소프트웨어 오류 자동 탐지 및 수정 기술

오학주

고려대학교 정보대학 컴퓨터학과



2026.11.13@미래SW기술포럼

소프트웨어 오류 문제

- 소프트웨어 오류 = 사회 모든 영역에서 발생



금융



항공



자동차



의료

- 사회경제적 비용 = 2.4조 달러 (미국, 2022)

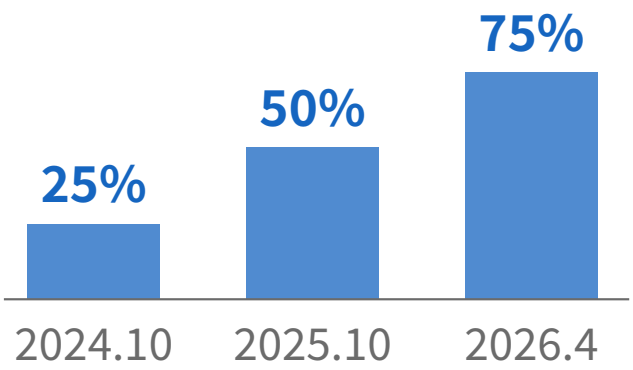


The Cost of Poor Software Quality in the US: A 2022 Report. CISQ

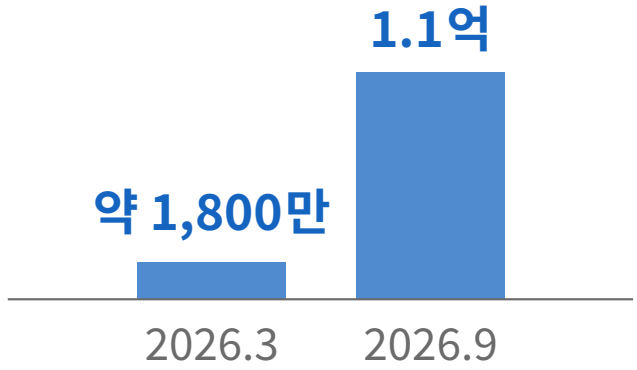
AI 생성 코드 검증 문제

- AI가 작성하는 코드가 급격히 증가

구글 신규 코드 중 AI가 생성한 비율



코딩 에이전트 PR 수 (GitHub)



