구조를 제안했습니다. 이처럼 응답 신뢰도를 정량적으로 추정하는 방향은 앞으로 더욱 보편화될 LLM 중심 환경에서 더 안전하고 신뢰할 수 있는 활용을 가능하게 할 중요한 연구 주제라고 생각합니다.

두 번째로, LLM을 포함한 AI 기술의 도입은 소프트웨어의 내부 구조에도 근본적인 변화를 가져오고 있습니다. 기존의 소프트웨어는 동일한 입력에 대해 항상 같은 출력을 내는 deterministic한 구조를 따랐지만, 최근에는 AI 모델이 중간에 개입하면서 실행 결과가 매번 달라질 수 있는 nondeterministic한 시스템이 늘어나고 있습니다. 이러한 변화는 기존의 자동 테스트나 디버깅 기법과 충돌을 일으킵니다. 예를 들어, 전통적인 회귀 테스트는 동일한 입력에 대해 동일한 출력을 기대하지만, AI가 포함된 시스템에서는 실행 결과가 달라질 수 있어 오탐을 유발하거나 테스트의 신뢰도를 떨어뜨릴 수 있습니다. 나아가, 이렇게 다양하게 생성되는 출력의 옳고 그름이나 품질을 자동으로 판단하는 일 자체도 여전히 어려운 문제입니다. (이는 앞서 언급한 LLM 응답의 신뢰도 검증 문제와도 밀접한 관련이 있습니다.) 저는 이러한 AI-enabled 소프트웨어를 위한 새로운 테스트 및 디버깅 기법을 연구하고자 합니다. 이는 단순히 기존 기법을 보완하는 차원을 넘어 소프트웨어의 정의와 품질 개념 자체가 변화하는 흐름에 대응하기 위해 꼭 필요한 방향이라고 생각하며, 그 과정에서 연구자로서 의미 있는 기여를 할 수 있기를 기대합니다.

Q. 마지막으로 소식지를 읽는 분들께 한 말씀 부탁드립니다.

저는 소프트웨어 공학이 시대의 변화와 함께 끊임없이 진화해 나가는, 매우 매력적인 연구 분야라고 생각합니다. 2025년의 소프트웨어가 2015년과 크게 달라졌듯, 10년 뒤의 소프트웨어는 또 어떤 모습일지 상상해보는 것만으로도 무척 흥미롭고 기대가 됩니다. 저 역시 이제 막 독립 연구자로서 첫발을 내디뎠습니다. 앞으로 한국의 많은 소프트웨어 공학 연구자분들과 활발히 교류하며, 새롭게 등장하는 문제들과 시대를 초월한 본질적인 문제들을 함께 고민하고 해결해 나가고 싶습니다. 앞으로 자주 뵙겠습니다. 감사합니다.



딥러닝 라이브러리를 하얗게 테스트하기 위한 경량 Concolic Testing의 첫걸음



• 김미정 부교수
울산과학기술원 컴퓨터공학과
mijungk@unist.ac.kr

본 원고에서는 지난 5월 캐나다 오타와에서 열린 소프트웨어공학 분야의 대표 국제학회인 ICSE 2025에서 발표된 “Lightweight Concolic Testing via Path-Condition Synthesis for Deep Learning Libraries” 논문 (이하 PathFinder)을 소개하고자 한다. 이 논문의 도구 이름은 PathFinder로, 딥러닝 라이브러리의 신뢰성을 높이기 위한 새로운 테스트 기술을 제안한다. 이는 기존의 블랙박스 기법에 의존하던 딥러닝 테스트 방식에서 한 걸음 더 나아간 시도라 할 수 있다. 딥러닝 라이브러리는 내부 구조가 복잡하고 수치 연산이 중심이기 때문에, 코드 내부 구조를 기반으로 한 테스트가 현실적으로 어려워, 그간 대부분의 테스트 연구가 블랙박스 기법에 머물러 있었다. 그러나 블랙박스 접근은 테스트 커버리지나 버그 탐지 효율에 한계가 있어, 보다 고도화된 테스트 방식에 대한 수요가 커지고 있다.

본 연구는 경량화된 Concolic Testing 기법을 활용하여, 수작업 없이 완전히 자동화된 추론기반 화이트박스 분석을 현실화하는 데 성공하였다. 특히 복잡한 수치 연산자 기반의 코드에 적용하면서도, 테스트 효율성과 정밀성 사이의 균형을 유지하는 데 초점을 맞췄다. 이 글에서는 해당 연구의 배경과 핵심 아이디어, 주요 성과, 그리고 연구 과정에서의 고민과 소회를 간략히 소개하고자 한다.

1. 연구의 시작과 발전 과정

이 연구는 필자가 미국 Purdue University에서 박사후연구원으로 근무하던 시기에 구상되기 시작했다. 그 당시 필자는 DocTer라는 연구를 진행하고 있었고, 이는 딥러닝 라이브러리의 API 문서에서 입력 전제조건을 추출하여 블랙박스 방식으로 테스트하는 방법을 제안한 연구이다. 이 연구는 FreeFuzz이라는 논문과 함께 딥러닝 라이브러리 테스트 분야에서 최초 연구 중 하나로 많은 주목을 받았으며, 이후 이 분야에 대한 장기적인 연구를 결심하게 되는 계기가 되었다.

사실 DocTer 연구 자체도 우여곡절이 많았다. 딥러닝 라이브러리는 그 당시만 해도 생소하고 다루기 어려운 분야였고, 관련 테스트 기법도 정립되지 않아 아이디어를 설명하고 설득하는 일이 쉽지 않았다. 연구는 2020년에 시작했지만, 여러 차례 학회 심사에서 거절을 겪은 끝에 2022년에 가서야 논문으로 출판할 수 있었다. 어려움이 있었지만, AI 기술이 다양한 분야에서 빠르게 확산되던 시기였고, 딥러닝 인프라의 신뢰성을 보장하는 일은 분명 중요한 문제라고 판단했다. 이 주제를 장기적인 연구 과제로 삼기로 마음먹은 이유이다.

그런 의미에서, PathFinder는 DocTer의 연장선이자 진화형에 가깝다. 실제로 DocTer를 시작점으로 이후 학계에서도 딥러닝 라이브러리를 테스트하려는 다양한 시도들이 이어졌고, 관련 논문들도 빠르게 증가했다. 블랙박스 수준을 넘어서 코드 내부 구조에 접근하고자 하는 요구가 생겨나던 흐름 속에서

PathFinder는 “화이트박스에 가까운 경량 분석”이라는 대안을 제시하였다.

PathFinder 연구가 본격화된 계기는 다소 우연한 자리에서 찾아왔다. 같은 학교의 이주용 교수님의 출장길, 기차 안에서 연구 이야기를 나누던 중, 필자가 기존 기법의 한계를 설명하자 이 교수님이 툭 던진 한마디가 매우 신선한 아이디어로 느껴졌고, 그 자리에서 바로 미팅을 잡고 협업을 시작하게 되었다. 아이디어는 있었지만, 구현과 성능 확보는 또 다른 과제였다. 본 논문의 주된 실험과 알고리즘 구체화 및 최적화 과정에서 1저자인 김세훈 학생이 수차례 알고리즘을 다듬고 실험을 반복해준 덕분에, 지금의 PathFinder가 완성될 수 있었다. 이 연구는 단지 하나의 논문으로 끝나는 작업이 아니라, 기존 문제의식의 연장선 위에서 공동연구와 학생의 주도적 기여가 더해져 만들어진 성과라고 할 수 있다.

2. 딥러닝 라이브러리 테스트의 연구 현황과 도전

딥러닝 라이브러리는 현대 인공지능 응용의 핵심 기반으로 자리 잡고 있다. PyTorch, TensorFlow와 같은 라이브러리는 고성능 수치 연산자, 복잡한 텐서 구조, 다양한 백엔드 디바이스 연동 기능 등을 포함하고 있어, 그 자체가 하나의 복합 시스템이라고 할 수 있다. 하지만 이처럼 방대하고 복잡한 구조는 곧 테스트의 어려움으로 이어진다.

실제로, 딥러닝 라이브러리에 대한 테스트 연구는 대부분 주어진 API에 다양한 입력을 생성해 실행해보는 블랙박스 테스트 방식이 주로 사용되어 왔다. 이러한 접근은 도입이 비교적 쉽고 확장성이 뛰어나다는 장점이 있지만, 코드 내부 로직에 대한 이해 없이 테스트가 이루어진다는 근본적인 한계를 갖는다.

특히 다음과 같은 문제가 지적되어 왔다:

- 테스트 커버리지가 낮고, 특정 경로만 반복적으로 탐색될 가능성이 높다.
- 발견된 버그가 내부 로직의 어떤 조건에서 발생하는지 파악하기 어렵다.
- 입력 다양성은 확보되지만, 실행 경로 다양성은 제한적이다.

이러한 한계로 인해 딥러닝 라이브러리 테스트의 품질을 한층 끌어올리기 위해서는, 라이브러리 코드 내부 구조를 부분적으로라도 활용할 수 있는 보다 정교한 접근이 필요하다. 그러나 현실적으로 딥러닝 코드베이스는 방대한 규모와 복잡한 수치 연산 흐름 등으로 인해 기존 화이트박스 분석 방식, 즉 concolic testing 방식을 직접 적용하기는 매우 어렵다.

바로 이 지점에서 본 연구는 대안을 모색한다. 내부 구조를 완전히 분석하지는 않더라도, 실행 경로에 간접적으로 개입하거나 변형할 수 있는 전략이 있다면, 블랙박스와 화이트박스 사이의 중간 지점, 즉 “추론 기반 화이트박스 분석(inferred whitebox analysis)”과 같은 접근이 가능하지 않을까?

PathFinder는 이러한 문제의식에서 출발하여, 기존 concolic testing이 지닌 적용 규모 (scalability) 제약을 피하면서도 실행 경로의 다양성과 의미 있는 테스트 입력 생성을 가능하게 하는 새로운 접근을 제안한다.

3. 논문의 핵심 아이디어

PathFinder는 딥러닝 라이브러리의 API를 대상으로, concolic testing의 정확성과 블랙박스 테스팅의 확장성 사이의 간극을 메우는 새로운 테스팅 기법을 제안한다. 핵심 아이디어는 실행 경로에 대한 정확한 조건식을 직접 수식화하지 않고, 입력과 분기 결과의 관계를 반복 실행을 통해 점진적으로 학습하는 “분기 조건 합성(path-condition synthesis)” 전략이다.

