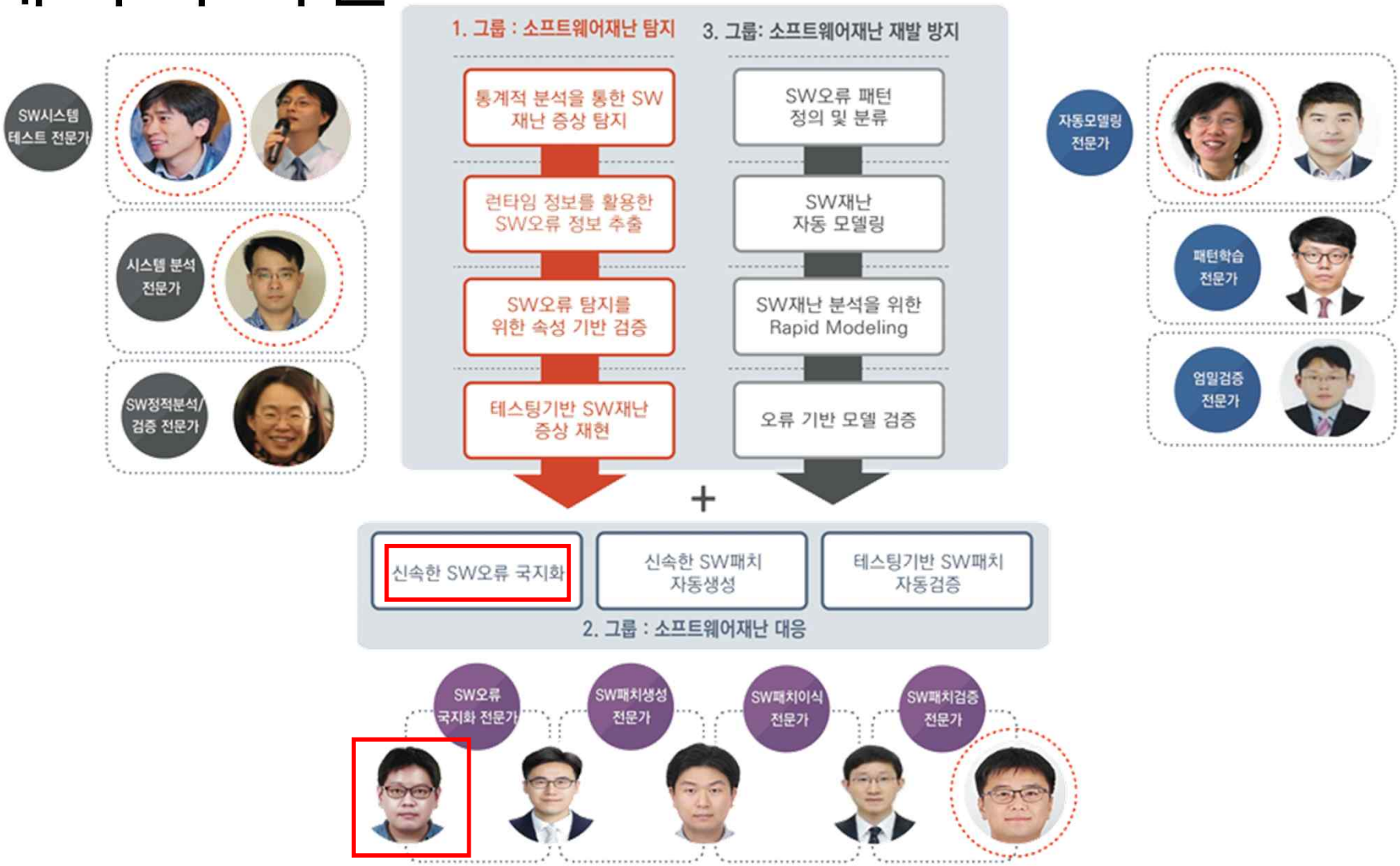
확장성 좋은 오류검출기술 연구

김윤호
한양대학교 컴퓨터소프트웨어학부

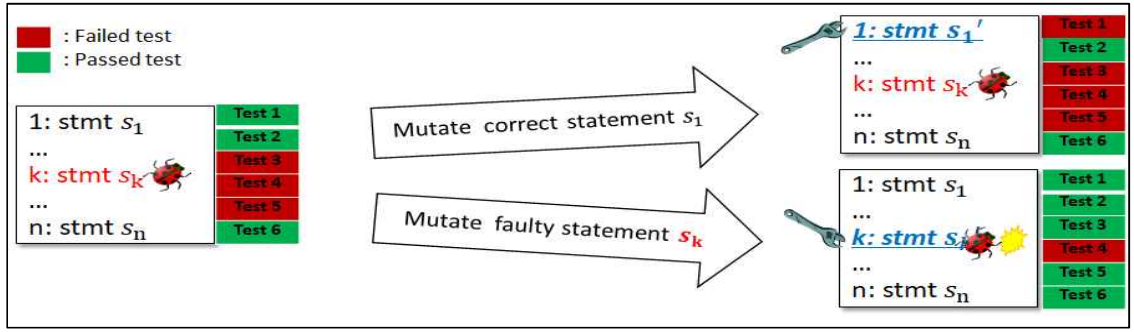
센터에서의 역할


SW시스템 테스트 전문가
시스템 분석 전문가
SW정적분석/검증 전문가
1. 그룹 : 소프트웨어재난 탐지
통계적 분석을 통한 SW 재난 증상 탐지
런타임 정보를 활용한 SW오류 정보 추출
SW오류 탐지를 위한 속성 기반 검증
테스팅기반 SW재난 증상 재현
3. 그룹: 소프트웨어재난 재발 방지
SW오류 패턴 정의 및 분류
SW재난 자동 모델링
SW재난 분석을 위한 Rapid Modeling
오류 기반 모델 검증
자동모델링 전문가