- 신뢰성 문제는 여전: 검증이 코드 증가 속도를 따라가지 못함

45% AI가 생성한 코드에서 보안 취약점이 발생한 비율

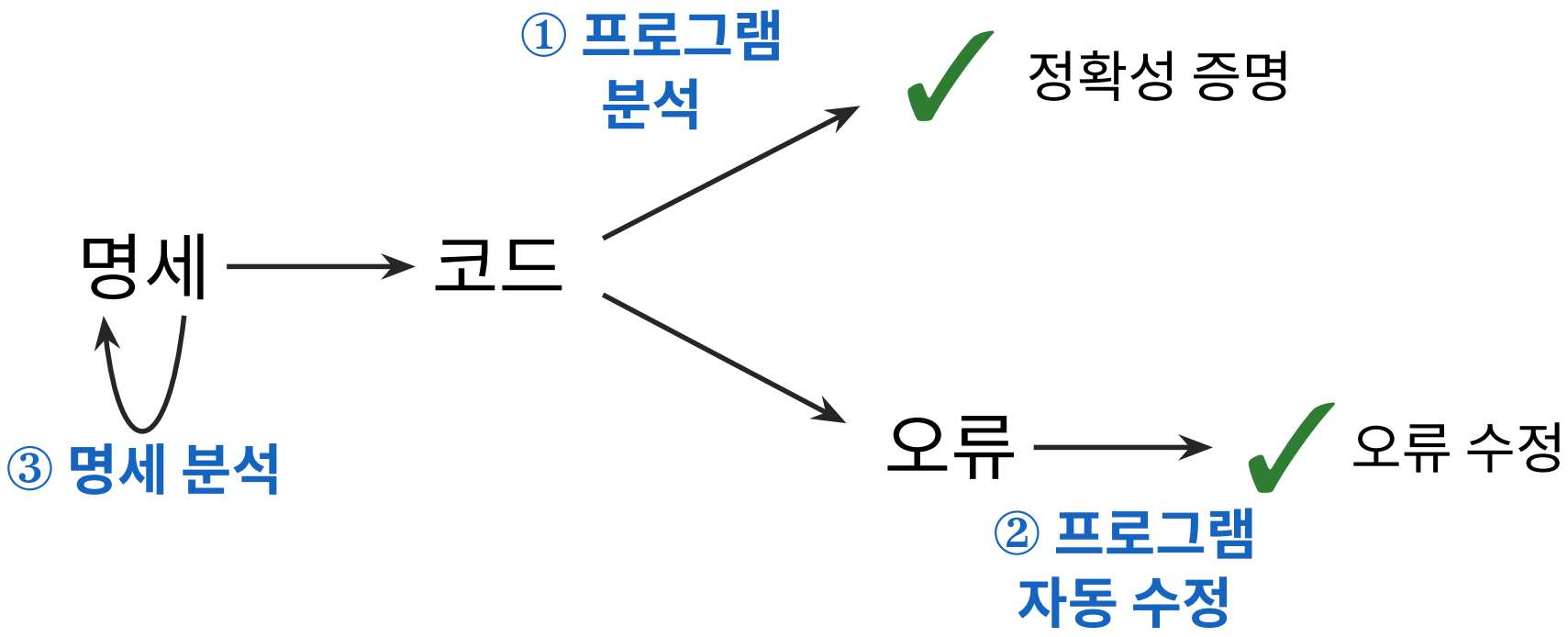
46% AI가 생성한 결과의 정확성을 신뢰하지 않는 개발자 비율

AI가 생성하는 스케일에 맞는 코드 검증 기술 필요

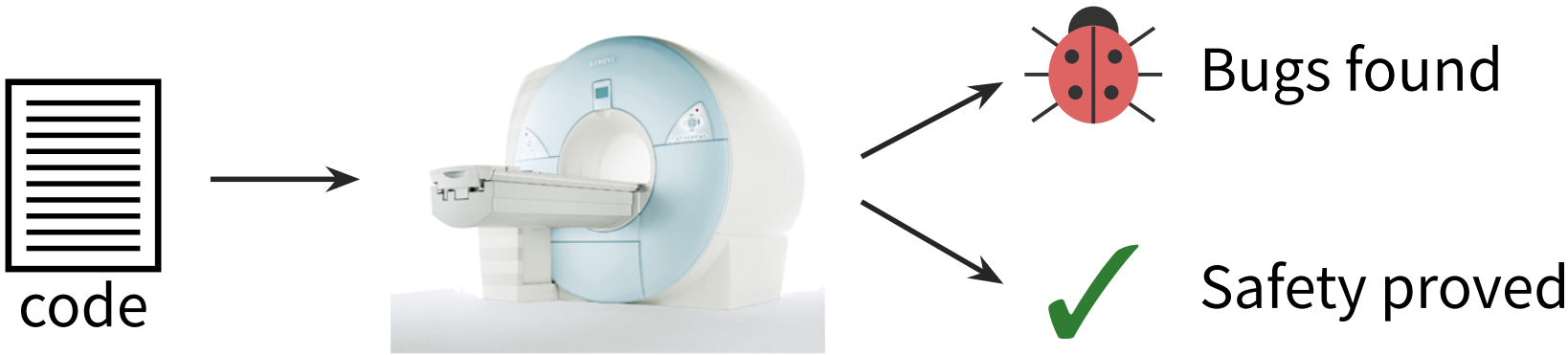
GitHub Octoverse 2026 Veracode, 2025 GenAI Code Security Report Stack Overflow, 2025 Developer Survey