기존의 concolic testing 기법은 프로그램을 실제 입력값으로 실행하면서 내부 조건문 (분기 조건)을 수식으로 추출하고, 그 수식을 바탕으로 새로운 경로를 유도할 입력을 SMT solver 를 사용하여 계산한다. 하지만, 딥러닝 라이브러리의 API는 고차원 텐서와 비선형 (non-linear) 계산이 많아, 이런 조건식을 정확히 추출하는 것도 어렵고, 추출한다고 하더라도 이러한 조건을 풀 수 있는 SMT solver도 마땅치 않다. 결과적으로, 딥러닝 API에 기존 concolic testing 접근을 적용하는 것은 테스트 생성 실패율이 높고 실용성이 낮다.

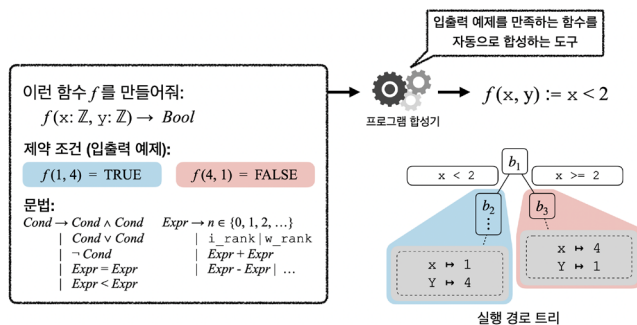


그림1. PathFinder가 분기 조건을 합성하는 방식을 설명하는 예시

PathFinder는 이러한 문제를 해결하기 위해, 내부 분기 조건을 직접 수식화하지 않고 입력과 실행 결과 간의 관계를 관찰하여 분기 조건을 추론하는 접근을 택한다. API를 여러 입력값으로 실행하면서 입력값과 분기 결과(True/False)의 쌍을 수집하고, 이 데이터를 기반으로 “어떤 입력이 어떤 경로로 이어지는지”를 기록한다. 예를 들어, 그림1 처럼, 입력값 $x=1$ 일 때 어느 분기 조건이 True였고, $x=4$ 일 때는 False였다면, 프로그램 합성기는 이를 기반으로 $x < 2$ 일 수 있다는 조건을 추론해 낸다. 이렇게 추론된 조건을 활용해 아직 탐색되지 않은 분기, 즉 $x \geq 2$ 를 유도할 새로운 입력을 SMT solver 를 활용해 생성한다.

이런 식으로 축적된 실행 기록을 바탕으로, PathFinder는 분기 조건에 대한 추정식을 점차 정제 (refine)해 나간다. PathFinder 가 활용하는 귀납적 프로그램 합성기¹⁾는 복잡한 symbolic expression을 요구하지 않으며, 단순한 수치 비교 기반 로직으로 구성되어 있기 때문에 합성된 조건이 비교적 단순하여 계산 비용이

적고, 딥러닝 API 같은 복잡한 코드에도 안정적으로 동작할 수 있다.

중요한 점은, PathFinder가 기존의 concolic testing 처럼 “실제” 조건식을 기호적으로 “분석(analyze)”하는 것이 아니라, 입출력 관찰을 기반으로 “근사적인 (approximate)” 조건식을 “합성(synthesize)”한다는 점이다. 기존 concolic testing 기법이 복잡한 내부 로직을 직접 파악하려 했던 것과 달리, PathFinder는 반복 실행에서 얻은 입출력 예시들을 바탕으로 조건을 추론한다는 점에서, 구조적으로 복잡한 프로그램에서도 의미 있는 경로 탐색이 가능하도록 한다. 이러한 방식은 테스트의 효율성과 정밀도 사이의 균형을 유지하면서도, 기존 블랙박스 테스팅 방식이 놓칠 수 있는 의미 있는 경로 탐색을 가능하게 한다.

PathFinder는 PyTorch와 TensorFlow의 다양한 API에 대해 기존 딥러닝 라이브러리 테스팅 도구들보다 더 높은 분기 커버리지와 버그 탐지 성능을 보여주었으며, 복잡한 API 구조에도 적용 가능한 실용적이고 확장 가능한 추론 기반 화이트박스 테스팅 기법의 가능성을 입증했다.

4. 연구를 돌아보며

이 연구를 정리하면서, ‘현실적인 테스팅’이 무엇인지 다시금 생각하게 되었다. PathFinder는 근본적으로 기존의 이상적인 화이트박스 분석이 적용되지 않는 상황에서 시작되었고, 그 대안으로 현실적인 조건에서 작동 가능한 경량 분석 기법을 고민한 결과물이다. 처음부터 완성된 설계가 있었던 것은 아니고, 시행착오와 실패를 반복하며 점차 방향을 잡아갔다. 특히, 분기 조건을 정확히 분석하려 하기보다는 실행을 거듭하며 조건을 추론하고 정제해 나간다는 전략은, 초기에 과연 이 방식이 잘 작동할지에 대한 확신이 부족한 상태에서 실험을 밀어붙여야 했기 때문에 쉽지 않았다. 실제로 예상치 못한 결과가 발생하기 일췌였고, 이를 보완하기 위해 알고리즘을 다듬고 실험 체계를 개선하는 과정이 반복되었다.

그럼에도 불구하고 끝까지 포기하지 않고 기술을 고도화해준 김세훈 학생의 끈기가 이 논문을 완성으로 이끌었다고 해도 과언이 아니다. 또한, 공동저자인 김용현 학생과 박다현 학생은 방대한 API 실행과 실험 결과 정리를 책임감 있게 수행하며, 수차례의 마감과 수정 과정에도 큰 힘이 되어주었다.

PathFinder의 아이디어는 단기간에 정리된 것이 아니다. 연구 초창기부터 그 당시 같은 학교 동료였던 전유석 교수님과 함께 기존의 기호실행 방식이 왜 딥러닝 라이브러리에 적용되기 어려운지, 그리고 대안으로 어떤 경량 분석이 가능할지를 계속 논의해왔다. 이후 출장 중 이주용 교수님과 기차 안에서 나누던 대화 내용이 PathFinder의 분기 조건 합성(path-condition synthesis) 개념의 실마리가 되었다. 이후 두 교수님과 함께 아이디어를 구체화하고 실험을 설계하면서 지금의 형태로 발전시킬 수 있었다. 개인적으로 이번 연구는, 협업이 단지 역할 분담을 넘어 연구의 방향성과 완성도를 함께 만들어가는 과정이 될 수 있다는 점을 다시금 실감하게 해준 소중한 경험이었다. 끝으로, 이 여정을 함께해 준 모든 동료들과 연구실 학생들께 진심으로 감사의 마음을 전한다.

1) 본 연구에서는 한양대 이우석 교수님 연구실에서 개발한 Duet이라는 귀납적 프로그램 합성기 (POPL 2021) [Lee, 2021]를 사용하였다.



소프트웨어공학과 학술대회의 미래: ICSE/FSE/ISSTA 2025를 둘러보고



• 유신 교수
한국과학기술원 전산학부

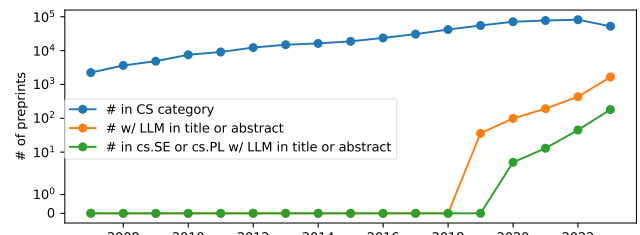
2025년 상반기에는 조금 무리인가 싶게 여러 개의 학술대회에 참여했다. IEEE International Conference on Software Testing, Verification & Validation (ICST 2025)에는 산업체 논문을 발표하러 참석했고, 소프트웨어공학 분야에서 가장 큰 학술대회인 IEEE/ACM International Conference on Software Engineering (ICSE 2025)에는 매년 그렇듯 학계 전체의 동향을 파악할 기회로 삼아 연구실 전원이 함께 다녀왔다 (아울러 연구실에 최근에 합류한 새로운 세대 구성원들은 다양한 워크샵에 논문을 발표하며 작게나마 자기 연구의 매듭을 짓는 연습을 할 수 있었다). 마지막으로 종강 후에는 한 군데서 공동개최한 ACM Foundations on Software Engineering (FSE 2025), 그리고 ACM International Symposium on Software Testing and Analysis (ISSTA 2025)에 논문을 발표하고 스웨덴 동료들과 LLM 검증에 대한 다음 연구를 논의하러 다녀왔다. 이 중 ICSE와 FSE는 IEEE/ACM International Conference on Automated Software Engineering (ASE)과 함께 소프트웨어공학 분야에서 가장 중요한 세 개의 학술대회로 손꼽히는데, 11월에 ASE 2025를 한국에서 개최하는 General Chair 역할을 맡은 만큼 행사를 잘 준비하려면 다른 학술대회 운영을 벤치마킹해야겠다는 욕심도 한 몫 했다. 소식지 지면을 빌어 최근 소프트웨어공학 연구 동향을 요약해보고, 학술대회라는 행사의 미래에 대한 고민도 함께 나눠보고자 한다.

생성형 모델의 충격

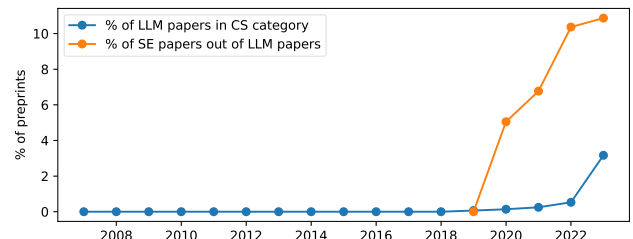
OpenAI가 2022년 공개한 대규모 언어 모델(Large Language Model)인 GPT-3.5는 자연어 프롬프트를 바탕으로 프로그램 소스코드를 생성하는 모습으로 많은 사람들에게 충격을 주었는데, 소프트웨어공학 연구자들도 예외는 아니었다. 전통적으로 소프트웨어 개발 생애주기를 요구사항 분석 → 설계 → 구현 → 검증 → 유지보수의 다섯 단계로 구분한다. 소프트웨어공학 연구는 나머지 네 가지 단계에서 다양한 기술을 이용해 더 높은 수준의 정확성(correctness) 및 자동화 수준을 추구했지만, 정작 구현 단계에서만은 코드는 사람이 작성하는 것이라는 근본적인 개념에 이의가 없었다고 생각한다. 새로운 프로그래밍 언어의 개발이나, 통합개발환경(Integrated Development Environment)와 같은 생산성을 높이기 위한 노력들은 구현 단계의 핵심이라기보다는 환경적인 요소라고 여겨졌고, 유전 프로그래밍(Genetic Programming)이나 프로그램 합성(Program Synthesis)과 같은 기법은 작은 함수 단위를 넘어가는 소프트웨어 시스템 전체를 생성하기에는 부족한 기술로 여겨졌다[21]. 자연어로 주어진 지시에 따라 상당한 분량의 코드를 다양한 언어로 만들어내는 생성형 모델은 설계사항을 코드로 옮기는 고도의 지능적인 행위는 사람만 할 수 있다는 암묵적인 가정을 깨뜨리는 것이었다.