패턴학습 전문가
엄밀검증 전문가
+
신속한 SW오류 국지화
신속한 SW패치 자동생성
테스팅기반 SW패치 자동검증
2. 그룹 : 소프트웨어재난 대응
SW오류 국지화 전문가
SW패치생성 전문가
SW패치이식 전문가
SW패치검증 전문가

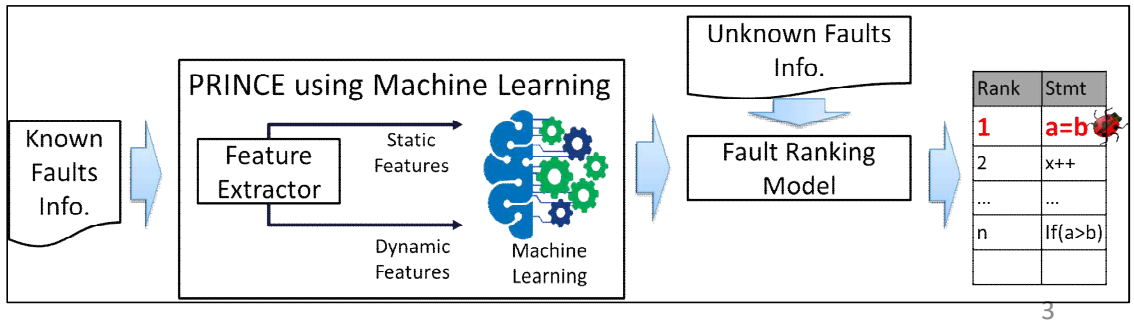
변이 분석을 사용한 정확한 오류 위치 찾기

1. MUSE [ICST 14]
   - 변이 분석을 활용한 정확한 오류 위치 찾기
2. MUSEUM [ASE 15, IST 17]
   - 변이 분석을 활용해 다중 언어 프로그램에서 정확한 오류 위치 찾기
3. PRINCE [TOSEM 19]
   - 변이분석에 기계학습까지 더해서 더 정확하게 오류 위치 찾기