연구 목표

- Q) 어떻게 안전한 소프트웨어를 손쉽게 만들 것인가?
- A) 소프트웨어 **오류 자동 검출 및 자동 수정 기술** 개발



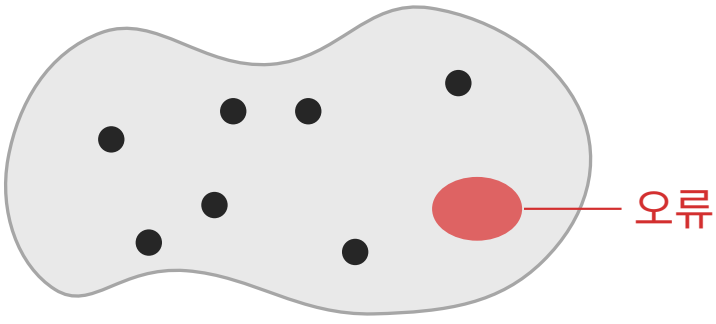
① 프로그램 분석



- 소프트웨어의 실행 성질을 엄밀히 확인하는 기술
  - 정적 분석: 실행 전 확인 (요약 해석, 모델 체킹 등)
  - 동적 분석: 실행 중 확인 (퍼징, 기호 실행 등)
- 소프트웨어 산업에서 적극적으로 활용

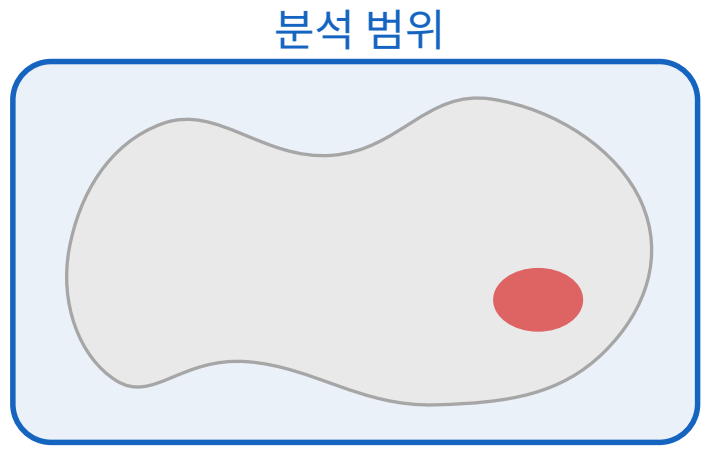
안전한 프로그램 분석 및 검증

- 특히 안전한(sound) 프로그램 분석 및 검증 기술 개발에 중점
  - 모든 실행을 수학적으로 분석하여 모든 오류를 찾는 기술



테스팅

일부 실행만 확인



안전한 분석 및 검증

모든 실행을 수학적으로 분석

- AI가 코드를 생산하는 시대에 특히 중요

사례 1) C 프로그램 오류 탐지

- 논리 기반 오류 탐지 및 검증: 프로그램의 모든 실행에 대해 오류가 없음을 보이거나 코너 케이스의 까다로운 오류를 탐지

Benchmarks	Versions	Error Types	Bug-Triggering Inputs
vim	8.1	Non-termination	`K1!1000100100111110(`
		Abnormal-termination	`H:w>>`"`\` [press 'Enter']
	5.7	Segmentation fault	`=ipI\-9~qOqw`
		Non-termination	`v(ipaprq&T$T`
gawk	4.2.1	Memory-exhaustion	`'+E_Q$h+w$8==++$6E8#'`
		Abnormal-termination	`'f[][][][][y]^/#['`
	3.0.3	Non-termination	`'$g?E2^=-E-2"?^+$=":/?/#["'`
grep	3.1	Abnormal-termination	`'\(\)\1*?*?|\W*\1W*'`
		Segmentation fault	`'\(\)\1^*@*\?\1*\+*\?'`
	2.2	Segmentation fault	`"_^^*9|^\(\)\'\1*$"`
		Non-termination	`'\({**+**\)*\++*\1*\+'`
sed	1.17	Segmentation fault	`'{:};:C;b'`

- 주요 논문
  - Diver: Oracle-Guided SMT Solver Testing with Unrestricted Random Mutations. ICSE 2023
  - SymTuner: Maximizing the Power of Symbolic Execution by Adaptively Tuning External Parameters. ICSE 2022
  - Enhancing Dynamic Symbolic Execution by Automatically Learning Search Heuristics. TSE 2021
  - Making Symbolic Execution Promising by Learning Aggressive State-Pruning Strategy. FSE 2020