우리가 아는 형태의 프로그래밍은 이제 끝났다는 감상마저 등장한 것을 보면[17] 이런 변화가 얼마나 큰 충격이었을지 짐작해볼만 하다.

최초의 충격이 지나간 다음 소프트웨어공학 연구자들은 코드건 문서건 자동으로 생성할 수 있는 이 새로운 기술에 열광했다. 그림 1은 Fan et al.[3]의 서베이 논문에서 가져왔는데, arXiv에 공개된 프리프린트를 대상으로 논문의 제목과 초록에 등장하는 단어를 분석해 LLM이 연구동향에 미친 영향을 분석한 결과이다. 그림 1a는 arXiv에 등록된 전산학 분야 논문 전체 숫자 대비 제목이나 초록에 대규모 언어 모델이 언급된 논문의 숫자, 그리고 그 중 세부 분야가 소프트웨어공학이나 프로그래밍언어로 지정된 논문의 수를 비교한 것이다.



(a) SE와 PL 분야에서 제목과 초록에 “LLM”이 등장하는 논문 수



(b) 제목과 초록에 “LLM”을 포함하는 논문 중 SE논문의 비율

그림 1. arXiv 논문 분석 [3]

2019년 무렵부터 LLM과 관련된 소프트웨어공학과 프로그래밍언어 연구가 폭발적으로 증가했음을 쉽게 알 수 있다. 그림 1b는 전산학 논문 전체 대비 대규모 언어 모델을 언급한 논문의 비율(%), 그리고 대규모 언어 모델을 언급한 논문 중 세부 분야가 소프트웨어공학으로 지정된 논문의 비율(%)을 비교한 것으로, 역시 소프트웨어공학 연구가 대규모 언어 모델을 빠른 속도로 받아들였음을 알 수 있다.

최근 연구 동향

LLM을 활용한 기술들은 지난 몇 년간 주요 소프트웨어공학 학술대회 안에서 큰 비중을 차지했다. 기존에 어떤 형태로든 코드를 자동생성하는 것을

목표로 했던 기술은 LLM을 이용해 다시 시도되었다고 해도 과언이 아닐 것 같다. 클래스 단위의 코드를 바로 합성하는 능력을 평가하는 것에서부터[2], 대상이 되는 코드를 프롬프트에 포함한 다음 단위 테스트를 생성하는 것[9], 딥러닝 라이브러리를 검증하기 위해 해당 라이브러리를 사용하는 임의의 코드를 생성하는 퍼징기술[1] 등 다양한 문제가 코드 생성이라는 관점에서 연구되었다. LLM이 아무런 제약 없는 자연어로 입출력을 허용하기 때문에, 기존에 자연어 처리 능력의 한계로 풀기 어려웠던 문제도 자동으로 해결할 수 있게 되었다. 자연어로 작성한 버그 리포트를 바탕으로 이를 재현하는 테스트를 합성한다던지[4,8], 코드와 주석이 의미적으로 일치하는지 확인하는 연구[13] 등이 좋은 예라고 할 수 있다.

한편으로는 최초의 충격에서 벗어나서 LLM의 코드 생성 능력, 그리고 LLM을 사용하는 소프트웨어 개발자들의 경험 자체를 차분히 들여다보는 연구도 눈에 띄었다. ICSE 2024에는 실제 개발자로부터 수집한 코드 자동완성 요청에 대해 LLM이 얼마나 정확한 성능을 보였는지 측정한 연구가 발표되었는가 하면[6], 올해 FSE에서는 학생들이 생성형AI를 프로그래밍에 어떻게 활용하는지 자세히 관찰한 연구가 흥미로웠다[11]. 또 LLM이 생성한 코드와 사람이 작성한 코드 사이에 구분 가능한 패턴이 있는지를 살펴본 연구[14], LLM이 생성한 코드 중 얼마만큼이 학습 데이터를 그대로 암기(memorisation)한 것인지를 평가하는 연구[19] 등도 LLM의 활용 분야가 넓어지면서 자연스럽게 등장한 연구 주제라고 생각된다.

추론과 에이전트

소프트웨어공학 연구자들이 생성형모델의 코드 생성 능력을 평가하고 활용하는 동안, 인공지능 연구는 생성형모델의 성능을 다음 단계로 끌어올리는 데 힘을 기울였고, 현재 그 결과는 추론과 에이전트라는 두 가지 개념으로 요약할 수 있을 것 같다. 연초 모두를 놀라킨 DeepSeek 모델은 흔히 추론 능력을 내재하고 있다고 일컬어지는데, 크게 보았을 때 기존에 알려진 프롬프팅 방법인 Chain-of-Thoughts[16]를 모델의 훈련 자체에 포함해 내재화한 것으로 이해할 수 있다. 에이전트 시스템은 주어진 문제를 해결하기 위해 자율적인 결정을 내리고, 외부에 있는 도구를 사용할 수 있는 인공지능시스템이라고 이해할 수 있는데, LangChain¹⁾과 같은 오픈소스 프로젝트, 그리고 OpenAI가 제공하는 API에 의해 ReAct 프롬프팅[20]에 기반한 에이전트 구현의 윤곽이 드러난 이후 최근에는 Anthropic이 공개한 Model Context Protocol²⁾의 인기에 힘입어 급격히 발전하고 있는 소프트웨어 형태이다.

소프트웨어공학 분야에 있어서도 주어진 문제를 해결하기 위해 추론을 수행하고 도구를 사용하는 에이전트는 이미 새로운 개념이 아니다. 이미 FSE 2024에 결함 위치 식별을 수행하는 에이전트가 발표되었으며[7], Google은 ICSE 2025의 Software Engineering in Practice 트랙에서 프로그램 자동 수정을 위한 에이전트인 Passerine을 공개했다[12]. 한편으로는 에이전트 아키텍처가

불필요하게 복잡하며, 현재 널리 사용되는 벤치마크 문제를 해결하는 데에는 에이전트가 필요 없다는 결과도 보고된 바 있다[18]. 하지만 소프트웨어 개발 과정에서 마주치는 많은 문제들이, 질문에 대한 단일한 답으로 해결되는 종류이기 보다는 단계적인 접근과 복잡한 분석을 통해서만 풀 수 있다는 점, 그리고 문제를 더 작은 단계로 쪼개서 풀 경우 기존에 소프트웨어공학 연구를 통해 개발한 다양한 자동화 도구를 적재적소에 사용하고 추론의 결과를 검증하기가 더 용이할 것이라는 점에서 필자는 에이전트 기반의 접근 방법이 앞으로 더 주목받지 않을까 예상하는 편이다.

새로운 형태의 소프트웨어

그런데 LLM을 사용하는 에이전트 기반의 소프트웨어공학 도구를 생각하다보면, 생각이 자연스럽게 확장된다: “굳이 소프트웨어공학 도구만 이렇게 만들라는 법은 없지 않을까?” 당연한 이야기지만 뛰어난 자연어 처리 능력을 가진 기술로 인해 기존에 불가능했던 기능을 소프트웨어로 구현할 수 있게 되는 현상은 소프트웨어공학에만 국한된 이야기가 아니며, 이미 정신건강 돌봄을 위한 에이전트 소프트웨어[5], NPC(Non-Player Character)의 대사를 LLM을 위해서 생성하는 게임[15] 등 기존에는 생각하기 어려웠던 응용 사례들이 등장하고 있다. 추론 능력을 지닌 에이전트라는 관점에서 LLM이 가진 가장 놀라운 능력은 프로그래밍 언어의 논리식과 분기로 표현하기 어려웠던 의사결정을 상식에 기반해 내릴 수 있다는 점이라고 생각한다. SMT Solver가 논리식으로 주어진 명제들에 대한 계산을 수행하는 엔진인 것 처럼, LLM은 자연어 문장으로 주어진 명제들에 대한 계산을 수행하는 또 다른 연산 엔진인 것이다. 이 새로운 연산 엔진은 매우 강력하고 사용이 편리하지만, 전통적인 소프트웨어와 달리 쉽게 그 실행을 분석할 수 없으며 환각 현상과 같은 근본적인 취약점을 가지고 있다. LLM과 관련한 초기의 소프트웨어공학 연구가 어떻게 소프트웨어공학 문제를 더 잘 해결하기 위해 LLM을 사용할 것인가, 즉 AI4SE의 형태였다면 앞으로는 LLM을 이용해 만든 응용 소프트웨어를 어떻게 잘 설계/구현/검증할 것인가, 다시말해 이른바 SE4AI 방향의 문제가 점차 대두할 것이다. LLM을 기반으로 한 새로운 형태의 소프트웨어를 위한 SE4AI 연구에는 LLM의 해석가능성(Interpretability) 문제 등 인공지능 연구가 풀어야 할 문제도 존재하지만, 비결정적(Non-deterministic)인 행동을 하는 구성요소를 어떻게 대규모 소프트웨어 시스템의 일부로 안전하게 사용할 것인가하는 고전적인 소프트웨어공학 문제도 존재한다. 단순히 LLM을 이용해 코드나 기타 생성물을 만드는 것을 넘어서서, 새로운 형태의 LLM 기반 소프트웨어를 어떻게 표현하고, 설계하고, 분석 및 검증할 것인지 미래를 내다보는 연구가 필요하다.

