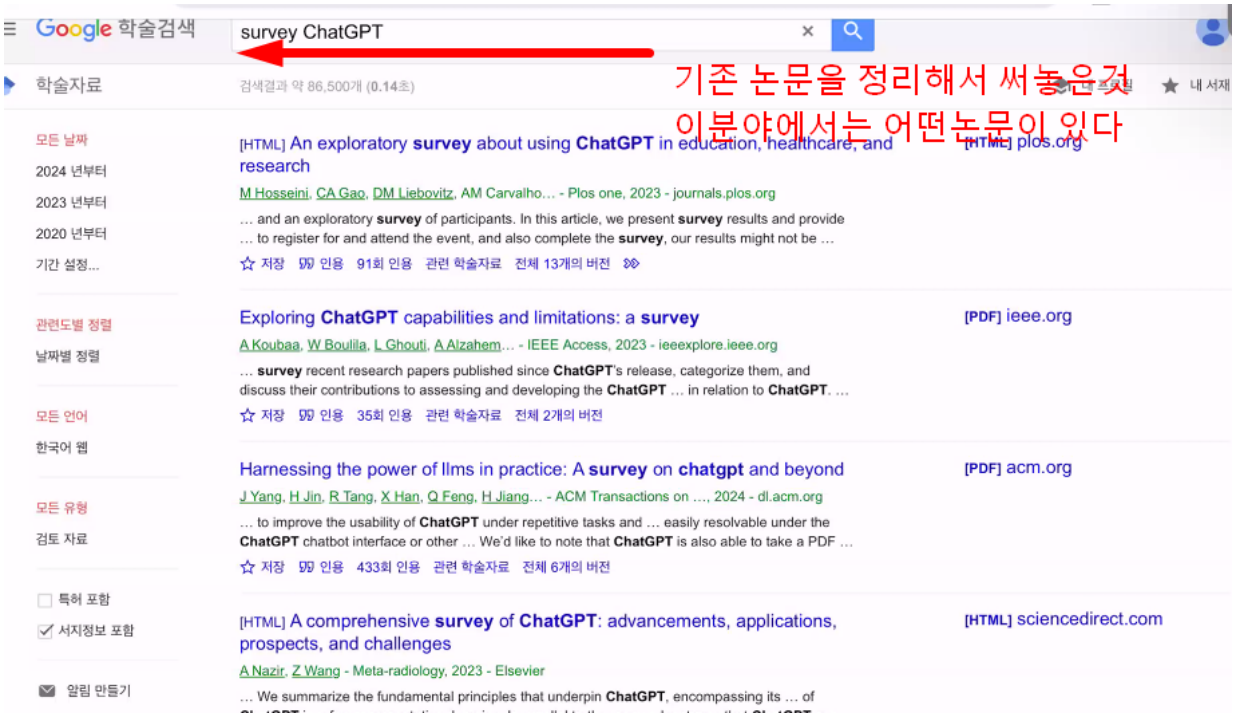
[논문리뷰]DIP: Dead code Insertion based Black-box Attack for Programming Language Model

논문-성균관대(이지형교수님) · 2024. 9. 6. 15:52

출처 : DIP: Dead code Insertion based Black-box Attack for Programming Language Model
ACL 2023
CheolWon Na, YunSeok Choi, Jee-Hyong Lee† Sungkyunkwan University
{ncw0034, ys.choi, john} @ skku.edu
CheolWon Na ncw0034@skku.edu
Information & Intelligence Lab @ SKKU

자연어학회: ACL, EMNLP, NAACL
딥러닝, 인공지능: ICML, ICLR, NIPS
이미지: CVPR



DIP: Dead code Insertion based Black-box Attack for Programming Language Model

Programming Language Model 에 대해 Dead code Insertion (데드코드삽입)방법으로 Black-box Attack(모델에 대해 정보를 알수없을때)

-> 모델을 속이려하는방법,

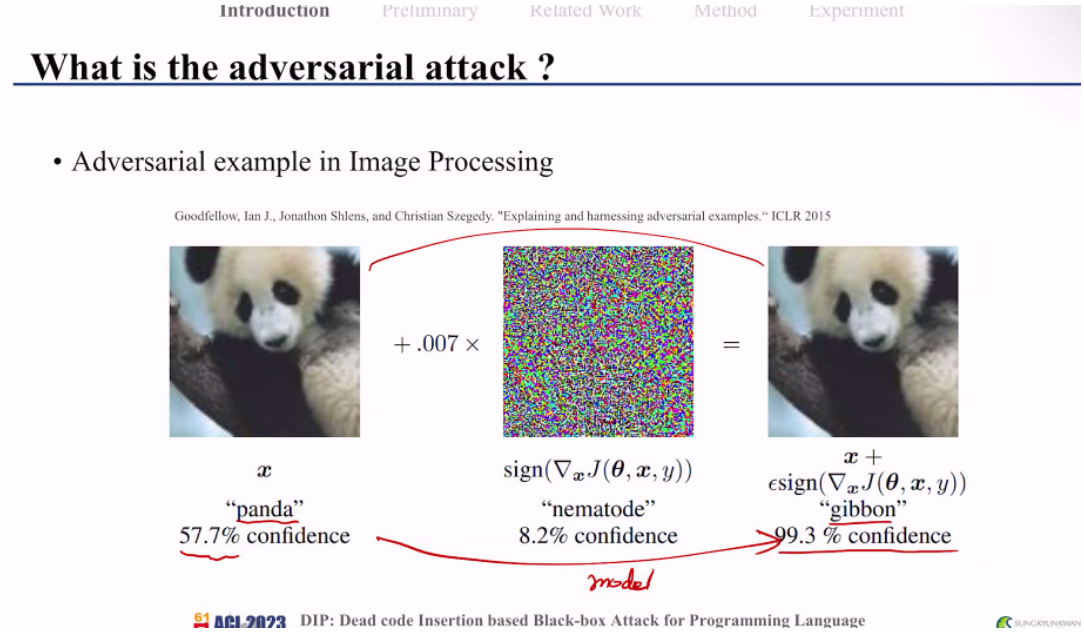
-> 은행어플에서 신분증을 올릴때 AI로 해서 검열을 통과시킬수 있음.

-> 유투브 썸네일 폭력적일때 차단일때 , 사람이 볼때 폭력적일때 검열을 통과하겠다.

-> 해킹방법

What is the adversarial attack ? 적대적 공격

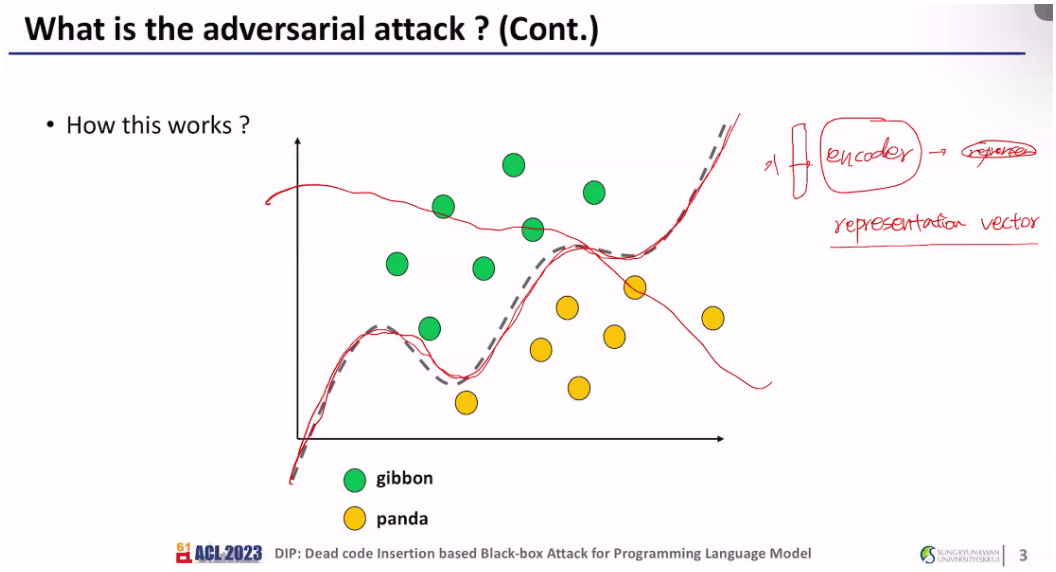
원래이미지에 사람이 보기엔 괜찮은데 모델이 보기엔 혼란스럽게 하는것



모델은 결정경계를 학습한다.