사례 2) 안전 필수 소프트웨어 검증 및 테스트

- 스마트 컨트랙트 검증: 취약점 탐지 및 안전성 증명

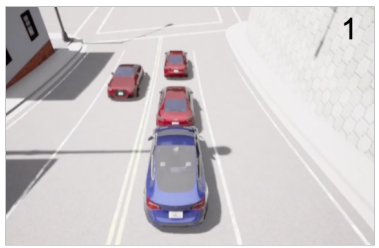
```
function transferProxy(address from, address to,
                       uint value, uint fee) ... {
  if (balance[from] < fee + value) revert();
  balance[to] += value;
  balance[msg.sender] += fee;
  balance[from] -= value + fee;
}
```

fee + value가 오버플로로 작아져 잔액 검사를 통과

최근 피해 사례 (2025~2026)

Cetus	**약 2.2억 달러**
Balancer	**약 1.3억 달러**
Truebit	**약 2,600만 달러**

- 자율주행 시스템 테스트: Autoware[1]에서 신규 오류 23개 발견



내리막길 추돌



보행자 충돌



안개 속 정지



교차로 충돌

- 주요 논문
  - Prunario: Testing Autonomous Driving Systems by Pruning Likely Redundant Scenarios. OOPSLA 2026
  - SmartFix: Fixing Vulnerable Smart Contracts by Accelerating Generate-and-Verify Repair … FSE 2023
  - SmarTest: Effectively Hunting Vulnerable Transaction Sequences in Smart Contracts … USENIX Security 2021
  - VeriSmart: A Highly Precise Safety Verifier for Ethereum Smart Contracts. S&P 2020

1) Autoware (https://autoware.org): 자율 발레파킹, 화물 운송 등 산업 현장에서 쓰이는 오픈소스 자율주행 소프트웨어

사례 3) Java, Python, 양자 프로그램

- Java 단위 테스트 자동 생성: 프로그램 분석과 LLM을 결합

Java 코드 → **프로그램 분석 + LLM** → 단위 테스트 (JUnit) 산업 현장 적용

- Python 엄밀 검증: 엄밀하게 정의된 중간언어로 변환하여 검증

Python 프로그램 → **중간언어** (MIR) → 안전한 정적 분석, 연역 검증

- 양자 프로그램 검증 및 테스팅: 오류 확률 증폭으로 측정 비용 절감

오류 탐지 **최대 3.6배** 가속

오류 없음 검증 **평균 2.4~7.0배** 가속

- 주요 논문
  - EvoLLM: High-Coverage, Readable, and Token-Efficient Unit Test Generation for Industrial Java Codebases. ASE 2026
  - SCOUT: A Scenario-Guided Unit Test Generation Tool for Compact Open-Source LLMs. ISSTA 2026
  - Effective Unit Test Generation for Java Null Pointer Exceptions. ASE 2024
  - Statistical Testing of Quantum Programs via Fixed-Point Amplitude Amplification. OOPSLA 2024
  - Quantum Probabilistic Model Checking for Time-Bounded Properties. OOPSLA 2024

② 오류 자동 수정 기술

- 개발 생산성은 소프트웨어 산업 경쟁력을 결정하는 주 요인
- 소프트웨어 개발에서 가장 생산성이 낮은 단계는 디버깅
  - 개발자들이 가장 부담스러워하는 단계[1)]
  - 개발자들은 전체 시간의 절반을 디버깅에 사용[2)]
  - 상용 소프트웨어 오류를 수정하는 데 평균 200일 소요[3)]
- AI 에이전트가 생성한 코드에 대한 디버깅 부담은 여전히 존재[4,5)]
  - 정확하지 않은 코드가 많아 디버깅 및 코드 리뷰가 병목

개발 시간 비중[2)]