좋은 문제란 무엇인가

인공지능이 소프트웨어공학 연구에 미친 영향에 대해 길게 설명했지만, 학술대회에는 수백편의 논문이 발표되고, 당연히 인공지능과 별 관계 없는 전통적인 연구, 기계학습을 사용하지만 LLM과는 관계가 없는 연구, 오직 인간

1) <https://www.langchain.com>

2) <https://modelcontextprotocol.io>

개발자의 행동만을 관찰하는 연구 등 여러가지 이야기가 담긴 논문들이 존재한다. 다양한 논문 발표를 지켜보면서 새삼스레 확인한 것은 최신 기술을 따라가는 시의성만큼이나 바로 남들이 미처 발견하지 못한 좋은 문제를 알아보는 눈이 중요하다는 사실이다. ICSE 2025에서 지난 몇 년간 가장 재미있게 발표를 들은 연구가 있었는데, 바로 Git 코드 저장소가 내부적으로 쓸모없어진 데이터를 모아 청소하는 Garbage Collection을 저장소의 입출력 속도 저하 없이 수행하기 위해 일정을 어떻게 학습할 것인가라는 문제에 대한 연구였다[10]. 연구자들은 강화학습을 이용해 다양한 Garbage Collection을 수행하는 적절한 타이밍을 학습하는 기술로 이 문제를 해결했는데, 이 발표를 들으며 무릎을 친 이유는 강화학습을 사용했기 때문이 아니라 거대한 Git 저장소의 입출력 속도 저하라는 문제 자체가 너무나 새로우면서 동시에 실용적인 문제였기 때문이다. 비슷한 맥락에서 Amazon Web Service에 존재하는 자원을 관리하는 규칙에 적용하기 위해 도메인 특화 언어를 개발하고 이를 위한 정형 검증 기법을 개발한 이야기를 전해준 AWS Director of Science인 Neha Rungta 박사님의 ICSE 2025 키노트³⁾도 기억에 남는다. “A는 원래 느리니까 어쩔 수 없는 것”이라거나, “X라는 기술은 의례 Y에만 적용하는 것”과 같은 고정관념에 얽매이지 않는 사람이 좋은 문제를 알아보고, 거기에 적절한 기술과 도구를 유형에 관계없이 잘 고를 수 있다.

학술대회 형식에 대한 고민

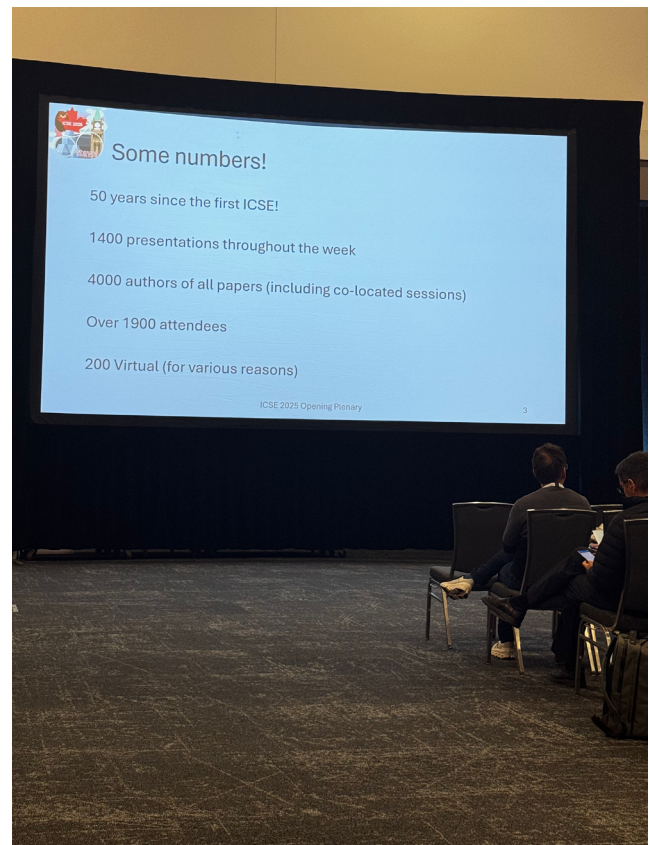
조만간 커다란 학회를 운영해야 하는 입장에서는 미래의 소프트웨어 형태만큼이나 미래의 학술대회 형태에 대한 고민이 크게 다가왔다. 지난 10여년 동안 소프트웨어공학 연구 공동체는 여러가지 측면에서 폭발적인 성장을 경험했다. 기술적으로는 딥러닝과 생성형 인공지능이라는 혁신을 통해 새로운 연구 기회가 유입되었고, 지정학적으로는 전통적으로 연구를 이끌던 북미와 유럽 이외의 지역에서도 수준 높은 연구가 진행되면서 새로운 인적 자원이 유입되었으며, 산업적으로는 빅테크 기업들의 폭발적인 성장으로 말미암아 소프트웨어공학 인력에 대한 수요가 발생했다. 지난 몇 년간 대부분의 주요 학술대회는 매 년 접수된 논문 수가 전년의 기록을 갱신하는 성장을 경험했는데, 이 폭발적인 성장은 판데믹 기간과 겹치면서 학술대회라는 행사의 형식이 가지는 한계 또한 여럿 드러냈다. 이 자리를 빌어 관련된 모든 문제점을 나열하고 분석하는 것은 무리이겠지만, 당장 쉽게 떠오르는 문제만 간추려도 다음과 같다.

- 투고된 논문 수의 폭발적인 증가로 인한 논문심사 업무량의 증가
- 발표되는 논문 수의 증가로 인한 평행 세션 증가 및 논문별 발표 시간 감소
- 지정학적 불안정성으로 인해 비자 발급이 용이하지 않은 문제
- 점차 증가하는 학술대회 참가의 경제적 비용
- 가상화된 학술대회에 대한 수요와 이를 구현하는 기술 수준간의 간극

어떤 문제는 학계 전체가 함께 고민해야 할 것이고, 어떤 문제는 학계의 능력 밖에 있는 것이다. 개별 학술대회를 진행하는 입장에서 해결을 도모해볼만한

문제는 평행 세션이 너무 많아져 원하는 발표를 듣기 어려운 문제, 논문별 발표 시간이 너무 짧아져서 깊이있는 기술적인 내용을 전달하기 어려워지는 문제 정도이다. 올해 ICSE는 논문당 10분 정도의 발표 시간을 준 데 비해, FSE와 ISSTA는 ACM이 학회 프로시딩을 Proceedings of ACM이라는 저널 형태로 출판하는 전략을 취함에 따라 학회에 논문이 받아들여져도 저자가 학회에 참석해 발표할 의무가 없어지는 매우 새로운 실험을 했다. 특히 ISSTA는 전통과 달리 평행 세션을 대거 받아들여 논문당 25분의 넉넉한 발표시간을 배정했는데, 처음에는 반가운 소식이라고 생각했지만 학회 기간 동안 25분 발표를 집중해서 듣는 것도 쉬운 일이 아니라는 사실을 깨달아야만 했다. 지난 10여년 동안 현대인이 집중할 수 있는 시간이 그만큼 짧아진 것이 아닐까. 아예 학회장에 랩탑을 가져가지 않는 노력을 해가면서 몇몇 세션의 좌장 역할을 성공적으로 마칠 수 있었다.

한편에서는 매 년 논문 숫자가 늘어나는 양적인 성장이 과연 진정으로 유익한 것인가 하는 성찰의 목소리도 존재했다. 우리 모두 논문을 \emph{덜} 쓰도록 노력해야 한다는 익살스런 우스개를 여러 번 들을 수 있었다. 하지만 연구를 양보다 질로 평가하는 것과 별개로, 지금의 학술대회가 가장 효율적인 의사소통을 보장하는 형식인지는 고민해볼 필요가 있지 않을까. 학술대회의 규모 그리고 전달해야 하는 내용의 복잡도를 모두 고려할 때, 많은 사람들이 소프트웨어공학 학술대회도 일부 거대규모의 인공지능/기계학습 학술대회처럼 포스터 위주의 행사로 진화할 수 밖에 없다고 예상하고 있다. 여건이 허락하는 한 ASE 2025에서 포스터 기반의 발표를 최대한 활용해보겠다고 마음먹었다.



3) <https://conf.researchr.org/details/icse-2025/icse-2025-main-icse-plenaries/3/Neha-Rungta-Keynote-Engineering-correctness-for-a-domain>

마치는 글

나는 학술대회 참석을 좋아한다. 시차와 긴 여행때문에 지치고 피곤하지만, 가는 동안 그리고 다녀온 뒤에 느끼는 일종의 흥분감이 있다. 가보지 못한 새로운 장소에 가는 설렘도 있기는 하지만, 단순히 장거리 여행이어서만은 아니다. 학회에는 정신적인 여행이 포함되어 있다. 참가자는 학회 기간 동안 반복되는 일상에서 스스로의 두뇌를 잠깐 분리해서 오직 학회의 주제만 생각할 수 있기 때문이다. 대부분 호텔에 묵으며 때가 되면 제공되는 커피와 식사로 연명(?)하니 의식주에 신경을 쓸 필요가 없고, 출장 중이라는 핑계로 당장 급하지 않은 다른 이메일들은 잠깐 미뤄 둘 수 있다 (물론 나중에 혹독한 댓가를 치른다). 대신에 1년에 한 번 정도 만나는 반가운 친구들과 하루 온종일 방금 들은 새로운 기술에 대한 이야기, 우리 분야는 대체 어디로 가는가 하는 이야기, 내년 학회는 어디에서 하는 게 좋은가 하는 이야기, 인공지능은 정말 대체 무엇인가 하는 이야기를 할 수 있으니, 일상에서 누리기 힘든 호사가 아닐 수 없다. 학회장 바깥 사람들에게는 유치해보이거나 도무지 이해할 수 없는 이야기겠지만, 고개만 돌리면 이런 이야기들을 하고 싶은 사람들로 주변이 가득하다!