데이터를 encoder를 통해서 representation vector화를 테스트해봄

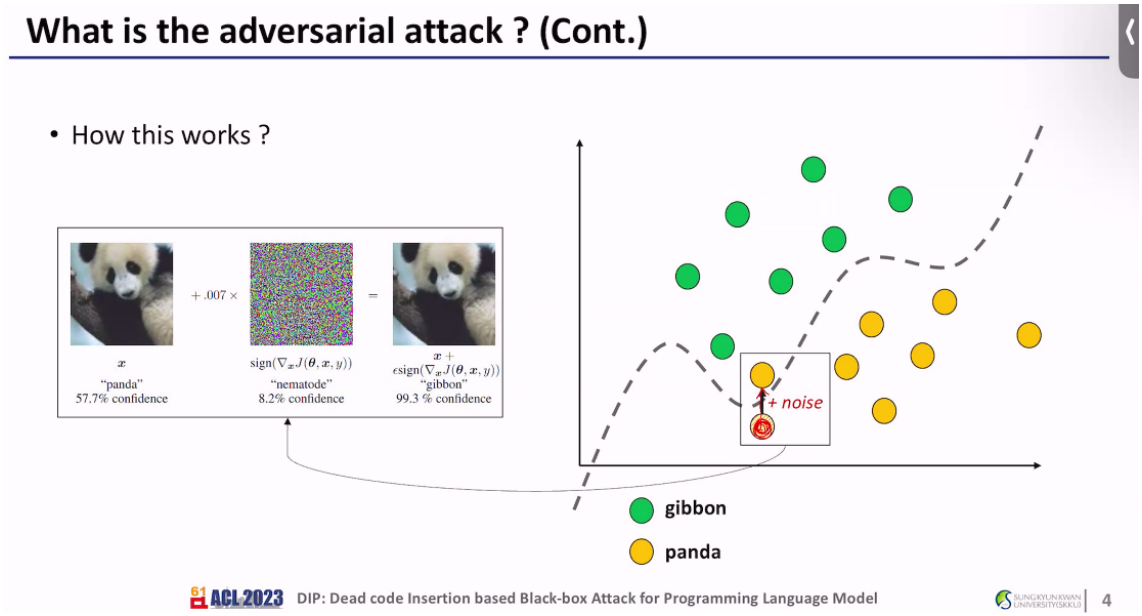
이미지, 자연어, Programming language



아래그림)

1) 이미지 모델(비전모델) attack

노이즈를 추가하면 경계를 넘어가게하는 방법을 찾는다.



아래그림)

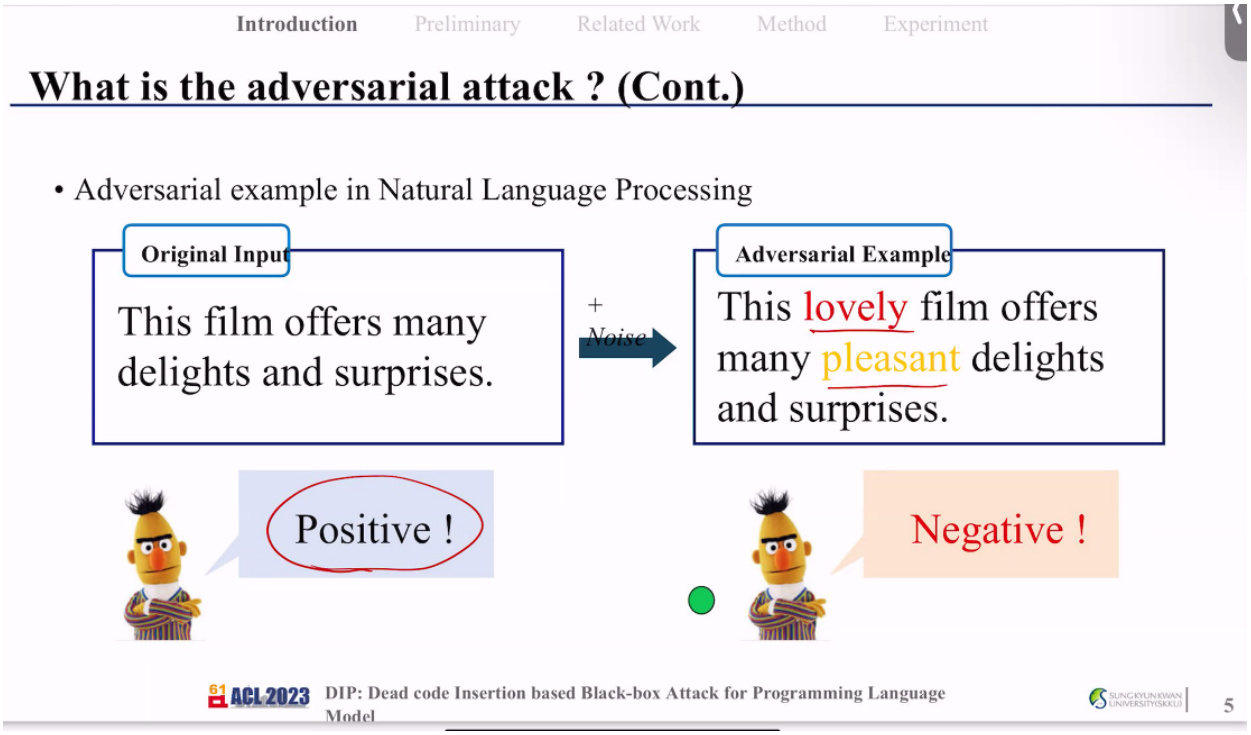
2) 자연어 모델 attack

자연어 데이터를 수정해서 자연어 모델을 속일수도있음.

adversarial attack (적대적공격)

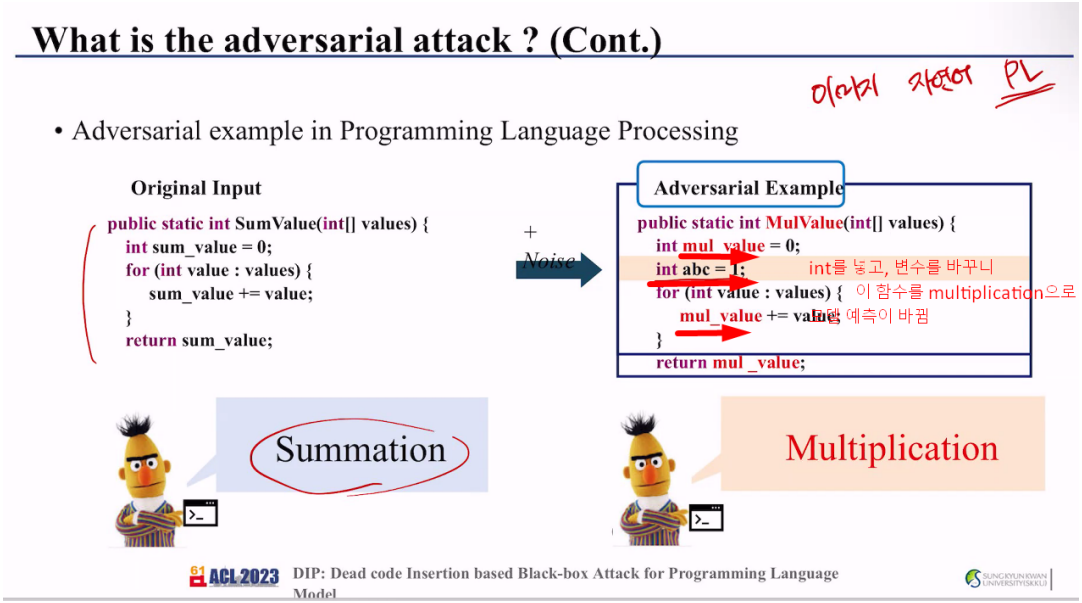
Sadv = > sentence 에 대한 적대적 공격을 수행해서 변형한 대이터

Sadv = adversarial example = > 이미지지나 자연어나 적대적 공격을 수행해서 변형한 대이터



3) programming representation

int를 넣고, 변수를 바꾸니 이 함수를 multiplication으로 모델 예측이 바뀜



아래그림)

목표

compile이 다 되야함 = minimizing perturbation

Sadv 가 Sentence랑 비슷해야하고

C (Sadv) 는 Sentence를 adversarial example를 Classification 한것이 y값(실제값)과 같지 않게 하는게 목적

compile 작성한 코드를 이진코드로 바꿔주는작업

기존의 자연어 공격방법을 그대로 가꾸올수 없음. compile이 되야하기 때문

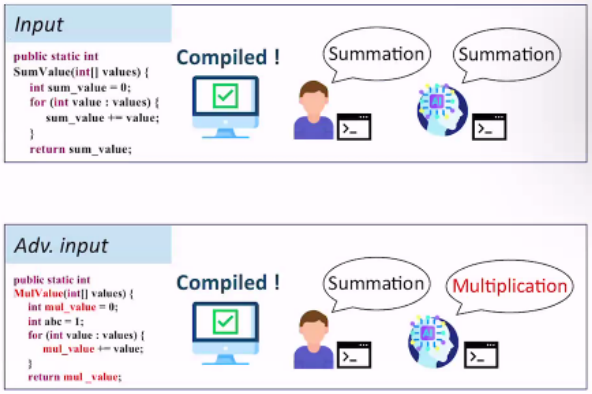
minimizing perturbation(변형) - 변형을 조금만해야한다.

= 너무 많이 변형해서 코드의 변경된것을 알아차리지 않게한것

Difference with NLP attack (Cont.)

- PLP attack
  - Given
    - S : input code, $S = \{t_1, \ldots, t_n\}$
    - C : classification model
    - y : class of S
  - Goal
    - Compilation guarantees
    - Functionality Similarity: $S_{adv} \approx S$
    - Misclassification: $C(S_{adv}) \neq y$
    - Minimizing Perturbation

Hard to directly applicate traditional NLP attack methods !