MUSE

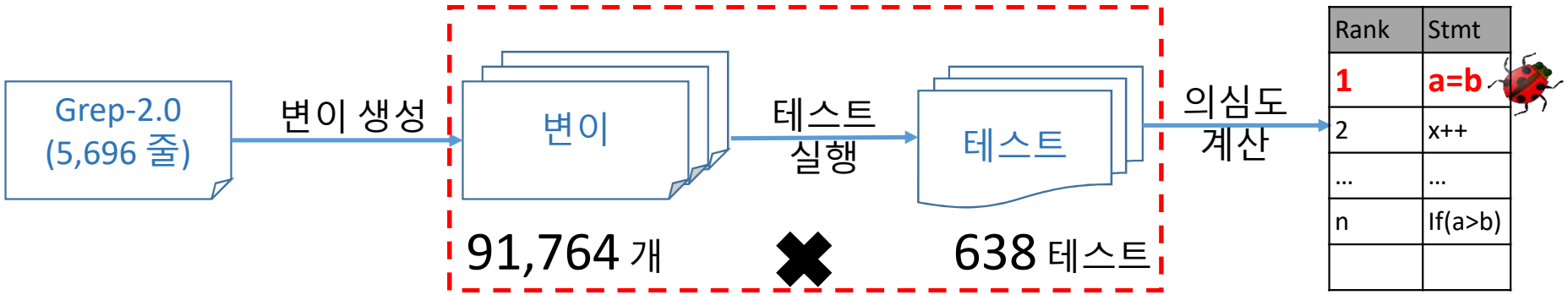


PRINCE



도전과제: 변이 분석이 너무 오래 걸려요

- 변이 분석은 시간 비용이 굉장히 비싸다
- 예: grep-2.0 에서 변이 분석을 사용해 오류 위치를 찾는 경우



- 대략 **~117 시간**

이전 결과 & 향후 계획(2022 여름 워크샵)

- 이전 결과
  - 오류 위치 찾는데 평균 7.2% 구문 탐색 필요 (기존 MUSE는 5.1%)
  - 학습 시간을 제외하고 오류 1개당 약 25분 시간 소요

- 향후 계획
  - 비싼 돈 들여서 학습했으면 최대한 우려먹자
    - 1.0 버전에서 학습한 모델이 있으면 1.15 버전까진 써먹을 수 있어야 남는 장사
    - Cross-version 성능 측정 실험 및 파인 튜닝 방법 연구
  - 더 좋은 학습 방법을 찾아보자
    - Random Forest가 과연 최선인가?
  - 자연어 채널을 사용한 최신 PMT 기술 도입
  - 학습 시간을 줄여보자

더 좋은 학습 방법 탐색

- 기존 학습 방법의 문제
  - 코드 메트릭에 의존한 코드 정보 학습
    - 정량적인 코드 메트릭이 코드의 의미 구조를 잘 반영할 수 있는가?
    - 프로그래밍 언어 의존적인 메트릭 정의
  - 과도한 모델 학습 데이터 및 시간 필요
    - 최대 생성된 변이 수의 제곱에 비례한 학습 데이터량 요구