1) The art of software testing (3rd edition). 2012 2) Reversible debugging software. University of Cambridge. 2013
3) How long did it take to fix bugs? Mining Software Repositories. 2006 4) Stack Overflow developer survey 2025
5) Many SWE-bench-passing PRs would not be merged into main. METR 2026

Linux Kernel 사례

Linux 커널 USB 드라이버

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

메모리 할당

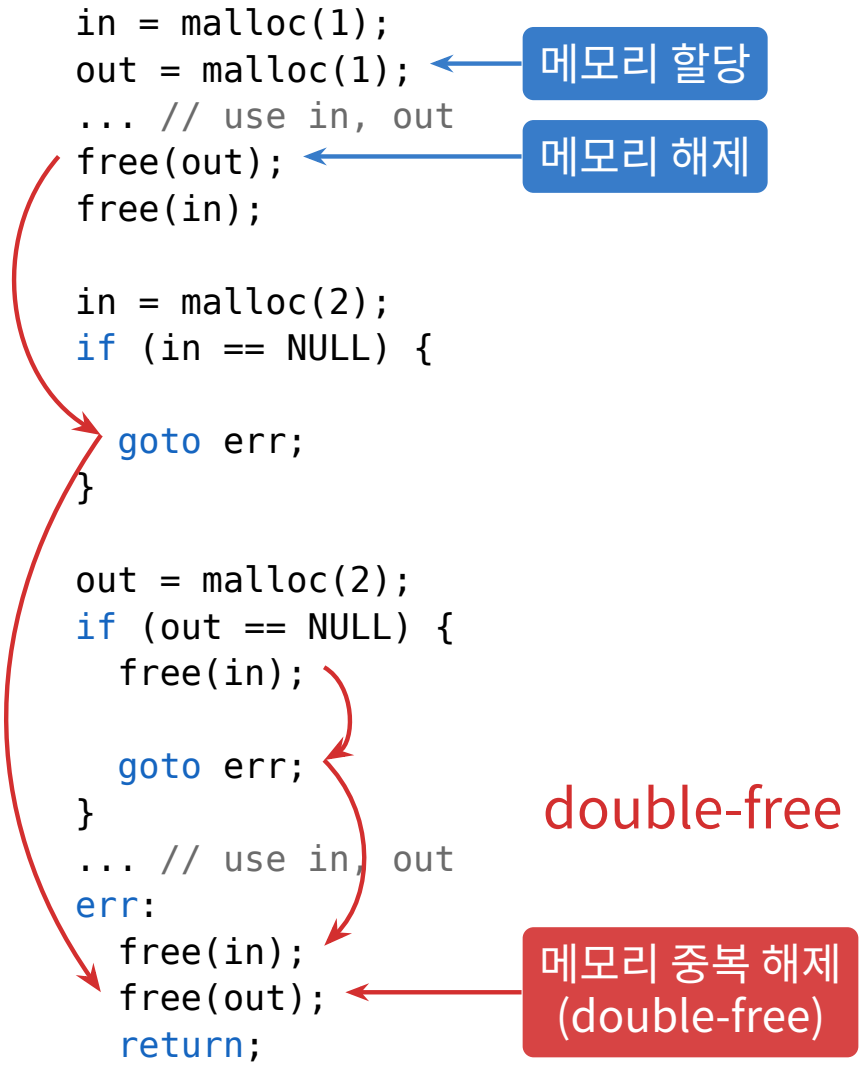
메모리 해제

Linux Kernel 사례

Linux 커널 USB 드라이버

메모리 중복 해제 (double-free)