아래그림) 일반적인 adversarial attack (적대적공격)을 위한 코드변형

변수명 변형

dead code 사용

statement Permutation 변수선언한거 순서변형

Loop Exchange 루프변형

Switch - to if switch를 if로 변형

Boolean Exchange

Code transformation

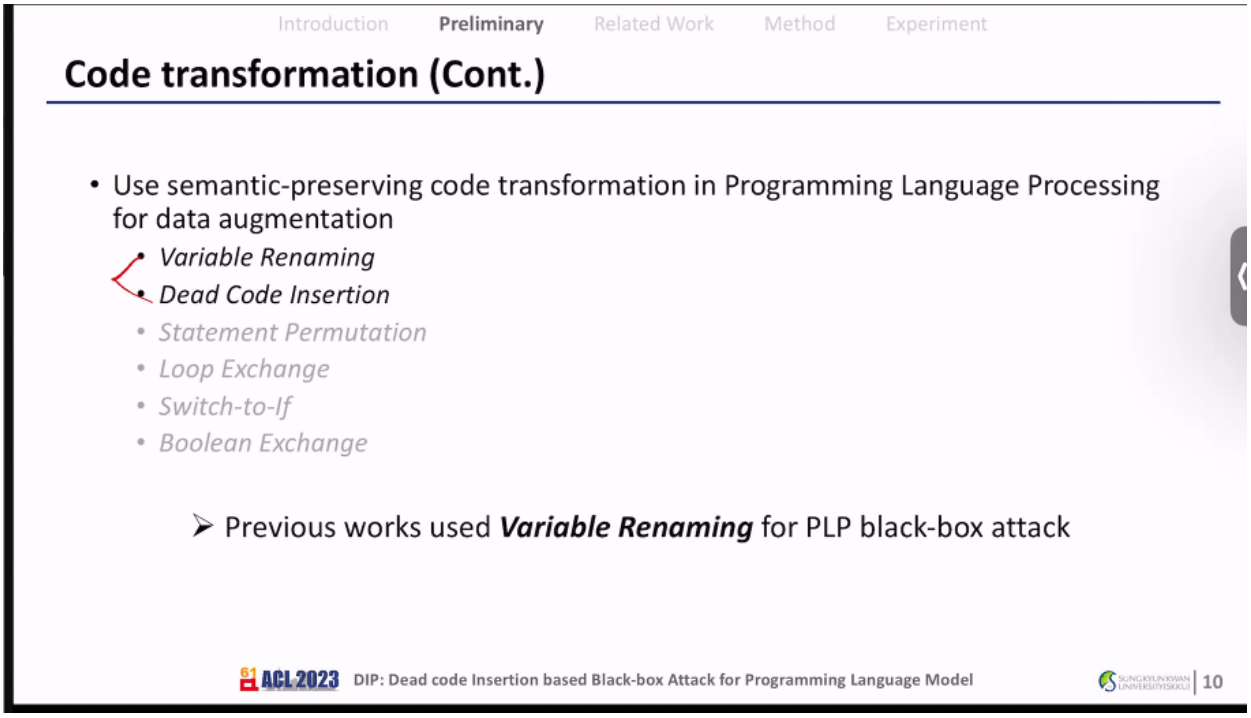
- Use semantic-preserving code transformation in Programming Language Processing for data augmentation
  - *Variable Renaming*
  - *Dead Code Insertion*
  - *Statement Permutation*
  - *Loop Exchange*
  - *Switch-to-If*
  - *Boolean Exchange*

code

아래그림) 여기 논문에서 쓸 code transformation방법

Variable Renaming

Dead Code Insertion



아래그림) 여기 논문은 밑줄을 볼때 blackbox랑, non targeted attack으로 실험함

whitebox에서는 parameter에 접근이 가능하고,

blackbox에서는 parameter에 접근이 불가능함.

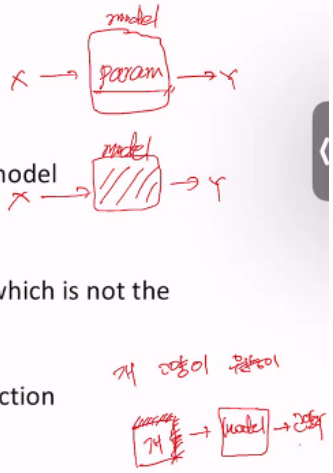
Goal (purpose of attack)

targeted attack : 특정한 output을 내도록 공격

non targeted attack : 정답이 안나오게만 하도록하는 공격

Adversarial Attack

- Setting (*When we attack the target model*)
  - *In Whit-box setting,* we can use all information from target model
  - *In Black-box setting,* we can not use any information from target model
- Goal (*purpose of attack*)
  - *Targeted attack* forces the model to output a specific prediction, which is not the correct prediction
  - *Non-targeted attack* forces the model to make any incorrect prediction



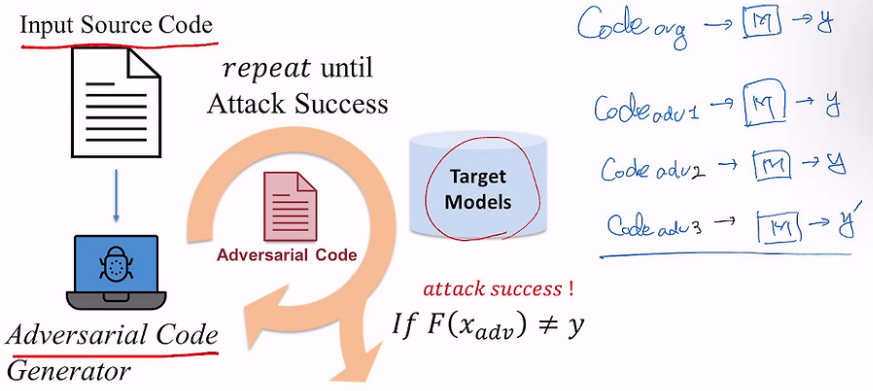
아래그림)

input source code -> adversarial code -> target model

정답이 안나오게만 하도록하는 공격으로, 정답이 안나오면 멈춤

Adversarial Attack (Cont.)

- Black-box Adversarial Attack for Programming Language Models



아래그림)

1. find vulnerable position ---공격받을 코드의 취약부분 파악후 코드삽입위치 파악
2. dis-similar code --- 공격받을 source code(원래코드) 랑 비교해서
   제일 비슷하지 않은 코드를 dataset에서 갖고옴

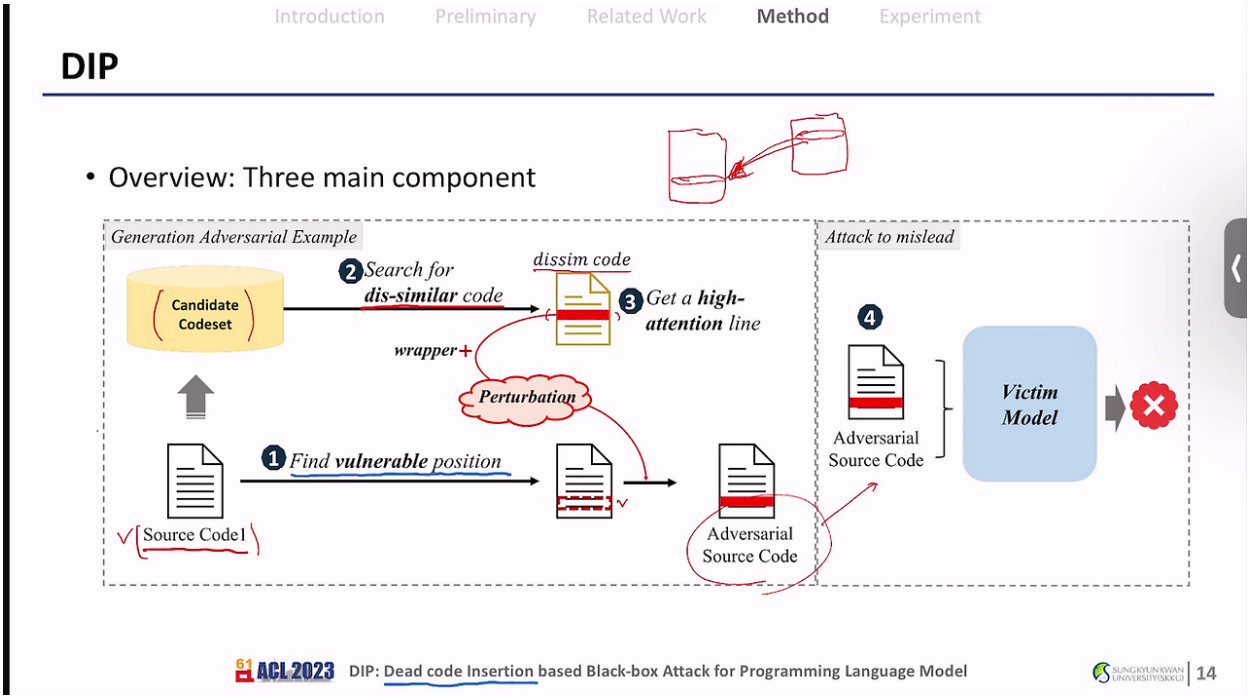
3. get a high attention line --- 제일 비슷하지않은 코드 - attention code에서 제일 높은 attention line을 고름

통째로 넣기는 perturbation이 많이 되서 high attention부분만 고른다.

3-1 source code의 취약부분에 넣음

4 바뀌어진 코드 - adversarial source code를 모델에 넣어보고 output이 동일한지 확인한다.