최근 어느 학술대회에선가 해외의 동료와 대화를 나누던 중, 팬데믹 기간 동안 모든 업무를 가상/원격 환경으로 수행하는 데 익숙해진 새로운 세대의 연구자들 중 일부는 학술대회 참석을 위해 장시간 비행기를 타고 여행을 떠나기를 꺼려한다는 이야기를 듣고 낯선 기분이었다. 학계에 연구자로 남는 것의 장점 중 하나가 연구 성과만 좋으면 학술대회를 통해 세계 여러 곳을 여행할 수 있는 거라고 이야기하곤 했는데, 격세지감이기도 하고 그만큼 팬데믹이 많은 것을 바꿔놓았다는 생각도 든다. 종잡을 수 없는 기후 변화를 경험하면서 이만큼의 탄소 발자국을 남기며 학회에 참석하는 게 옳은가 하는 고민도 멈추지 않는다. 하지만 학술대회가 온라인이나 오프라인이냐에 관계없이 부인할 수 없는 사실이 있다: 현재의 첨단 과학 기술 연구는 결코 혼자서 할 수 있는 것이 아니라 사회적/집단적인 활동이고, 따라서 같은 관심사를 공유하는 연구자들이 서로 의견을 교환하고 호흡을 함께하는 것이 꼭 필요하다는 점이다. AI와 같이 급속도로 발전하는 기술이 외부로부터 충격을 가해왔을 때는 더욱더 그러하다. 많은 국제학술대회가 개최 지역을 북미와 유럽 이외로 다변화하려는 노력을 하고 있기 때문에, 앞으로 학생 연구원들에게는 학술대회에 참여할 기회가 이전보다 조금 더 늘어나지 않을까 한다. 아무쪼록 본인의 연구를 마음껏 뽐내고, 뜻깊은 인연과 마음 맞는 친구를 만들 기회를 많이 갖기를 바란다.

References

- [1] Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, and Lingming Zhang. Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), pages 841–853, 2024.
- [2] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu,

- Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Evaluating large language models in class-level code generation. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), pages 982–994, 2024.
- [3] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang. Large language models for software engineering: Survey and open problems. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSEFoSE, pages 31–53. IEEE Computer Society, May 2023.
- [4] Sidong Feng and Chunyang Chen. Prompting is all you need: Automated android bug replay with large language models. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), pages 803–815, 2024.
- [5] Yining Hua, Hongbin Na, Zehan Li, Fenglin Liu, Xiao Fang, David Clifton, and John Torous. A scoping review of large language models for generative tasks in mental health care. npj Digital Medicine, 8(1):230, 2025.
- [6] Maliheh Izadi, Jonathan Katzy, Tim van Dam, Marc Otten, Razvan Mihai Popescu, and Arie van Deursen. Language models for code completion: A practical evaluation. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), pages 956–968, 2024.
- [7] Sungmin Kang, Gabin An, and Shin Yoo. A Quantitative and Qualitative Evaluation of LLM-based Explainable Fault Localization. Proceedings of the ACM on Software Engineering, 1(FSE):64:1424–1446, July 2024.
- [8] Sungmin Kang, Juyeon Yoon, and Shin Yoo. Large language models are few-shot testers: Exploring llm-based general bug reproduction. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, 2023.
- [9] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K. Lahiri, and Siddhartha Sen. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pages 919–931, 2023.
- [10] Shane McIntosh, Luca Milanese, Antonio Barone, Jacek Centkowski, Marcin Czech, and Fabio Poncioli. Using reinforcement learning to sustain the performance of version control repositories. In 2025 IEEE/ACM 47th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE–NIER), pages 116–120, 2025.
- [11] Christian Rahe and Walid Maalej. How do programming students use generative ai? Proc. ACM Softw. Eng., 2(FSE), June 2025.

- [12] Pat Rondon, Renyao Wei, José Cambronero, Jürgen Cito, Aaron Sun, Siddhant Sanyam, Michele Tufano, and Satish Chandra. Evaluating Agentbased Program Repair at Google, 2025.
- [13] Guoping Rong, Yongda Yu, Song Liu, Xin Tan, Tianyi Zhang, Haifeng Shen, and Jidong Hu. Code Comment Inconsistency Detection and Rectification Using a Large Language Model. In 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), pages 1832–1843, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.
- [14] Yuling Shi, Hongyu Zhang, Chengcheng Wan, and Xiaodong Gu. Between Lines of Code: Unraveling the Distinct Patterns of Machine and Human Programmers. In 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), pages 1628–1639, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.
- [15] Li Song. Llm-driven npcs: Cross-platform dialogue system for games and social platforms, 2025.
- [16] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.
- [17] Matt Welsh. The end of programming. Commun. ACM, 66(1):34–35, December 2022.
- [18] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying Llm-based software engineering agents. Proc. ACM Softw. Eng., 2(FSE), June 2025.
- [19] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. Unveiling memorization in code models. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), pages 867–879, 2024.
- [20] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In Proceedings of the International Conference on Learning Representation, ICLR 2023, 2023.
- [21] Kwaku Yeboah-Antwi. Evolving software applications using genetic programming – PushCalc: the evolved calculator. In Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '12, pages 569–572, New York, NY, USA, 2012. Association for Computing Machinery.





소프트웨어공학 단기 강좌 후기: 단기전문가강좌를 마치고



• 류덕산 교수
전북대학교/단기강좌 프로그램위원장

한국정보과학회 소프트웨어공학 소사이어티는 매년 여름마다 단기 전문가 강좌를 개최하고 있습니다. 2025년 8월 21일~22일 양일간 KAIST 도곡캠퍼스에서 열린 제13회 소프트웨어공학 단기 전문가 강좌에서는 최근 가장 주목받는 주제 중 하나인 '소프트웨어공학을 위한 LLM 기반 에이전트 개발'이라는 주제로 한국과학기술원 유신 교수님과 한국과학기술원 윤주연 연구원님의 강의가 진행되었습니다.

유신 교수님의 연구 분야는 메타휴리스틱 최적화, 기계학습 및 인공지능 등 다양한 계산지능을 이용해 소프트웨어공학 문제를 해결하는 AI4SE, 그리고 소프트웨어공학 기술을 이용해 인공지능 기반 소프트웨어의 품질을 향상시키는 SE4AI 두 분야에 걸쳐 있으며 소프트웨어 테스팅, 디버깅, 프로그램 자동 수정 등의 분야에서 다양한 연구 결과를 발표하였습니다.

윤주연 연구원님은 대규모 언어 모델(LLM)을 활용한 소프트웨어 품질 향상을 주요 연구 주제로 삼고 있으며, 최근에는 LLM 기반 에이전트를 통해 GUI(Graphical User Interface) 테스트를 자율적으로 수행하는 방법과 LLM 기반 애플리케이션을 위한 테스트 기법에 대한 연구를 진행하며 관련 논문을 발표하였습니다.

1일차 강의에서는, 대규모 언어 모델의 기본 개념을 확인한 후, Chroma를 이용한 Vector DB + RAG 및 MCP 기반 Python 테스트 생성 자동화 구현 기법을 다루었습니다.

2일차 강의에서는, 에이전트 기술 및 LLM 기반 최신 연구 현황 및 PlayWright를 이용한 End-to-End 웹 테스팅 에이전트의 구현 실습을 진행했습니다.

이를 동안 총 14시간에 걸쳐 진행된 이번 단기강좌는 LLM에 기반한 소프트웨어공학 연구의 기본 개념과 그 한계를 이해하고, MCP 프로토콜의 원리와 디자인 목표를 이해하고 에이전트 구성에 사용하는 방법을 다루었습니다. 이론과 코딩 실습을 연계한 이 강좌는 LLM 및 에이전트와 소프트웨어 테스팅, 그리고 융합분야를 연구하는 연구자들에게 유익한 기회를 제공했습니다.

이번 단기강좌에는 총 37명이 등록하였으며, 그중 24명이 학생이고, 13명은 교수 및 산업체 관계자들이었습니다. 강의 평가가 4.5 / 5.0을 기록할 정도로 수강생들로부터 높은 평가를 받았습니다. 훌륭한 강의 장소를 마련해주신 고인영 교수님과 백준모 교수님께 감사드리며, 식사장소 섭외와 운영에 힘써준 전북대 인공지능&소프트웨어공학 연구실 학생들에게 깊은 감사를 표합니다.





기업탐방 슈어소프트테크 : Software for Safe World

슈어소프트테크(주) 최경화 연구소장/상무

편집자: 2002년 창립 이후, 국내에서는 드물게 상용 SW 검증 자동화 도구 전반을 자체 개발해 오셨습니다. 슈어소프트테크의 핵심 솔루션(정적 분석, 코드·모델·시스템 검증, 테스트 자동화 등)의 차별점과 주요 산업 적용 사례를 소개 부탁드립니다.

최경화 상무: 슈어소프트테크는 '소프트웨어로 만드는 안전한 세상'이라는 비전을 가지고, 독자적 테스트 자동화 원천 기술을 바탕으로 2002년 소스코드 단위 통합 시험 자동화 도구에서부터 소프트웨어 개발 lifecycle 전 단계를 아우르는 상용 SW 검증 자동화 도구를 개발해 왔습니다. 이 솔루션들은 모델 검증, 코드 검증, 시스템 검증 등 검증 전체를 아우르는 End to end V&V(Verification & Validation)를 구축한 것이 특징입니다. 또한 Black-Box(기능) 테스트 뿐 아니라, 코드 실행 경로까지 심층 분석하는 White-Box(구조) 테스트 자동화 역량을 갖추고 있어 소프트웨어 결함을 더 효과적으로 찾을 수 있습니다.

저희 솔루션은 특히 자동차, 국방, 항공, 철도, 원자력, 의료 등 소프트웨어의 오류가 치명적인 영향을 초래하는 Mission critical 산업 분야에서 많이 활용되고 있습니다. 그 중에서도 가장 많이 활용되고 있는 분야가 자동차 및 국방 분야입니다.

우선 자동차 도메인에서는 제어기 소프트웨어 검증시 국제 표준에 부합하는 단계별 검증 활동에 저희 도구들이 활용되고 있습니다. MISRA 등 국제 표준 코딩 가이드라인 검증이나 실시간 에러를 발생시킬 수 있는 메모리 문제, 산술 오류 등을 소프트웨어 실행 없이 분석하는 정적검증에 STATIC이, 실제 소프트웨어를 실행시키면서 검증하는 단위 통합 검증, 요구사항 기반 검증을 수행하는 동적 검증에 CT가 활발히 사용되고 있습니다.