오류 처리 경로에서
이미 해제한 메모리를 다시 해제



Linux Kernel 사례



디버깅의 어려움 1:
오류가 제거되었는지
확신하기 어려움

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

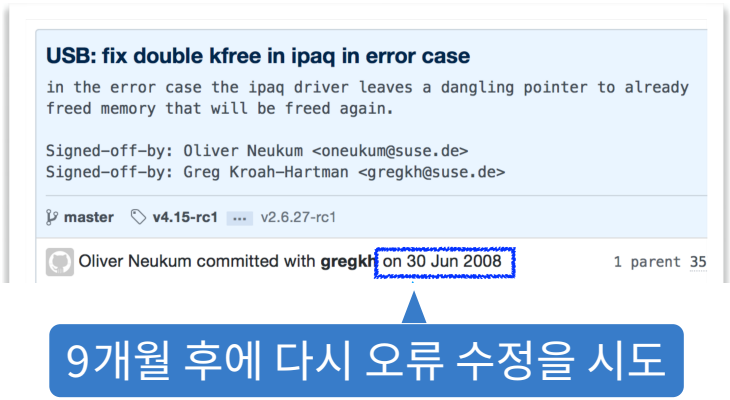
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Linux Kernel 사례

memory leak

디버깅의 어려움 2:
오류 수정 과정에서
새로운 오류가 발생



9개월 후에 다시 오류 수정을 시도

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Linux Kernel 사례

디버깅의 어려움 3:
오류는 제거했지만
코드 품질이 떨어짐



오류 발견에서 수정까지 총 10개월 소요

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);
out = NULL;
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

오류 자동 수정 기술

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  goto err;
}
... // use in, out
err:
  free(in);  // double-free
  free(out); // double-free
  return;
```



- ✓ 개발 생산성 향상
- ✓ SW 품질 향상
- ✓ 패치의 올바름 검증

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

개발 기술 vs. Gen AI

- AI의 가장 큰 약점: 생성된 패치를 100% 신뢰할 수 없음
- 예: 오픈소스 snort-2.9.13 (32만 줄)에서 발생한 memory leak

```
int append_data(Node *node, int *ndata) {
  Node *n = malloc(sizeof(Node));
  if (!n) return -1;                        // failed to be appended
  n->data = ndata;                          // linked into the list
  n->next = node->next; node->next = n;
  return 0;                                 // successfully appended
}
for (Node *node = lx; node; node = node->next) {
  int *dptr = malloc(sizeof(int));
  if (!dptr) return;
  *dptr = *(node->data);
  append_data(ly, dptr);                    // memory leak
}
```

GPT-4가 생성한 패치

```
append_data(ly, dptr);

free(dptr);
```

✗ 리스트에 연결된 메모리 해제

⇒ use-after-free, double-free

개발 기술이 생성한 패치

```
if (append_data(ly, dptr) == -1)