5. 틀릴때까지 반복 (제한선- 10번까지 등)



아래그림)

DIP : For the efficiency

Find Vulnerable Position

vocab 은

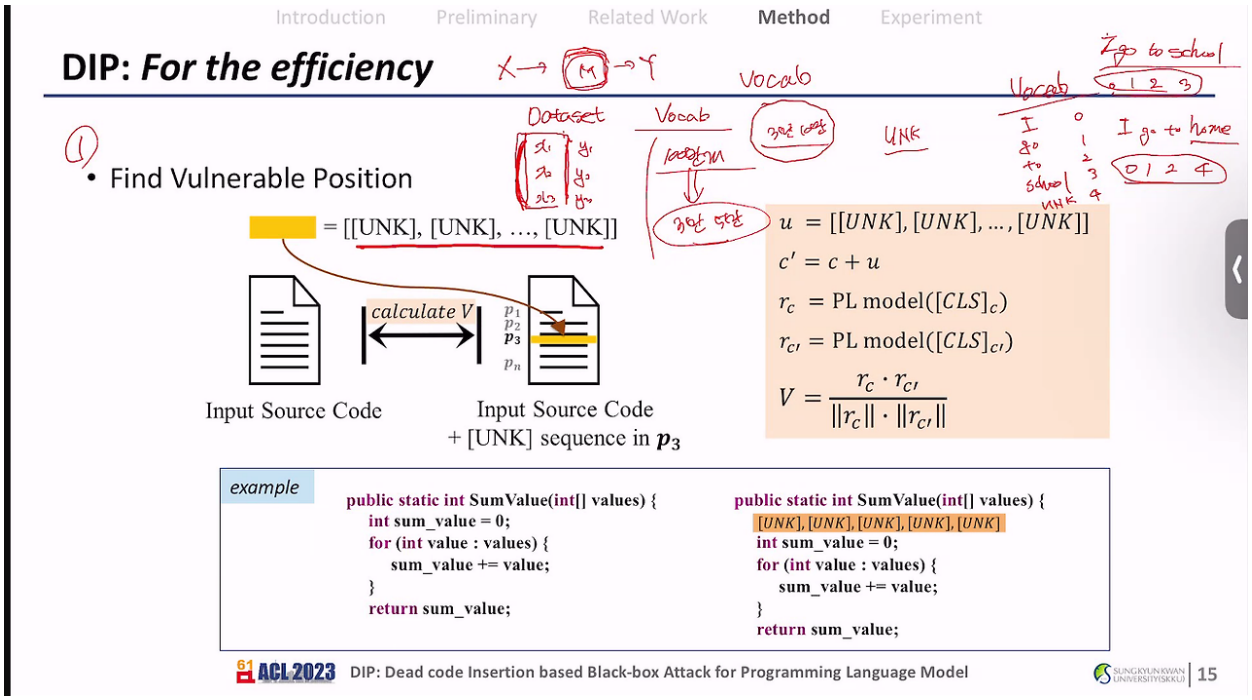
dataset의 x들에서 어떤단어는 몇번이고 .. 알려주는건데 사이즈를 줄여놓은것

훈련다끝난후 새로운 data들어올때 잘 예측하도록 학습시킴

unknown token이

새로운 token들어올때 처리하는 특수토큰

처음부터 unknown token이 vocab에 존재함



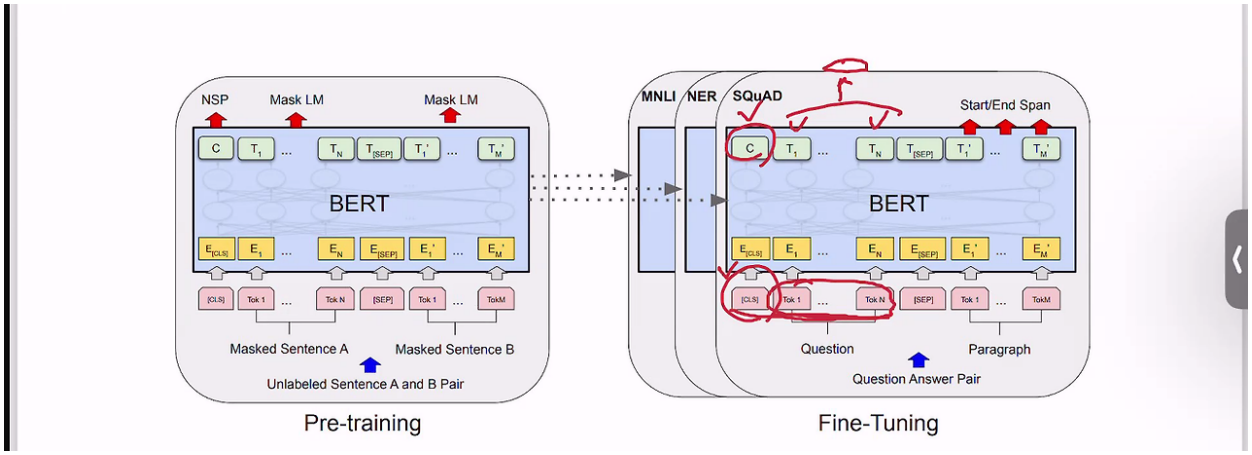
아래그림)

cls toekn 은

cls token을 넣으면 문장전체를 담는 vector를

바로 얻어낼수 있다.

출처: bert논문



아래그림)

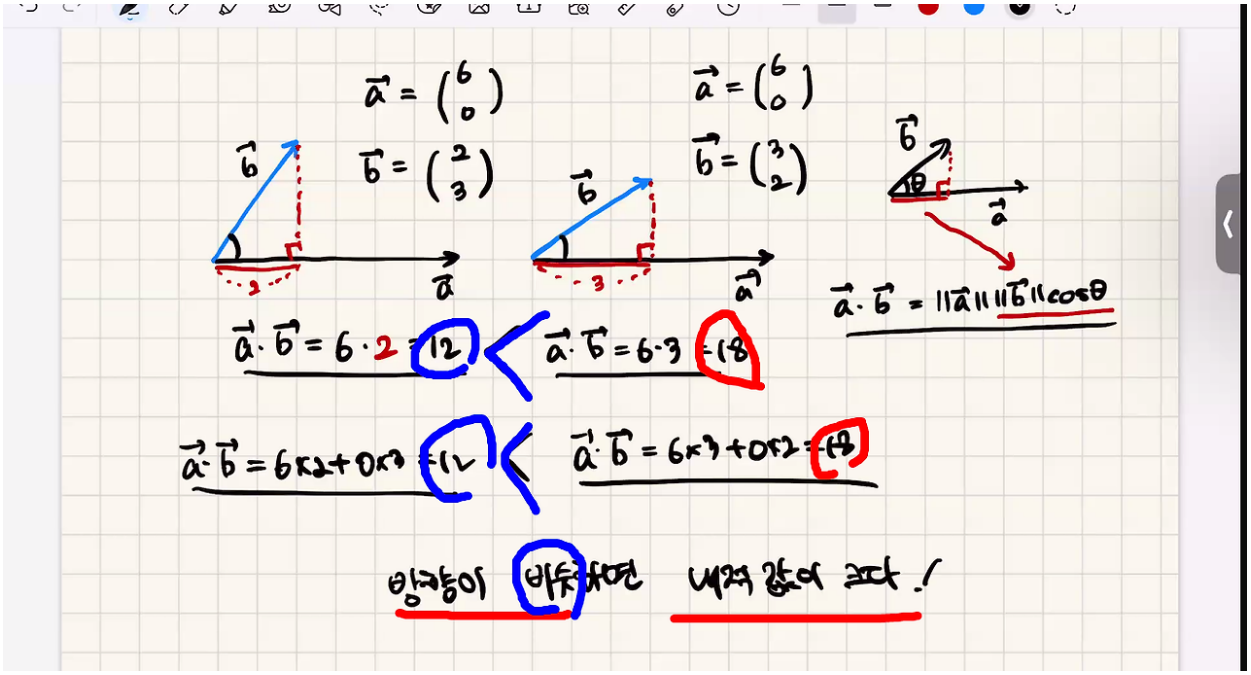
unknown token이 원래 vocab에 있는데,

[UNK] [UNK] [UNK] [UNK] 한줄을 코드에 넣어서

얼마나 변형되는지 확인

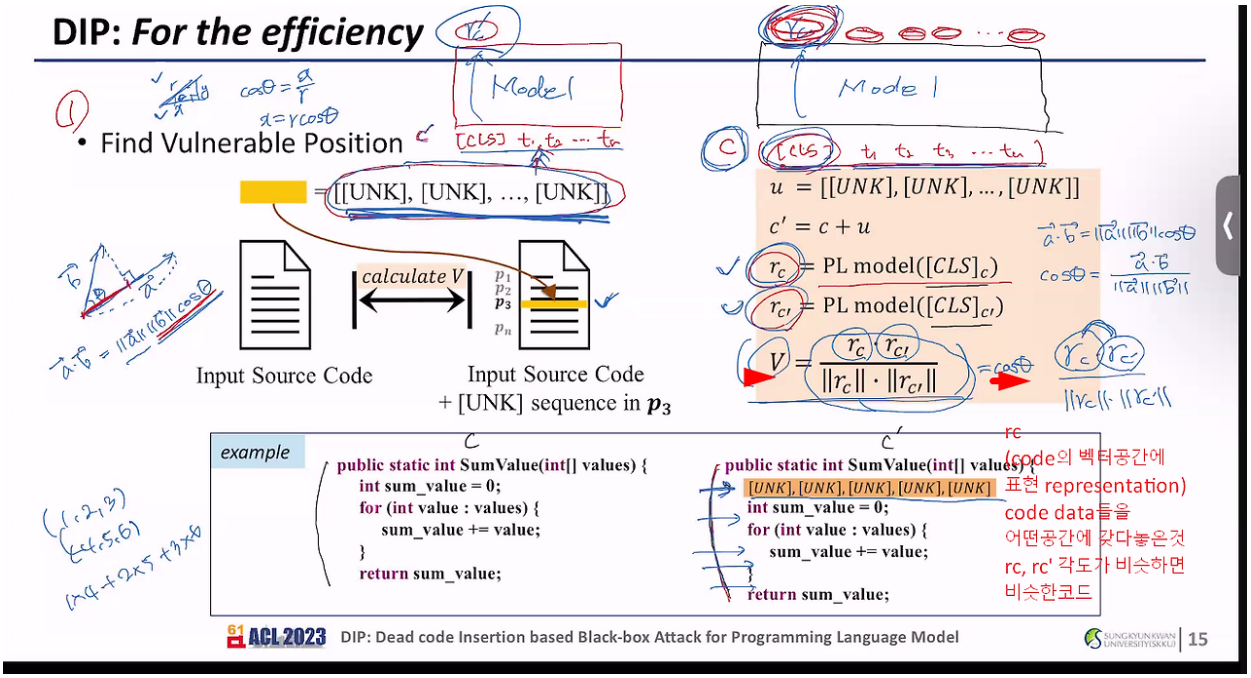
방향이 비슷하면 내적값이 크다.

cosine 크기가 크면 거리가 수직으로 내린점에서부터 멀다.



rc

(code의 벡터공간에 표현 representation) code data들을 어떤공간에 갖다놓은것 rc, rc' 각도가 비슷하면 비슷한코드



1번위치에 unknwon 코드 넣은값이 : 0.9 rc rc1 의 cosine 과

2번위치에 unknwon 코드 넣은값이 : 0.7 rc rc2 의 cosine 구해서 rc2 비교값이 더 커서 rc1보다 많이 변형됐다.