제어기 SW 검증

- 국제 표준 부합 단계별 검증 수행 (ISO 26262, MISRA-C, ISO 29119 등)

정적 검증

- 코딩 룰 검증
- MISRA-C, ISO 26262 등
- 코드 메트릭 검증
- 함수 복잡도, 호출관계, 재귀호출 등
- Run-time Error 검증
- Buffer overflow, Memory leak 등

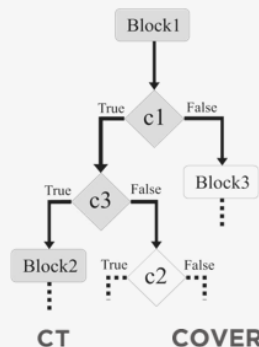


코드 실행 없이 구문 분석을 통해 검증 수행

STATIC

동적 검증

- 단위 및 통합 시험 수행
- 요구사항 및 설계 분석을 통한 테스트 케이스 개발 및 수행
- 구조적 커버리지 확인
- 구문, 분기, MC/DC, 함수호출 등의 커버리지 측정

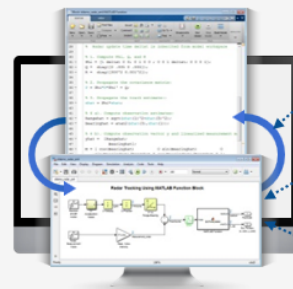


CT

COVER

모델 검증

- 정적 검증 (모델 룰 검증)
- MISRA-AC, MAAB, Secure Coding 등
- 동적 검증
- 모델상에서 기능 검증 수행
- 일치성 검증
- 모델-코드 간 정합성 검증



일치성
검증

모델 정적
검증

모델 동적
검증

그림 1. 자동차 제어기 소프트웨어 검증

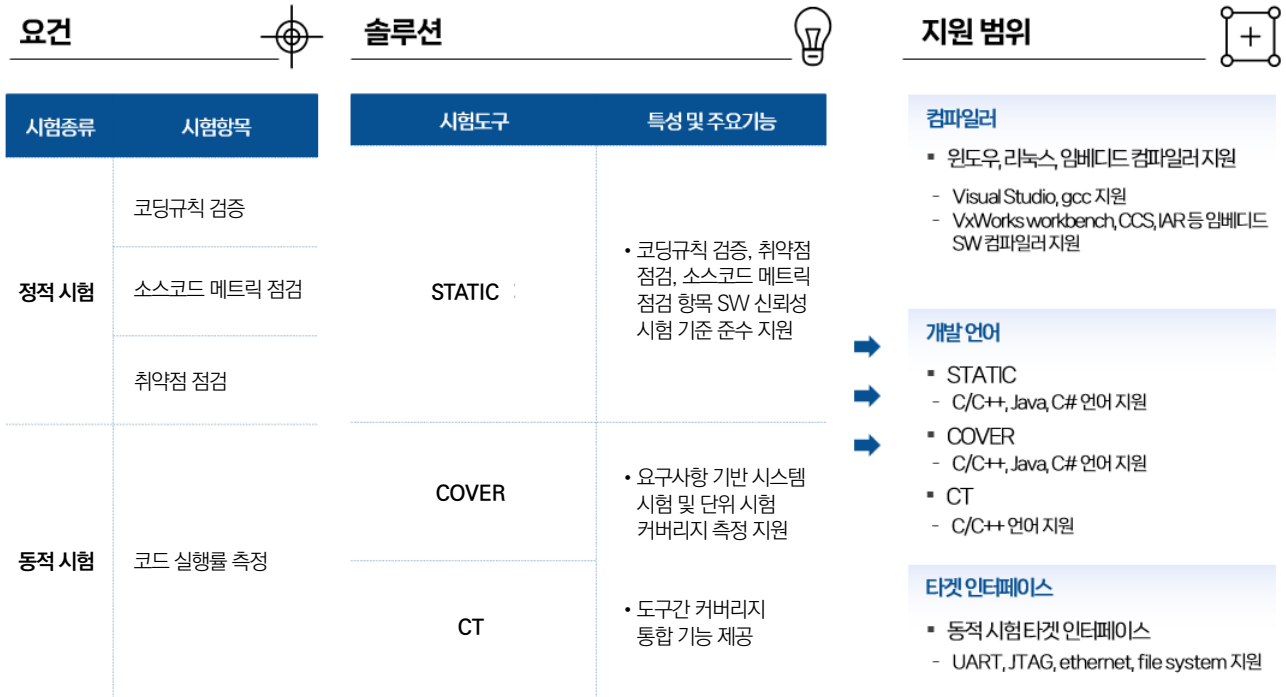


그림 2. 국방 무기체계 소프트웨어 신뢰성 시험 요건 및 솔루션

국방 무기체계 소프트웨어의 경우 방위사업청 지침에 따라 신뢰성 시험을 수행하고 있는데, 저희 정적 시험 도구인 STATIC과 동적 시험 도구인 COVER, CT를 활용하고 있습니다.

편집자: 국방, 자동차, 항공·우주, 철도, 의료, 원자력 등 다양한 고신뢰 산업 분야에서 SW 검증 자동화 도구의 활용 가치와 기대 효과는 어떻게 체감하고 계신지요? 산업별 주요 요구사항과 도입 초기의 반응은 어땠는지도 궁금합니다.

최경화 상무: 소프트웨어의 오류가 치명적인 영향을 초래하는 Mission critical 산업 분야에서는 특히나 검증이 중요합니다. 저희 솔루션들이 주로 사용되는 곳도 이러한 고신뢰 산업 분야입니다. 이러한 산업분야에서 가장 큰 요구사항은 국제 표준 기반의 검증 활동을 지원해 주는 것입니다. 슈어소프트테크는 자동차(ISO26262), 철도(EN50128), 원자력(IEC61508), 의료(IEC62304), 항공·우주(DO-178C) 등 소프트웨어 안전성(Safety)과 관련된 국제 표준을 기반으로 검증에 필요한 솔루션들을 공급해 주고 있습니다. 초기에는 국내 기술로 만든 솔루션에 대한 의심의 눈초리도 있었지만 도입 경험에 누적되고, 성공사례가 축적될수록 도입 기업들의 기대치도 높아졌고, 산업 요구에 맞춰 기능 개선과 현장 밀착 지원이 강화되었습니다. 이제는 화이트박스 테스팅 기술력, 신속 대응력, 국제 인증 테스트 등에서 점차 차별성과 실효성을 인정받고 있습니다. 저희 소프트웨어 검증 솔루션을 통해 치명적 결함을 예방함으로써 소프트웨어의 안전성과 신뢰성을 확보할 수 있고, 검증 프로세스를 자동화함으로써 개발 생산성 및 효율성을 높일 수 있습니다. 또한, 국제 사회에서 요구하는 안전성 인증을 확보하여 산업 활성화에 기여하고 있다고 말씀드리고 싶습니다.

편집자: SW 산업 전반에서 정형성, 신뢰성, 표준 준수 등 다양한 검증 요소가 모두 중요하게 다뤄지고 있습니다. 그럼에도, 실제 현장에서 자주 간과되는 요소가 있는지, 그리고 있다면 어떤 요소이며 그 이유가 무엇이라 생각하시는지요?

최경화 상무: 개발자들은 항상 빠른 개발 일정에 쫓기고 있습니다. 품질 활동의 중요성에 대한 인식만큼 실제 검증 자동화 인프라를 잘 갖추지 못하고, 품질 확보를 위한 검증 활동이 개발 일정에 밀리는 경우가 발생하고 있습니다. 효율적으로 빠른 일정에 대응할 수 있는 자동화 인프라의 확보가 필요합니다. 또한 실제 현장에서 자주 간과하는 부분이 변경에 대한 품질 관리 부분이라고 생각합니다. 소프트웨어는 지속적으로 기능 개선 및 오류 수정이 이루어지기 때문에 변경시마다 잘 정의된 품질 활동이 필요합니다. 급하게 소스코드만 수정하고 관련된 요소들을 잘 추적하고 반영하지 않을 경우 그로 인해 사이드 이펙트가 발생하게 됩니다. 따라서 설계, 코드, 테스트케이스와의 추적성이 잘 관리되고, 리그레션 테스트를 확보하여 CI/CD 등을 통해 자동화함으로써 빠른 패치 및 릴리즈에 대응할 수 있어야 합니다.

편집자: 최근 슈어소프트테크에서는 AI 모델 검증 자동화 도구인 VERIFAI-M을 출시하셨습니다. 신뢰할 수 있는 AI를 위한 검증 과정에서 어떤 점을 가장 중요하게 보고 계시며, 이 분야에서 현재 가장 큰 과제는 무엇이라고 생각하시는지요?

최경화 상무: 생성형 AI, Agentic AI 시대를 누리면서 다양한 AI 모델들, AI 어플리케이션들이 우리의 삶을 많이 바꾸고 있습니다. 반면에 알고리즘 편향과 차별 문제, 가짜 정보 및 딥페이크, 윤리성 등 다양한 사회적인 이슈도 많아지고

항목	기존 AI 검증	VERIFAI-M
입력 구성	모델 + 정답	모델 + 정답 + 자동 생성된 테스트케이스
시험 데이터 생성	없음 또는 수동 생성	적대적 / 커버리지 / 행동 테스트 기반 자동 생성
평가 방식	정확도 등 정량 지표 위주	정량 + 설명 가능한 평가(XAI)
검증 범위	추론 결과 비교	취약 구간 탐색, 시험 설계 포함
이력 관리	없음 또는 단순 추론 결과 기록	프로젝트 / 테스트 / 테스트케이스 별 자동 기록 및 리포트화

**모델이 정답을 맞춰는지만 보는 건 평가가 아닙니다.
VERIFAI-M은 "왜 맞았는지, 왜 틀렸는지"까지 확인합니다.**

그림 3. AI 검증 도구 VERIFAI-M의 강점

있습니다. 이에 따라 Trustworthy AI, 신뢰할 수 있는 AI에 대한 국내외 요구가 커지고 있고, 안전하고 신뢰할 수 있는 인공지능을 위한 글로벌 규제 논의도 증가하고 있습니다. 저희는 AI 모델 및 AI 시스템의 검증을 위한 솔루션으로 VERIFAI-M을 출시하고 Trustworthy AI에 기여하고자 합니다. Trustworthy AI를 구현하기 위한 요구사항이 여러가지로 제시되고 있는데, VERIFAI-M에서는 “기술적 강건성과 안전성”, “투명성”에 더 초점을 맞춰서 인공지능을 검증하고자 합니다. 적대적 시험/커버리지 시험/행동 시험 세가지 방법론에 근거에 시험 데이터를 자동 증강하고 시험을 수행합니다. 또한 국제 표준 기반 정량적/정성적 분석을 통해 모델의 신뢰도를 다각도로 평가하고 취약점을 분석합니다. 여기에 더해 블랙박스처럼 작동하는 AI 모델의 추론 결과에 대해 영향을 주는 입력 인자를 식별함으로써 설명 가능한 AI 평가(XAI)를 제시합니다.