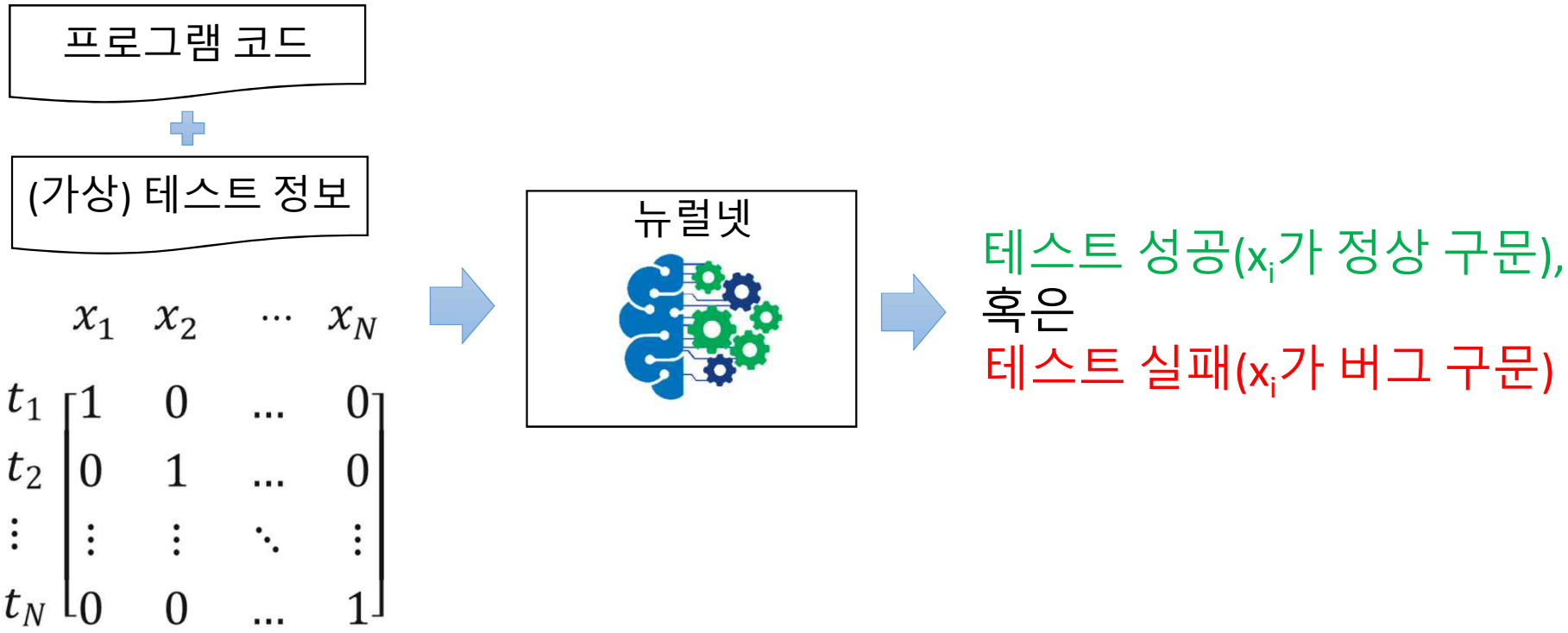
- 뉴럴넷 기반의 학습 방법 탐색
  - 코드 메트릭을 활용한 명시적인 특징 추출 불필요
  - 변이 수에 크게 영향받지 않는 학습 가능

아이디어

- 상상: 각 테스트가 한 번에 딱 하나의 구문만 실행한다면 오류 위치를 아주 쉽게 알 수 있지 않을까?

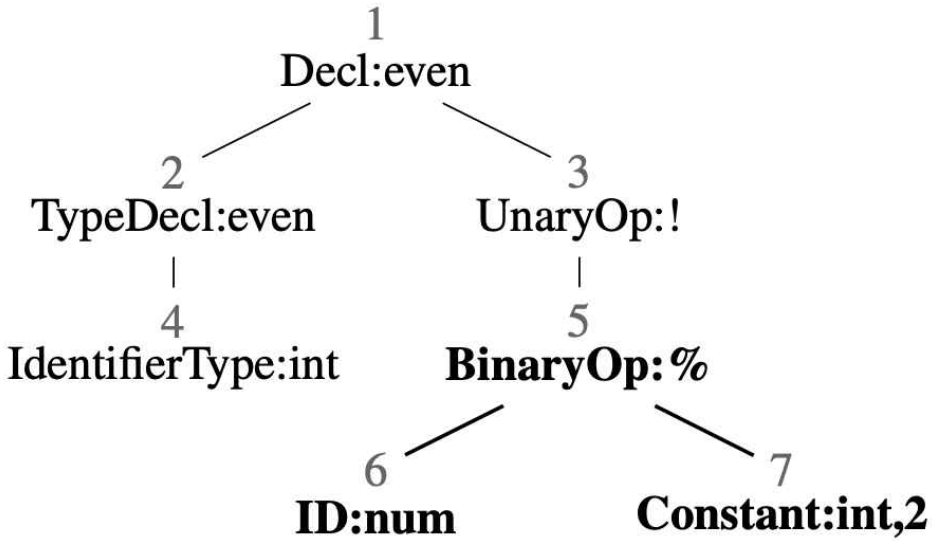
- 현실: 프로그램 코드와 (가상) 테스트 정보가 주어졌을 때 테스트 성공/실패 결과를 예측하는 뉴럴넷 모델을 만들고 예측해보자
  - 테스트 정보: 해당 테스트가 실행한 코드 커버리지 정보