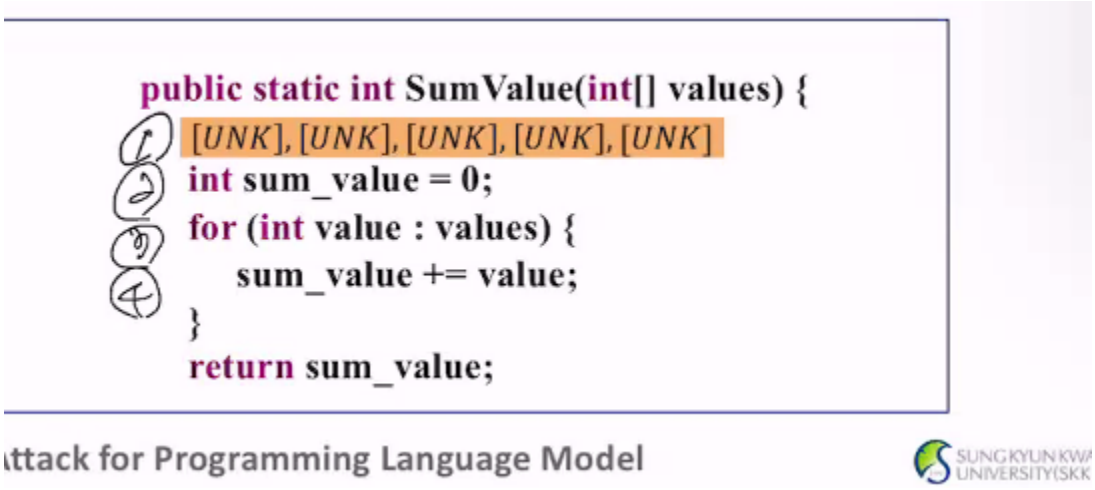
3번

4번

5번

결론 : 삽입위치 찾음---공격할곳은 제일 많이 변형된곳이 취약부분이라서 거기에 코드를 삽입할예정

-> 공격할곳 = 많이 변형된곳 = 기존코드랑 비슷하지 않은코드



Search Dis-similar Cod

code database --> sampling K --> input source code --> calculate the distance of the representation of the [CLS] token

기존 코드에서 가장 멀리있는거로 공격해보고

안되면 다음식

반복

sampling한 코드 k개중

예를들어,

C1 C2 C3 C4 C5 를 sampling한후 V (코사인각도) 로 비슷하지 않은 순으로 아래처럼 정렬

C3 C2 C4 C1 C5 -> attention score 로 한줄을 뽑은다음

d3 d2 d4 d1 d5

input source code 에 순서대로 넣어본다

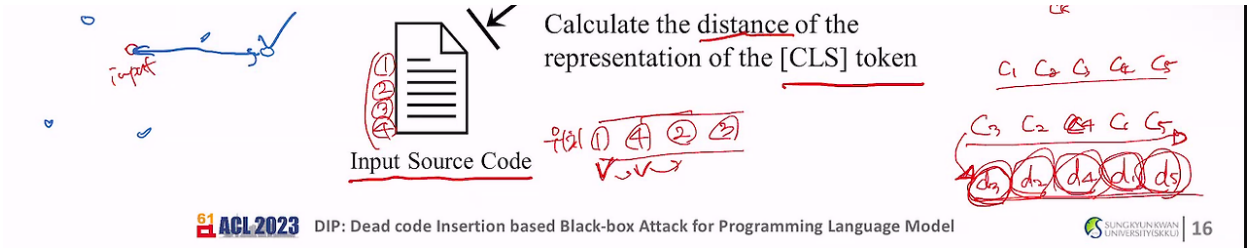
위치 1 4 2 3 --> 이 위치에 아래처럼 query = 몇번 넣어봤는지

1번에 d3넣고 안되면 d2 안되면 d4 안되면 d1 안되면 d5

2번

3번

4번



아래그림

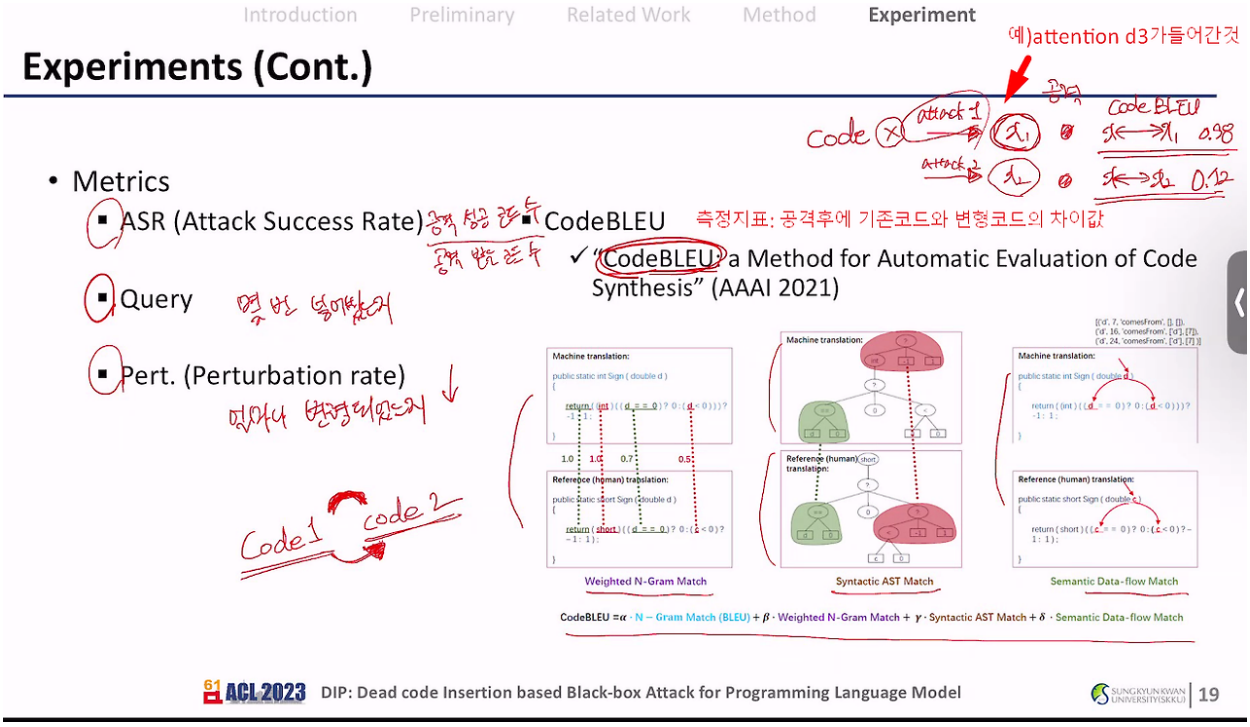
--> ASR (Attack Success Rate) 떨어지면 ACR (공격성공코드수 / 공격받는 코드수)

--> Query -> 몇번 넣어봤는지 4 x 5 1,2,3,4 (위치) d1,d2,d3,d4,d5(attention)

code BLEU

측정지표: 공격후에 기존코드와 변형코드의 차이값

차이값이 클수록 원래코드가 변형이 덜되어서 좋은 코드



아래그림)

Query, Pert, CodeBlue 각각 측정

next is undefined.

Number of Queries (Query) $Query$ is the average query number of successful attacks. Our method is a black-box approach, so queries are the only accessible way to the target model. The number of queries is one of important metrics to evaluate efficiency of attack methods. Let q_i denote the number of queries for i-th succeed attack, $Query$ is defined as follows:

$$Query = \frac{\sum q_i}{\sum f(i)} \tag{4}$$

where $i \in \{j | f(j) = 1\}$, and

$$f(j) = \begin{cases} 1, & \text{if } M(C_j^{adv}) \neq y_j \wedge M(C_j) = y_j \\ 0, & \text{otherwise} \end{cases}$$

Ratio of Perturbation (Pert) The ratio of perturbation indicates how many perturbations are injected into the original source code. A lower $Pert$ indicates that examples are generated with less perturbation. Let C_i^{adv} is an adversarial example of C_i of which the truth label is y_i, and $t(\cdot)$ is the number of tokens. $Pert$ is defined as follows:

공격되서 변형된코드

$$Pert = \frac{\sum t(C_i^{adv}) - t(C_i^{adv} \cap C_i)}{\sum t(C_i)} \tag{5}$$

where i are defined same as in Eq. 4.