편집자: 실제 산업에 활용 가능한, 임팩트 있는 연구에 대한 요구가 학계 내에서 점점 커지고 있습니다. 슈어소프트테크는 현재 어떤 형태로 학계와 협력하고 있으며, 기업의 입장에서 학계가 더 집중하길 바라는 분야나 연구 방향이 있다면 말씀 부탁드립니다.

최경화 상무: 저희는 인턴십, 협력 체결, 교육 인프라 제공, 공동 연구 등 다양한 형태의 산학협력 활동을 진행하고 있습니다. 우선 20여개의 학교와의 인턴십을 통해 실제 국방/자동차 등 고신뢰 산업 분야의 검증 프로젝트에 참여하여 전문성을 키울 기회를 제공하고 있습니다. 또한 다양한 대학과 MOU 등 협력 관계를 맺고 국내 SW 및 AI 인재 양성, 산업 실무 고도화에 협력하고 있습니다. 경상국립대학교와는 우주항공 방위산업 발전 및 전문인력 양성을

위한 MOU를 체결하고, 소프트웨어 테스트 장비 지원을 통해 인력을 양성하고 있고, 국민대학교와는 미래자동차 산업실무형 인재 양성을 위한 산학협력 협약을 체결하고 SW 개발 검증 도구 및 실습 교육 인프라를 지원하고 있습니다. 그 외에도 실무적으로 필요한 기술들을 확보하기 위해 주요 대학의 연구실과도 공동 연구를 진행하고 있습니다. 최근에는 KAIST, UNIST, 충북대와 함께 디버깅 자동화를 위한 연구를 진행했고, 부산대와 이미지 학습 데이터 증강 관련 연구도 진행하고 있습니다. 저희 회사가 기본적으로 소프트웨어 공학, 프로그래밍 언어, 인공지능 쪽에서 다양한 솔루션을 개발하고 있기 때문에 관련 분야에 대한 연구가 활발히 이루어지고, 실제 산업 현장에서 적용될 수 있도록 기술과 연구인력의 교류가 활발해지기를 기대합니다.

편집자: 한국은 아직 SW 검증 자동화 분야에서 해외에 비해 관심이나 인프라가 부족하다는 평가가 있습니다. 이러한 현실을 어떻게 보고 계시며, 글로벌 진출, 신산업 확장, AI 검증 분야 진입 등을 포함한 슈어소프트테크의 미래 비전 및 전략에 대해 말씀 부탁드립니다.

최경화 상무: 한국은 SW 검증 자동화 기반 기술력과 응용 역량을 꾸준히 향상시키고 있으나 산업 내 적용율, 인프라, 생태계와 글로벌 네트워크, 표준화 주도력, 대규모 프로젝트 확산 경험 측면에서는 선두국가에 비해 부족한 부분이 존재합니다. 국내 소프트웨어 산업이 글로벌 경쟁력을 갖추기 위해서는 CI/CD/CT를 비롯한 소프트웨어 검증 인프라를 갖추고, 전문 인력 양성 및 품질 확보를 위한 비용 투자에 지속적 관심이 필요합니다.

슈어소프트테크는 소프트웨어, 그리고 AI의 안전성 확보를 통한 보다 나은

안전한 사회를 만들기 위한 비전을 가지고 글로벌 진출, AI를 비롯한 차세대 기술 선점 및 신산업 확장에 공격적으로 투자하는 도전적 성장 전략을 펼치고 있습니다. 중국 해외 법인이나 일본 현지 기업과의 전략적 제휴를 통한 현지화 전략을 통해 글로벌 시장을 공략하고 있고, 빅데이터 플랫폼 및 대용량 데이터 분석, AI 분석 솔루션에 강점을 가진 모비젠을 자회사로 합병하고, 피지컬 AI 및 로봇 공간 디지털화 기술을 확보한 블랙에의 전략적 지분 투자를 통해 국내 최대 강점인 자동차·국방·원자력 뿐 아니라 자율주행, E-모빌리티, 미래 로봇, 의료 등 신산업으로 포트폴리오를 빠르게 확장하고 있습니다. 또한 AI로 테스트(Test by AI)하고 AI를 테스트(Test of AI)하는 두 가지 AI 테스트 전략을 가지고 AI 기술

개발을 하고 있습니다. 첫번째 Test by AI는 AI로 테스트를 비롯한 소프트웨어 공학 Task들의 생산성, 효율성을 높이는 전략으로 ALIRA-AI를 통해 산업별 맞춤 AI 에이전트 솔루션을 공급하고 있고, 두번째 Test of AI는 AI 모델/AI 시스템 자체의 신뢰성, 강건성, 윤리성을 확보하는 전략으로 VERIFAI-M을 핵심 솔루션으로 자체 개발해서 공급하고 있습니다.

슈어소프트테크는 생성형 AI, 피지컬 AI, 디지털 트윈, 멀티 에이전트 등 미래 SW·AI 트렌드에 맞는 기술 투자를 지속하여 'AI 안전·신뢰성 검증'을 기반으로 미래 산업의 디지털 전환과 혁신을 통해 성장함으로써 글로벌 SW·AI 신뢰성 인증 및 검증 플랫폼 리더로 도약하고자 합니다.

슈어소프트테크 소개

SOFTWARE for **SAFE WORLD**

24년간 미션 크리티컬 산업에서 이어 온 소프트웨어 안전 미션, **AI 시대에도 그 여정은 계속됩니다.**

SURESOFT	회사명 슈어소프트테크 주식회사 대표이사 배현섭 본사위치 경기도 성남시 수정구 판교 제2 테크노밸리	설립연도 2002년 상장연도 2023년 홈페이지 www.suresofttech.com
-----------------	--	---

AI로 여는 SDV의 미래



SDV 검증 by AI,
시뮬레이션 기반 자율주행 검증으로
차세대 모빌리티 선도

K-방산의 신뢰성 주도



AI-powered SW 시험 도구로
무기체계 신뢰성 지원,
K-방산의 글로벌 경쟁력 강화

SMR/원전 안전성 강화



SMR 검증·인허가 지원,
AI 기반 보안 기술로
SMR/원전 안전성 강화

※ SDV(Software Defined Vehicle): 소프트웨어로 차량의 기능과 작동을 제어하는 자동차 ※ SMR(Small Modular Reactor): 전기출력 300MWe 이하의 공장 제작 후 현장에서 조립하는 소형 원자로



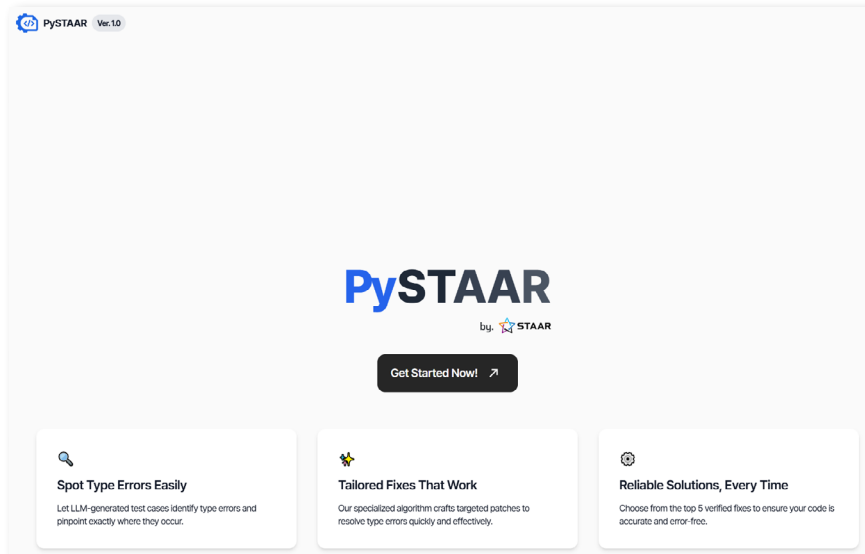
소사이어티 광장

축하합니다!

- 임용
 - 안가빈 박사 (KAIST, 지도교수: 유신), 고려대학교 컴퓨터학과 조교수 임용, 2025.09
 - 김기섭 박사 (룩셈부르크 대학, 지도교수: Tegawendé F. Bissyande, 김동선), DGIST 전기전자컴퓨터공학과 조교수 임용, 2025.08
- 퇴임
 - 강성원 교수, KAIST 전산학부 정년퇴임, 2025.08
- 이직
 - 김태호 교수, IITP SW 분야 PM, 2025.05

소개합니다!

- 오픈 저장소 소개: PySTAAR <https://pystaar.org/>
 - 소프트웨어재난연구센터(센터장: 경북대학교 최윤자 교수)에서 Python 코드의 타입 오류를 자동으로 수정해주는 도구인 PySTAAR를 공개합니다. 웹인터페이스를 통해 쉽게 도구를 사용할 수 있으니 관심 부탁드립니다.



출간

- KAIST 강성원 교수가 저서『연역 논리학: 논증의 증명과 반박(바른복스)』를 집필하여 2025년 4월 출간하였습니다.



연역 논리학은 전제들이 참이면 결론도 필연적으로 참인지 판별하게 해주는 학문이다. 이 책은 엄밀하고 간결한 연역 논리학 입문서이다. 기존의 책들과 달리 논증의 증명 방법과 반박 방법의 소개를 통해 연역 논리학을 독자에게 전달한다. 이를 위해 명제 논리와 술어 논리의 초보적인 논증 증명방법과 반박방법을 먼저 소개하고, 고급 기법으로서 자연연역과 진리나무 방법을 소개한다. 이 책은 연역 논리학과 이산수학을 공부해야 하는 이학 공학 분야의 대학생들, 연역 논리학의 논증 증명 방법과 반박 방법에 관심 있는 철학도, 인문학도, 사회과학도 및 일반인들을 위해 쓰였다.

강성원 교수의 저서로는 『소프트웨어 아키텍처로의 초대(2012)』, 『체계적인 소프트웨어 제품라인 개발(2017)』, 공저서로는 『소프트웨어 제품라인 개발 입문(2021, 이지현, 한영훈, 정필수 공저)』, 번역서로는 『지식인의 자격(2024)』이 있다.