  free(dptr);
```

✓ 추가 실패 시에만 해제

⇒ 안전성 검증

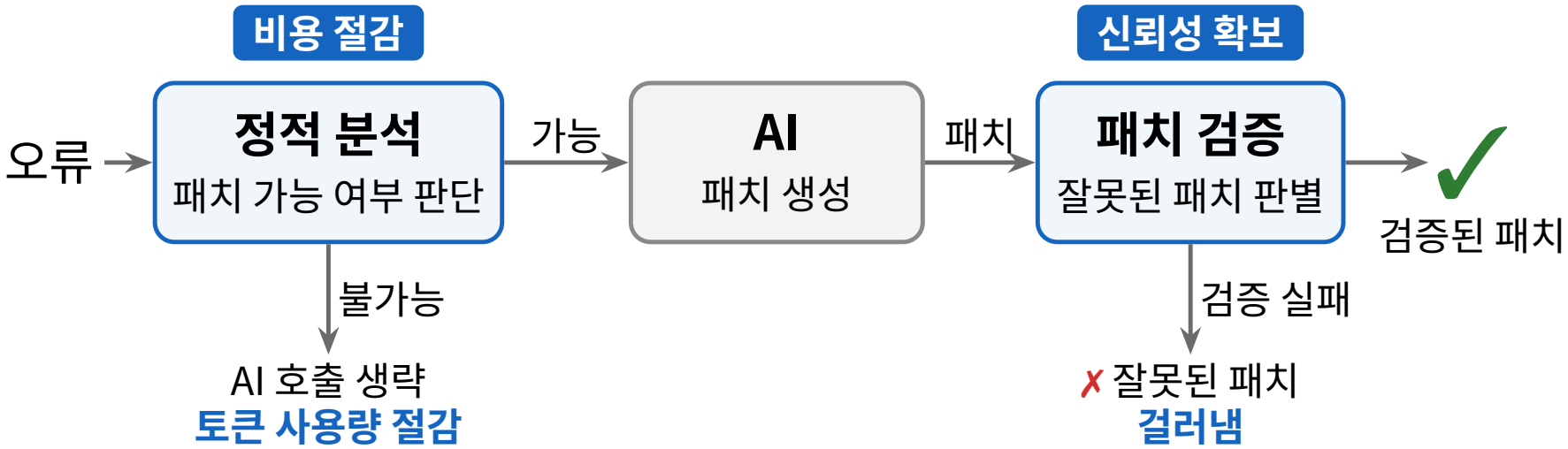
대상 언어 및 오류

대상	오류, 취약점	특징
C/C++	메모리 누수, double-free, use-after-free	새 오류가 없음을 보장 산업 현장 적용
Java	널 포인터 예외	테스트 없이 명세를 추론해 검증
Python	타입 오류	동적 언어의 오류를 정적으로 탐지하고 수정
Solidity	정수 오버플로 등 보안 취약점	검증기로 안전성이 확인된 패치

- 주요 논문
  - TypeCare: Boosting Python Type Inference Models via Context-Aware Re-Ranking and Augmentation. ICSE 2026
  - Towards Effective Static Type-Error Detection for Python. ASE 2024
  - SmartFix: Fixing Vulnerable Smart Contracts by Accelerating Generate-and-Verify Repair … FSE 2023
  - PyTER: Effective Program Repair for Python Type Errors. FSE 2022
  - NPEX: Repairing Java Null Pointer Exceptions without Tests. ICSE 2022
  - SAVER: Scalable, Precise, and Safe Memory-Error Repair. ICSE 2020

3단계: AI 기반 패치 생성 및 검증 시스템

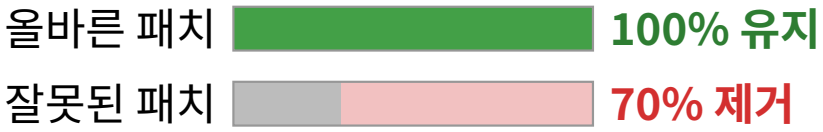
- AI는 패치를 잘 생성하지만 비용과 신뢰성이 문제



비용: 오류 수정에 드는 AI 비용



신뢰성: 패치 검증 결과

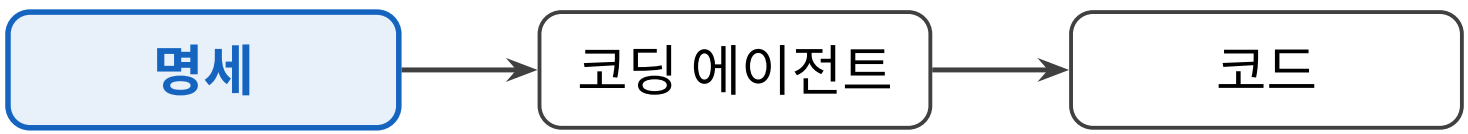


③ 명세 정적 분석

- **과거**: 사람이 코드를 작성
  - “프로그램”은 곧 코드, 코드 작성이 병목
  - 정적 분석의 대상도 코드



- **현재**: 개발자는 명세를, 코딩 에이전트가 코드를 작성
  - 명세가 새로운 “프로그램”, 좋은 명세 작성이 병목



연구 목표: 명세를 대상으로 하는 정적 분석

요약

- **연구 분야**: 소프트웨어 오류 자동 탐지 및 수정
- ① **프로그램 분석**: 오류를 찾거나 오류가 없음을 증명
  C, 안전 필수 소프트웨어, Java, Python, 양자 프로그램에 적용
- ② **오류 자동 수정**: 검증된 패치를 자동 생성
  C/C++, Java, Python, Solidity 대상
- ③ **명세 분석**: 명세를 대상으로 하는 프로그램 분석
  명세가 새로운 “프로그램”, 좋은 명세 작성이 병목

감사합니다!

오학주 hakjoo_oh@korea.ac.kr prl.korea.ac.kr