CodeBLEU Ren et al. (2020) proposed $CodeBLEU$ to measure generated code by machine learning models. $CodeBLEU$ considers functional and structural information of code such as AST match and Data-flow match. $CodeBLEU$ is a more efficient metric than $BLEU$ to measure the consistency of generated code. If $CodeBLEU$ is close to 1, the code preserves the semantic meaning of the original code.

$CodeBLEU$ is defined as follows:

$$CodeBLEU = \alpha \cdot BLEU + \beta \cdot BLEU_{weight} + \gamma \cdot Match_{ast} + \delta \cdot Match_{df} \tag{6}$$

아래그림

pert비율이 낮을수록 변형덜됨

Number of Queries (Query) $Query$ is the average query number of successful attacks. Our method is a black-box approach, so queries are the only accessible way to the target model. The number of queries is one of important metrics to evaluate efficiency of attack methods. Let q_i denote the number of queries for i-th succeed attack, $Query$ is defined as follows:

$$Query = \frac{\sum q_i}{\sum f(i)} \tag{4}$$

where $i \in \{j | f(j) = 1\}$, and

$$f(j) = \begin{cases} 1, & \text{if } M(C_j^{adv}) \neq y_j \wedge M(C_j) = y_j \\ 0, & \text{otherwise} \end{cases}$$

Ratio of Perturbation (Pert) The ratio of perturbation indicates how many perturbations are injected into the original source code. A lower $Pert$ indicates that examples are generated with less perturbation. Let C_i^{adv} is an adversarial example of C_i of which the truth label is y_i, and $t(\cdot)$ is the number of tokens. $Pert$ is defined as follows:

$$Pert = \frac{\sum t(C_i^{adv}) - t(C_i^{adv} \cap C_i)}{\sum t(C_i)} \tag{5}$$

where i are defined same as in Eq. 4.

CodeBLEU Ren et al. (2020) proposed $CodeBLEU$ to measure generated code by machine learning models. $CodeBLEU$ considers functional and structural information of code such as AST match and Data-flow match. $CodeBLEU$ is a more efficient metric than $BLEU$ to measure the consistency of generated code. If $CodeBLEU$ is close to 1, the code preserves the semantic meaning of the original code.

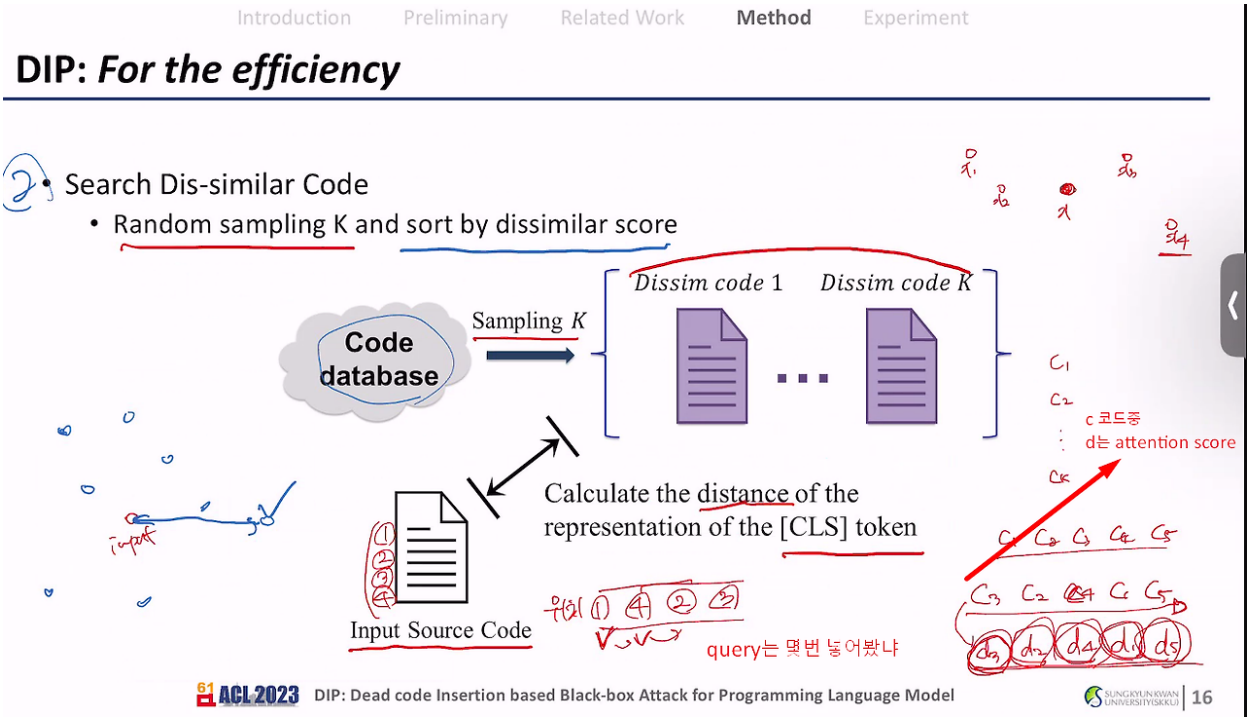
$CodeBLEU$ is defined as follows:

$$CodeBLEU = \alpha \cdot BLEU + \beta \cdot BLEU_{weight} + \gamma \cdot Match_{ast} + \delta \cdot Match_{df} \tag{6}$$