2025 국내외 학술대회

Full name	학술대회	대회일자	논문마감	URL	기준 (행사, 제출)
International Systems and Software Product Line Conference	SPLC 2025	2025/09/01~09/05		https://2025.splc.net/	행사
Requirements Engineering 2025	RE 2025	2025/09/01~09/05		https://conf.researchr.org/home/RE-2025	행사
International Conference on Software Maintenance and Evolution	ICSME 2025	2025/09/07~09/12		https://conf.researchr.org/home/icsme-2025	행사
International Conference on Model Driven Engineering Language and Systems	MODELS 2025	2025/10/05~10/10		https://conf.researchr.org/home/models-2025	행사
Conference on Object-Oriented Programming, System, Languages, and Applications	OOPSLA 2025	2025/10/12~10/18		https://2025.splashcon.org/track/OOPSLA	행사
International Symposium on Software Reliability Engineering	ISSRE 2025	2025/10/21~10/24		https://issre.github.io/2025/	행사
International Conference on Automated Software Engineering	ASE 2024	2025/11/16~11/20		https://conf.researchr.org/home/ase-2025	행사
Asia-Pacific Software Engineering Conference	APSEC 2025	2025/12/02~12/05		https://conf.researchr.org/home/apsec-2025	행사
International Conference on Software Analysis, Evolution and Reengineering	SANER 2026	2026/03/17~03/20	2025.10.16	https://conf.researchr.org/home/saner-2026	제출
International Conference on Software Engineering-Software Engineering in Practice	ICSE-SEIP	2026/04/12~04/18	2025.9.29	https://conf.researchr.org/track/icse-2026/icse-2026-software-engineering-in-practice	제출
Symposium on Software Engineering for Adaptive and Self-Managing Systems	ICSE-SEAMS 2026	2026/04/13~04/14	2025.10.23	https://conf.researchr.org/home/seams-2026	제출
International Conference on Software Testing, Verification and Validation	ICST 2026	2026/05/18~05/22	Early January 2026 (TBD)	https://conf.researchr.org/home/icst-2026	제출
Conference on Software Engineering Education and Training (CSEET)	CSEET	2026/06/20~06/22	2026.2.10	https://cseet26.techconf.org/	제출
International Symposium on Software Testing and Analysis	ISSTA 2025	2026/10/3~10/9	2026.1.29	https://conf.researchr.org/home/issta-2026	제출
Korea Software Congress	KSC 2025	2025/12/16~12/19	TBD	TBD	행사, 제출
Korea Conference on Software Engineering	KCSE 2026	2026/02/04~02/06	TBD	TBD	행사, 제출



: 기고문 및 소식 모집



Dream



Think



Idea



come true

소프트웨어공학 소사이어티 소식지는 여러 연구자분들의 생각과 소식을 나누는 광장입니다. 다음과 같은 구성으로 소식지를 구성하고자 하오니, 여러분들의 적극적인 참여를 바랍니다. 투고글의 형식은 자유형식이며, 분량은 A4 기준 2~4페이지입니다.



- 기고문: 소프트웨어공학 및 소사이어티에 대한 생각 (자유주제)
- 신진연구자 소개: 만 40세 이하 또는 박사학위 취득 후 7년 이내의 연구자 소개
- 국내외 학술행사 소개: 주요 학술행사 소개, 학술행사 참여 후기 등
- 기관소개: 소프트웨어공학연구 관련기관 소개
- 소사이어티 광장: 소사이어티의 새로운 소식 나눔

▶ 소사이어티 알림

- 소프트웨어공학 소사이어티에서는 매년 소프트웨어공학 우수논문상을 추천하여 시상하고 있습니다. 최우수 학술대회에 논문 발표로 참가하는 학생에게 장려금(약 100만원 수준)을 지원할 예정입니다.
- 소프트웨어공학 소사이어티 소개 동영상 : <https://www.youtube.com/watch?v=HWGsy-Pyle0>
- 소프트웨어공학 소사이어티 페이스북 : <https://www.facebook.com/groups/668196744037453>

제출방법

- 이메일 제출 (소프트웨어공학 소사이어티 편집부: ksepup@gmail.com)

문의처

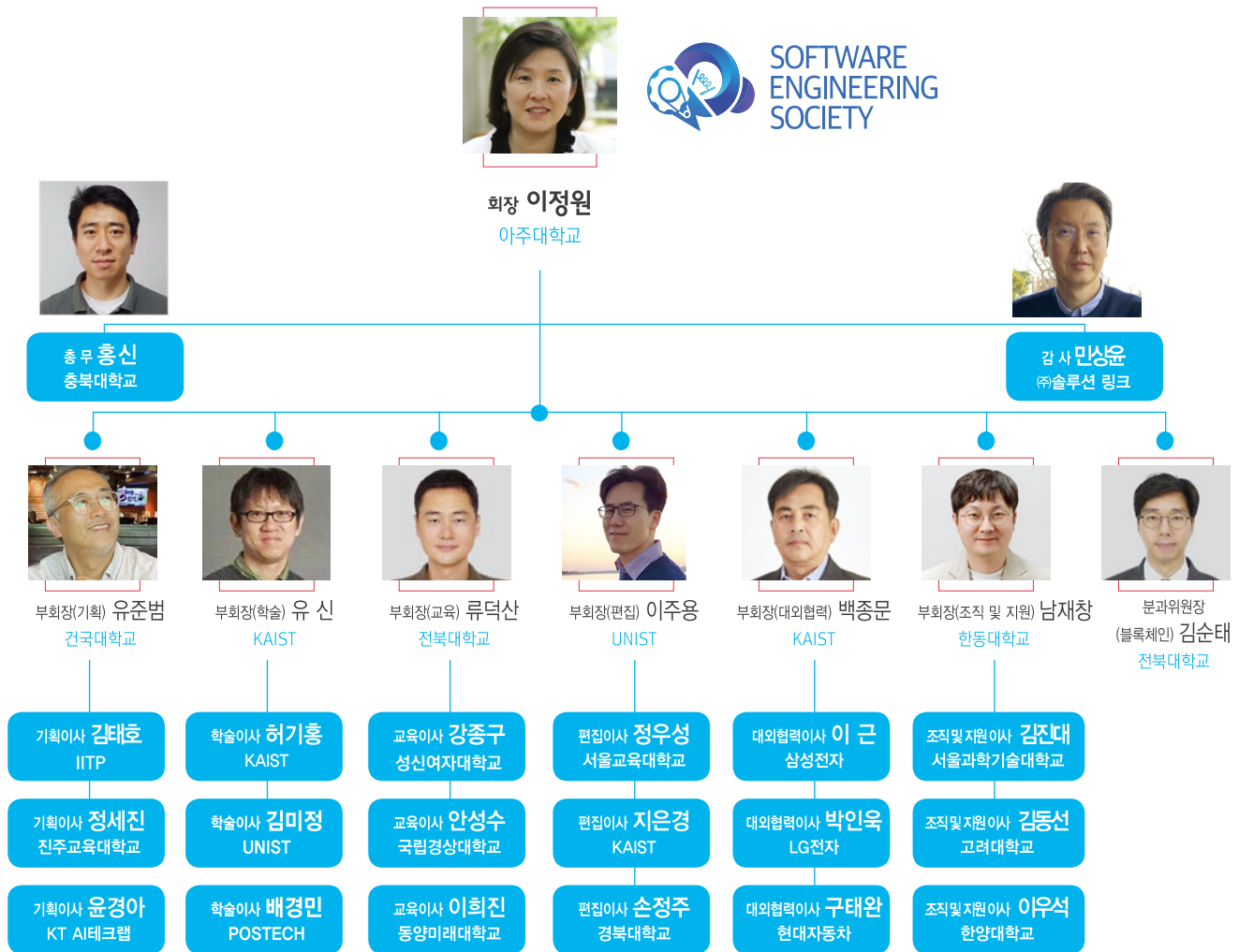
- 이주용 교수 (UNIST, 052-217-2123, jooyong@unist.ac.kr)
- 정우성 교수 (서울교육대학교, 02-3475-2562, wsjung@snue.ac.kr)
- 지은경 교수 (한국과학기술원, 042-350-7810, ekjee@se.kaist.ac.kr)
- 손정주 교수 (경북대학교, 053-950-5556, jeongju.sohn@knu.ac.kr)



SOFTWARE
ENGINEERING
SOCIETY



소사이어티 조직도



발행정보

발행일 2025년 9월 1일

발행인 이정원

발행처 사단법인 한국정보과학회 소프트웨어공학소사이어티

연락처 경기도 수원시 영통구 월드컵로 206 아주대학교 원천관 305호 (우: 16499)

이정원 (전화: 031-219-1813, 홈페이지: <https://eslab.ajou.ac.kr/members/professor>)

홈페이지 <https://www.sigsoft.or.kr>



슈어소프트테크는 지난 20여년이 넘는 시간 동안
자동차, 국방, 원자력 등 다양한 Mission-Critical 산업에서
SW를 통해 더 안전한 세상을 만들기위해 노력해왔습니다



AI로 여는 SDV의 미래



K-방산의 신뢰성 주도



SMR/원전 안전성 강화



End to End V&V Solution

요구사항 작성부터 시스템 시험까지 End to End 검증/개발 서비스 제공
: 코드 검증 솔루션, 모델 검증 솔루션, 시스템 검증 솔루션



DX Solution

가상화/시뮬레이션 등 미래 기술로 SW 산업의 혁신을 주도
: 가상화 기술, 차량 제어 개발 기술, 자율주행 기술, 사이버 보안 기술



Big Data · AI Solution

데이터 과학 및 AI 관련 서비스를 제공하여 비즈니스 혁신 지원
: 빅데이터 관리 및 시각화, AI 검증, 빅데이터 및 AI 컨설팅, AI 경량화



KOLAS 국제공인시험기관

ISO/IEC 국제 표준 기반 공인 시험성적서 발급
: 일반 시험성적서 발급, KOLAS 공인시험성적서 발급





SOFTWARE
ENGINEERING
SOCIETY

사단법인 한국정보과학회 소프트웨어공학소사이어티

주소: 경기도 수원시 영통구 월드컵로 206 아주대학교 원천관 305호 (우: 16499)

이정원 (전화: 031-219-1813, 홈페이지: <https://eslab.ajou.ac.kr/members/professor>)

홈페이지: <https://www.sigsoft.or.kr>