테스트 결과 예측을 활용한 오류 위치 추정



프로그램 코드의 표현 방법

- 프로그램 코드의 AST를 2차원 벡터로 표현하여 뉴럴넷 학습에 활용 [Gupta et al., NeurIPS 2019]
- 예제: `int even = !(num%2)`

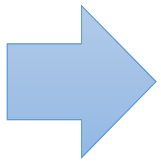


$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 0 \\ 3 & 5 & 0 \\ \mathbf{5} & \mathbf{6} & \mathbf{7} \end{bmatrix}$$

테스트 실행 정보의 표현 방법

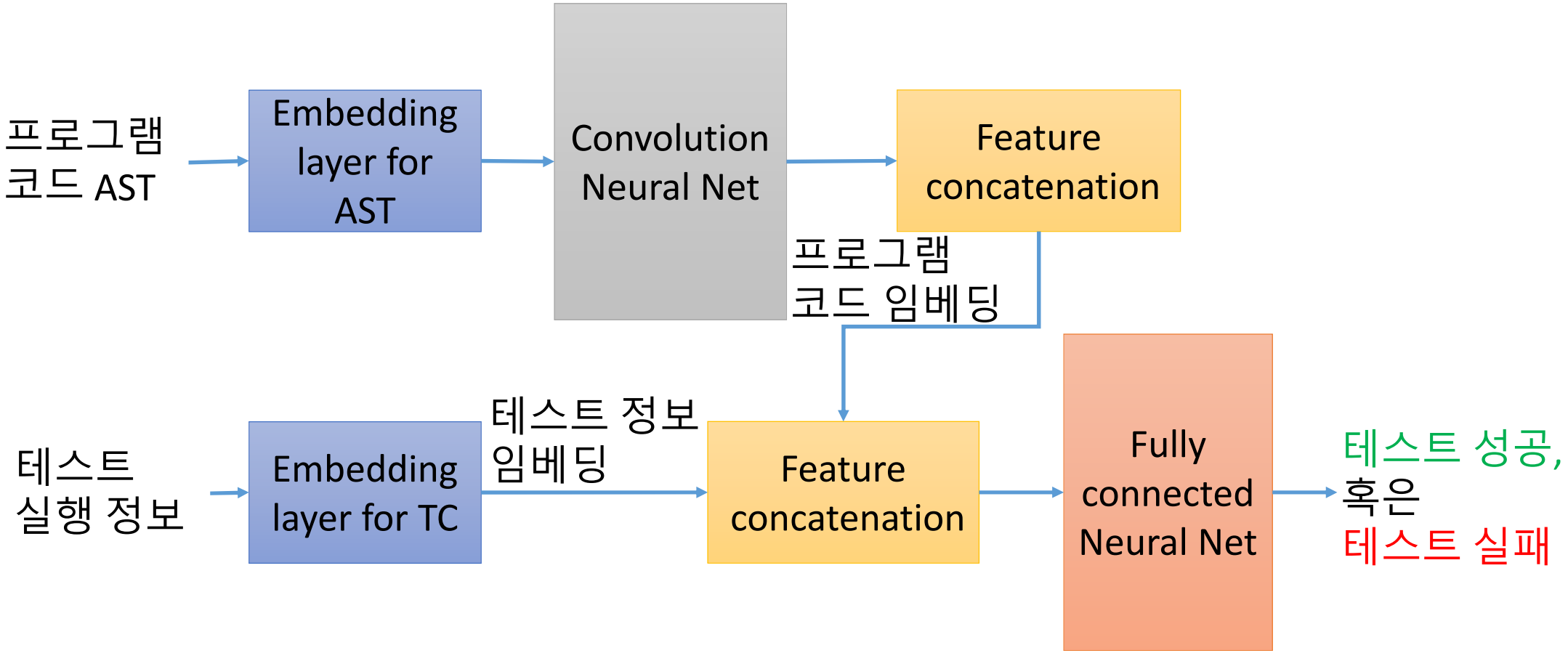
- 기존 커버리지 매트릭스를 AST 노드 정보를 활용하는 형태로 변형하여 표현함