변형된코드갯수 기존코드갯수

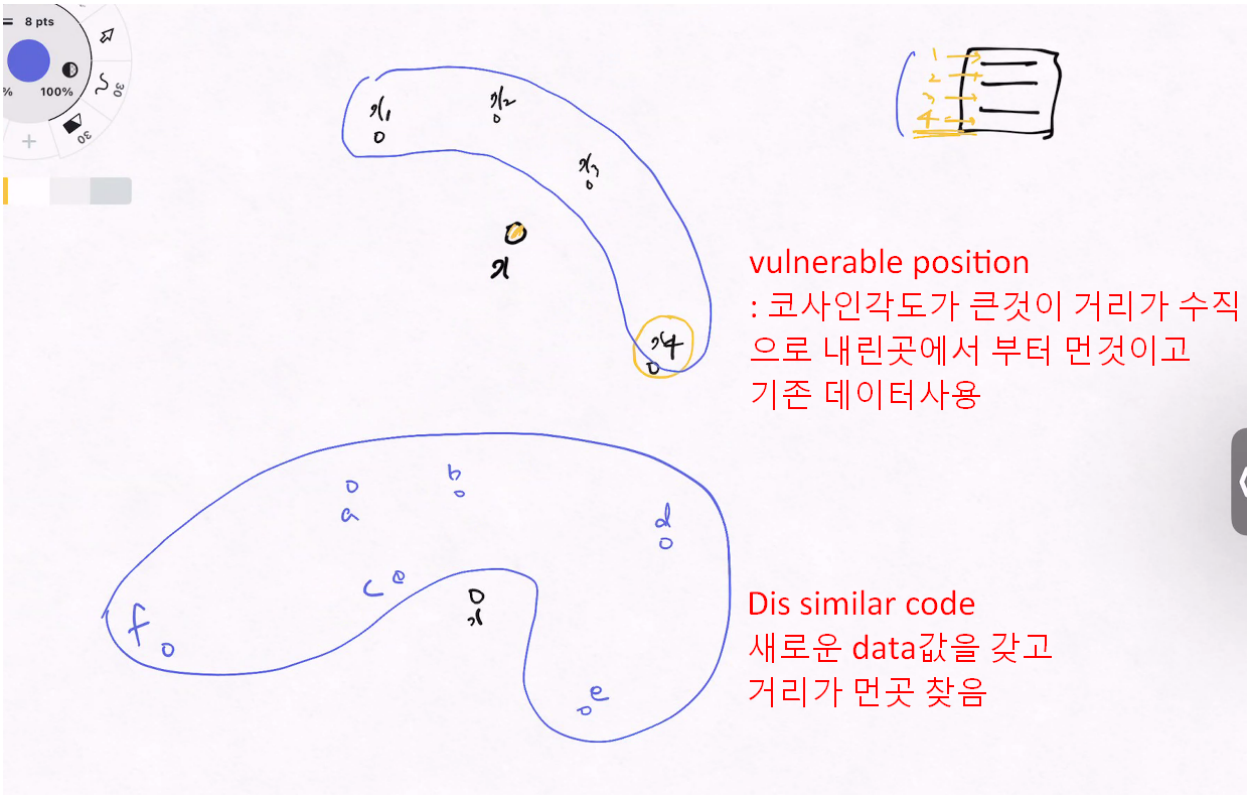
중복코드갯수

이 pert비율이 낮을수록 변형덜됨



vulnerable position vs. dissimilar code

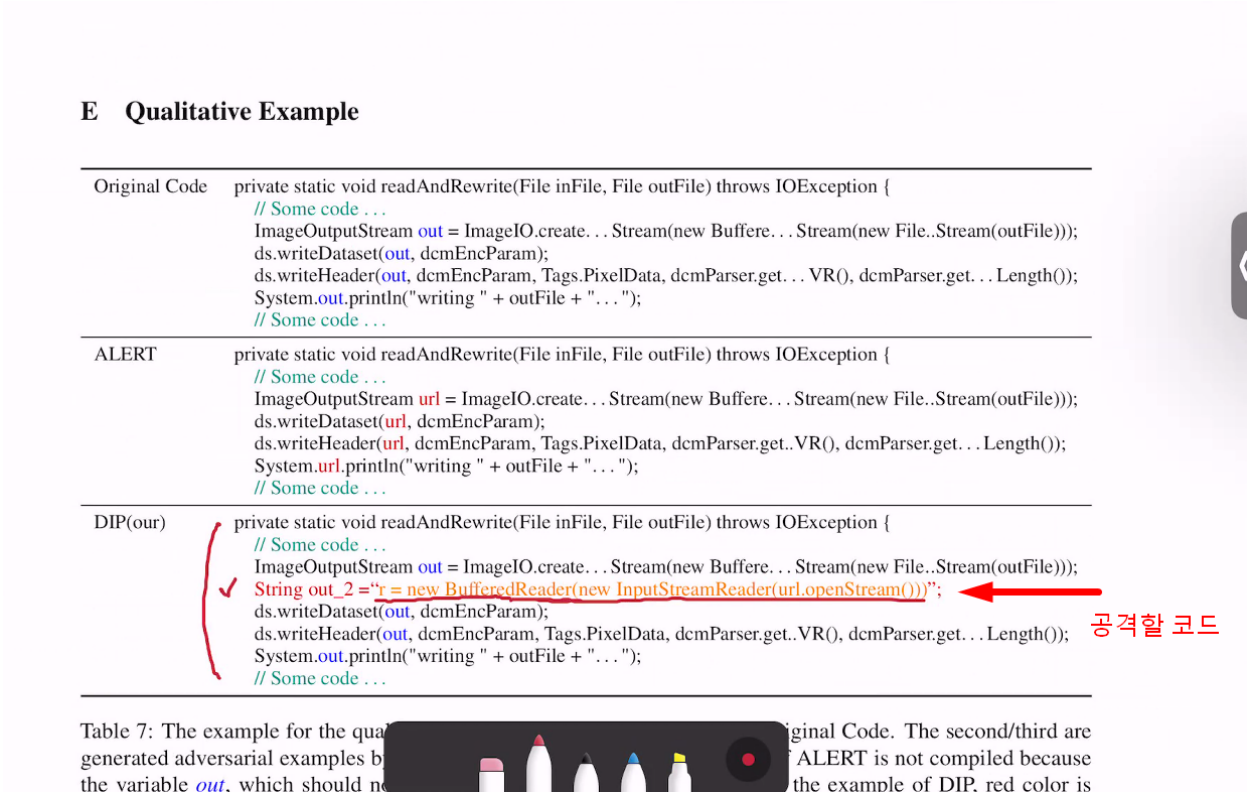
기존코드에서 삽입할거찾음 새로운 코드에서 넣을거 찾음



아래그림

새로운 코드에서 넣을거 찾은것에서 attention score로 한줄만 골라서 한줄을 기존코드에 넣는다.
똑같은 코드가 있을수 있어서 공격할 코드를 wrapper로 감싸준다.

아래그림) wrapper 예시 (다른논문)



E Qualitative Example

Original Code	private static void readAndRewrite(File inFile, File outFile) throws IOException {
	// Some code . . .
	ImageOutputStream out = ImageIO.create. . . Stream(new Buffere. . . Stream(new File..Stream(outFile)));
	ds.writeDataset(out, dcmEncParam);
	ds.writeHeader(out, dcmEncParam, Tags.PixelData, dcmParser.get. . . VR(), dcmParser.get. . . Length());
	System.out.println("writing " + outFile + ". . . ");
	// Some code . . .

ALERT	private static void readAndRewrite(File inFile, File outFile) throws IOException {
	// Some code . . .
	ImageOutputStream url = ImageIO.create. . . Stream(new Buffere. . . Stream(new File..Stream(outFile)));
	ds.writeDataset(url, dcmEncParam);
	ds.writeHeader(url, dcmEncParam, Tags.PixelData, dcmParser.get..VR(), dcmParser.get. . . Length());
	System.url.println("writing " + outFile + ". . . ");
	// Some code . . .

DIP(our)	private static void readAndRewrite(File inFile, File outFile) throws IOException {
	// Some code . . .
	ImageOutputStream out = ImageIO.create. . . Stream(new Buffere. . . Stream(new File..Stream(outFile)));
	String out_2 ="r = new BufferedReader(new InputStreamReader(url.openStream()))";
	ds.writeDataset(out, dcmEncParam);
	ds.writeHeader(out, dcmEncParam, Tags.PixelData, dcmParser.get..VR(), dcmParser.get. . . Length());
	System.out.println("writing " + outFile + ". . . ");
	// Some code . . .

공격할 코드

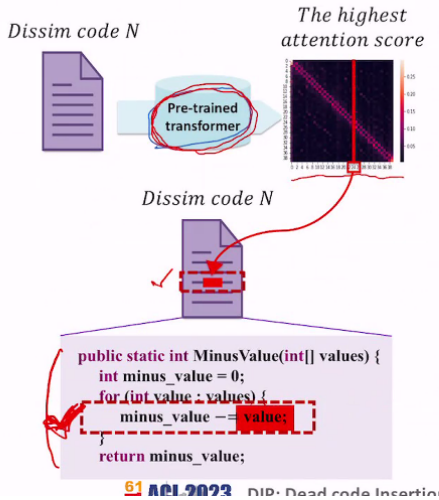
Table 7: The example for the qua ginal Code. The second/third are generated adversarial examples b ALERT is not compiled because the variable *out*, which should n the example of DIP, red color is

아래그림
공격해서 안됬으면 다음코드로 반복

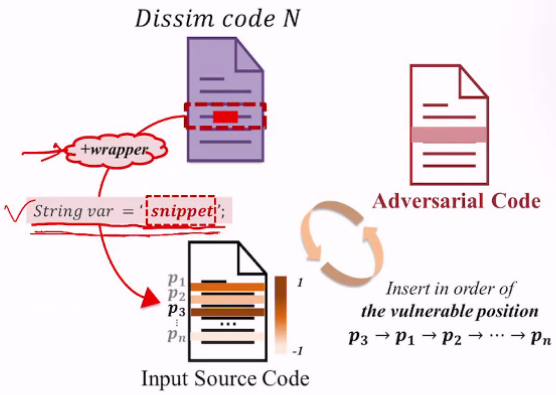
IP: For the effectiveness

a = "minus_value -= value;"

- High-attention Snippet Extraction



- Adversarial Code Generation



아래그림)

authorship attribution task - 66명 author 중 1명 맞춤

widely used benchmark in the clone detection task. This dataset contains true clone pairs and false clone pairs from Java projects.

- *Defect Detection* aims to find whether a given source code is insecure or not. We use a dataset served by (Zhou et al., 2019). This dataset is extracted from large-scale open-source C projects. This dataset includes 27,318 functions.
- *Authorship Attribution* task is to identify the author of a given source code. We use the Google Code Jam (GCJ) dataset, which is collected by Alsulami et al. (2017). This dataset contains 660 python files (66 authors and 10 files per author).

66사람중 한사람 맞춤
코드를 보고 누가작성한 코드인지
맞추는 코드

Target Models We analyze the vulnerability of three popular and powerful pre-trained PL models: *CodeBERT*, *GraphCodeBERT*, and *CodeT5*. *CodeBERT* (Feng et al., 2020)'s objectives are masked language modeling on NL-PL pairs and replaced

difference in CLS vectors. We set K=30. If K is small, the attack success rate will be low, and if it is large, the attack will be inefficient. We choose a number small but large enough to include various source code. For comparison with the baselines, we use the same maximal input length to 512.

4.2 Metrics

We use four metrics to evaluate our method. To measure the *efficiency* of the generated adversarial code, we use ASR and $Query$. We also use $Pert$ and $CodeBLEU$ to measure *quality* of the generated examples. We define the following metrics.

Attack Success Rate (ASR) ASR is the success ratio of attacks. The higher ASR, the better performance of an attack method. Let C^{adv} denote a generated adversarial example from the original input C, M is the target model, and y is the true label. Then ASR isndefined as follows:

$$ASR = \frac{|\{C|M(C^{adv}) \neq y \wedge M(C) = y\}|}{|\{C\}|} \quad (3)$$

아래그림)

실험 세팅

Experiments

- Pre-trained PL models
  - CodeBERT
    - ✓ CodeBERT: A Pre-Trained Model for Programming and Natural Languages (EMNLP Findings 2020)
  - GraphCodeBERT
    - ✓ GraphCodeBERT: Pre-training Code Representations with Data Flow (ICLR 2021)
  - CodeT5
    - ✓ CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation (EMNLP 2021)
- Dataset (3 tasks)
  - Code clone detection (Java) binary cls
  - Defect detection (C/C++) binary cls ✓
  - Authorship attribution (Python) multi cls

➢ 3 PL Models x 3 Tasks = 9 Target models

Task	Model	Acc.
Clone Detection (# of Classes: 2)	CodeBERT	97.3%
	GraphCodeBERT	97.8%
	CodeT5	97.1%
Defect Detection (# of Classes: 2)	CodeBERT	63.3%
	GraphCodeBERT	64.2%
	CodeT5	63.7%
Authorship Attribution (# of Classes: 66)	CodeBERT	82.6%
	GraphCodeBERT	81.1%
	CodeT5	85.6%

아래그림

ASR : 공격성공코드수가 높으면 좋음

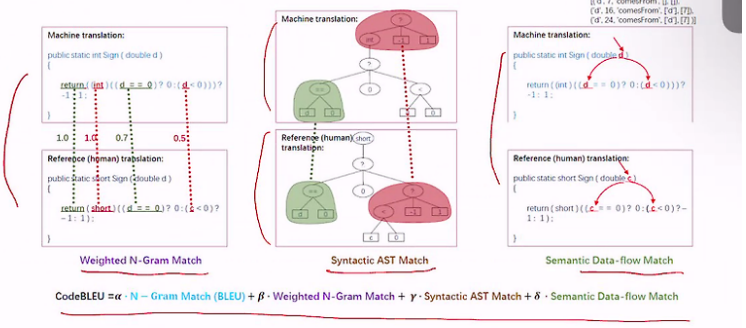
Query: 몇번넣어봤는지가 적으면 좋음

Pert: 변형이 덜되면 좋음

CodeBlue: 기존코드와 변형코드가 비슷할수록 좋음 = 높으면 높을수록 좋음

Experiments (Cont.)