	Line 1	…	Line n
TC 1	0	…	1
…	…	…	…
TC m	1	…	1



	AST 1	…	AST k
TC 1	0	…	1
…	…	…	…
TC m	0	…	0

신경망 구조


프로그램
코드 AST
Embedding layer for AST
Convolution Neural Net
Feature concatenation
프로그램
코드 임베딩
테스트
실행 정보
Embedding layer for TC
테스트 정보
임베딩
Feature concatenation
Fully connected Neural Net
테스트 성공,
혹은
테스트 실패

파인 튜닝

- 학습 데이터로 선행 학습된 모델이 현재 오류 위치를 찾고 싶은 프로그램과 테스트 실행 결과 정보에 아주 딱 맞지는 않음

- 기존 학습된 모델에 새로 주어진 정보를 바탕으로 맨 마지막 레이어로 Fully Connected 레이어를 추가하여 현재 주어진 프로그램의 정보를 반영함
  - 학습 데이터와 새로 주어진 정보의 데이터 경향이 크게 다르지 않다는 믿음
  - 학습 데이터에 비해 새로 주어진 정보의 양이 더 적음

연구 질문

- 신경망을 사용한 오류 위치 추정의 효용성
  - RQ1: 얼마나 정확하게 오류 위치를 찾을 수 있는가?
  - RQ2: 얼마나 빠르게 오류 위치를 찾을 수 있는가?

- 신경망 기반 오류 위치 추정 요소 기술의 효과성
  - RQ3: 변이 분석을 사용한 학습 데이터 생성은 효과적인가?
  - RQ4: 신경망 파인 튜닝은 효과적인가?

실험 설정

- 5개 SIR 벤치마크 프로그램의 75개 버그 대상
  - Bash와 Vim은 너무 커서 일단 제외

프로그램	버그 수	코드 줄 수	테스트 수
flex	19	7254	567
grep	18	5696	809
gzip	16	3040	208
make	19	9820	1006
sed	3	3980	360
Average	15.0	5879.8	275.8

- 구현 방법
  - Keras를 사용하여 신경망 구현
  - 프로그램 단위로 5-fold cross validation 수행

실험 결과 – RQ1:정확도

- 신경망을 사용할 때 오류 위치 탐색에 4.5% 구문 탐색이 필요하여 MUSE 보다 약 11% 향상

Expense(%)	MUSE	Op2	MUSE with PMT[2]	신경망
flex	13.7	19.4	14.5	6.4
grep	1.3	2.6	4.1	2.7
gzip	5.3	6.7	7.7	5.8
make	3.9	15.5	6.3	3.4
sed	1.2	11.4	3.4	2.2
평균	**5.1**	**11.1**	**7.2**	**4.5**

실험 결과 – RQ2: 실행 시간

- 신경망 추론: 1개 버그당 평균 약 10분 소요
  - 지난 결과 대비 버그 1개당 소요 시간 1/3으로 단축
- 사용 머신: AMD EPYC 7763 (최대 3.5GHz, 64코어), 512GB RAM, 2 * A100 GPU

시간(h)	MUSE	Op2	MUSE with PMT[2]	신경망
flex	475.7	0.01	0.4	0.2
grep	599.2	0.01	0.7	0.1
gzip	388.9	0.01	0.5	0.2
make	531.4	0.02	0.9	0.2
sed	362.8	0.01	0.8	0.1
버그 1개당 소요시간	496.4	0.01	0.6	0.2

향후 계획

- CrowdQuake 지원을 위한 Python/Java 지원 추가
- 변이를 사용한 학습 데이터 생성 및 파인 튜닝 결과 확인 및 분석
- 다양한 오류 데이터를 통한 검증 및 신경망 구조 개선