- Metrics
  - ASR (Attack Success Rate)
  - Query
  - Pert. (Perturbation rate)
  - CodeBLEU
    - ✓ "CodeBLEU: a Method for Automatic Evaluation of Code Synthesis" (AAAI 2021)



아래그림)

DIP(ours)가 이논문이 실행한것

9개의 상황(task, 공격받을 model) 에 대해 평균적으로 이논문 DIP(ours) 방법이 성능이 제일 좋았다.

Experiments (Cont.)

- Comparison with baseline
  - Set all of the hyper-parameters (α, β, γ, and δ) in CodeBLEU to 0.25, respectively. The best performance is in **boldface**, and the next is underlined.

Task (Language)	Victim Model	Attack Method	Attack efficiency ASR ↑	Attack efficiency Query ↓	Attack quality Perl ↓	Attack quality CodeBLEU ↑
Clone Detection (Java)	CodeBERT	MHM	20.2	667.7	0.32	0.56
		ALERT	28.6	529.4	**0.13**	0.73
		DIP (ours)	**46.7**	**19.9**	0.13	**0.92**
	GraphCodeBERT	MHM	4.2	1025.9	0.36	0.32
		ALERT	9.2	448.6	0.13	0.72
		DIP (ours)	**36.6**	**78.2**	**0.14**	**0.85**
	CodeT5	MHM	4.6	104.5	0.26	0.42
		ALERT	22.0	762.2	0.14	0.73
		DIP (ours)	**31.8**	**38.2**	**0.11**	**0.93**
Defect Detection (C/C++)	CodeBERT	MHM	27.4	451.9	0.33	0.32
		ALERT	31.4	277.6	**0.11**	0.76
		DIP*	**44.6**	**47.6**	0.19	**0.91**
	GraphCodeBERT	MHM	41.3	316.7	0.33	0.31
		ALERT	46.7	263.6	**0.10**	0.76
		DIP (ours)	**49.7**	**71.0**	0.13	**0.79**
	CodeT5	MHM	49.3	333.5	0.10	0.78
		ALERT	46.9	187.4	**0.08**	0.80
		DIP (ours)	**49.7**	**61.0**	0.16	**0.92**
Authorship Attribution (Python)	CodeBERT	MHM	15.9	444.0	0.13	0.78
		ALERT	29.6	545.4	0.13	0.79
		DIP (ours)	**31.1**	**300.15**	**0.09**	**0.85**
	GraphCodeBERT	MHM	26.5	774.9	0.30	0.49
		ALERT	50.8	573.2	0.15	0.75
		DIP (ours)	**61.4**	**292.6**	**0.08**	**0.81**
	CodeT5	MHM	36.4	684.6	0.16	0.78
		ALERT	41.7	373.4	0.10	0.83
		DIP (ours)	**43.9**	**47.2**	**0.07**	**0.92**
All	Average	MHM	25.1	537.7	0.25	0.53
		ALERT	34.1	440.1	**0.12**	0.76
		DIP (ours)	**43.9**	**106.2**	**0.12**	**0.88**

아래그림

Ablation study - 요소를 하나씩 빼서 실험함

w/o Dissim : C1, C2, C3, C4, C5 --> sorting없이 랜덤하게 가져와서 공격함

w/o Position: 어느위치에 넣을지 랜덤하게 넣어봄

w/o Att-line: attention score없이 랜덤하게 C1이런 코드에서 갖고옴

--> 결론: 덜좋아짐

Experiments (Cont.)

- Ablation study

Method	ASR	Pert	Query
DIP	**46.7**	**0.13**	**19.9**
w/o Dissim	45.2±0.9	0.14±0.01	26.0±5.1
w/o Position	46.7±0.0	0.14±0.00	29.6±5.0
w/o Att-line	47.0±0.5	0.12±0.00	110.3±6.1

아래그림

CodeBERT, G.CodeBERT, CodeT5를 공격해서 얻은 C^adv를 다른 모델에도 넣어본다.

- Transferability

Adversarial Examples	Code-BERT	G. Code-BERT	Code-T5
against CodeBERT	-	31.3	22.0
against G.CodeBERT	12.6	-	17.8
against CodeT5	24.8	29.3	-

61 ACL 2023 DIP: Dead code Insertion based Blac

아래그림

Adversarial train

원래는 C1을 넣은 예측값과 정답인 y1를 비교하는건데

이제는 C1^adv를 넣은 예측값과 정답인 y1을 비교해서 학습을 추가적으로 함

- Adversarial train

Method	*MHM*	*ALERT*	*DIP*	*All*
MHM	2.5	1.5	9.2	1.0
ALERT	5.5	4.5	**28.1**	4.5
DIP (ours)	**32.0**	**41.5**	15.0	**8.0**

- Various snippet extractions

Snippet	ASR%	Pert	Query

아래그림

sample code k를 늘려봤을때 30정도까지만 효과가 있고 이상부터는 효과가 없다.

Experiments (Cont.)

- Different parameter *K* of dissimilar code

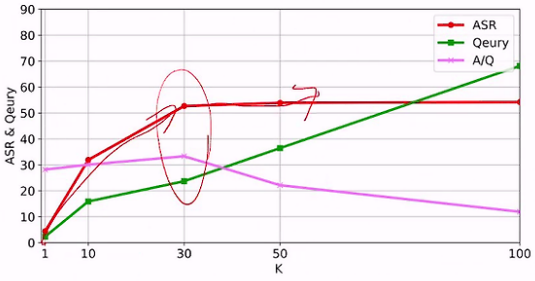


Figure 3: Using different parameter K from 1 to 100 in our attack method. The ASR and A/Q increase until K to 30

아래그림

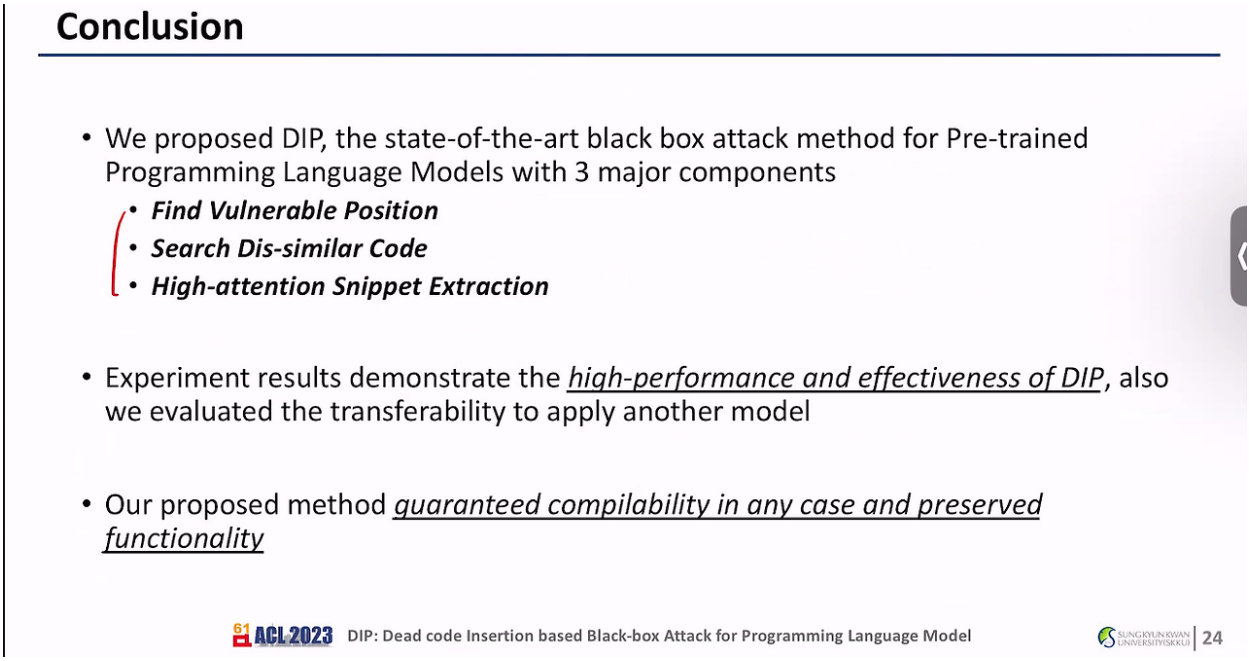
결론

find vulnerable position

search dis-similar code

high-attention snippet extraction

을 해서 효율적이고 높은 성능의 adverserial attack방법을 찾아냄.



2024. 9. 6. 17:48

익명

비밀댓글입니다.

2024. 9. 6. 18:27

익명

비밀댓글입니다.

2024. 9. 6. 18:47

익명

비밀댓글입니다.

2024. 9. 6. 19:05

이름

비밀번호

내용을 입력하세요.

등록