

LLM Mutation 논문 리뷰 및 Robot LLM Mutation

2025. 01. 14 (Tue)

Department of Computer And Information Security, MinGeon Park



Table of Contents

1. LLM Mutation Papers

- Large Language Model are Zero-shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models (**TitanFuzz**)
- Fuzz4All : Universal Fuzzing with Large Language Models (**Fuzz4All**)
- Large Language Model guided Protocol Fuzzing (**ChatAFL**)
- ProphetFuzz: Fully Automated Prediction and Fuzzing of High-Risk Option Combinations with Only Documentation via Large Language Model (**ProphetFuzz**)

2. Summary

3. Next Step

1. LLM Papers : TitanFuzz

- Large Language Model are Zero-shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models (ISSTA 2023 | Citation: 228 | UIUC | Yinlin Deng 외 5명)
- Overview
 - 대규모 언어 모델(LLM)이 Zero-shot Task(사전에 학습된 데이터에 포함되지 않은 새로운 클래스나 작업에 대한 수행)에 특화된 퍼저로 활용할 수 있음을 제시한 논문으로 기존의 퍼징 기법의 한계(L1~L3)를 극복하고 다양한 도메인에 적용 가능한 방법론 TitanFuzz를 제안
 - 입력 제약이 복잡한 딥러닝 라이브러리 (PyTorch, TensorFlow)를 TitanFuzz에 의해 효과적으로 퍼징하여 다양한 Zero-shot Task에 대한 적용 가능성을 제시
- Limitation of Traditional Fuzzers (e.g. FreeFuzz, LEMON ...)
 - L1. 제한된 API 시퀀스 탐색 공간
 - API-Level Fuzzer : 단일 API 호출에 의존하여, 체인 연결된 API 시퀀스에서 발생하는 버그 탐지 불가
 - Model-Level Fuzzer : Layer 기반 연산(Conv2d ...)에 초점을 맞춰 특정한 시퀀스 API 테스트 불가
 - L2. 제안된 코드 다양성
 - 사전 정의된 입력 생성 규칙에 의존하여 코드 구조와 입력의 다양성 부족
 - L3. 도메인 확장성의 부족
 - 기존 기술은 특정 도메인이나 라이브러리에만 맞춰 설계되어 새로운 시스템에 적용이 어려움

1. LLM Papers : TitanFuzz

• 체인 연결의 예

```
import torch

# Step 1: Random tensor 생성
x = torch.randn(3, 3, dtype=torch.float32)

# Step 2: log 연산 (NaN 값 생성 가능)
intermediate = torch.log(x * 2 - 1)

# Step 3: matrix_exp 연산
output_cpu = torch.matrix_exp(intermediate) # CPU: NaN 포함
x = x.cuda()
output_gpu = torch.matrix_exp(intermediate) # GPU: NaN 포함하지 않음
```

1. LLM Papers : TitanFuzz

Task 1: Import `TensorFlow` 2.10.0 target library
Task 2: Generate input data
Task 3: Call the API `tf.nn.conv2d(input,filters,strides,padding,data_format='NHWC',dilations=None,name=None)`
step-by-step Prompts

Main Idea

- Zero-shot Fuzzing 개념을 활용한 효과적인 Test Case 생성
 - LLM을 활용하여 사전 정의(학습) 없이 유효하고 다양한 test case 생성 (**Fully Automatic**)
 - 특정 API의 문법, 의미론, 복잡한 제약 조건을 이해한 생성 (**Context Aware**)
- 생성과 변형을 적절히 조합한 Approach
 - 생성 기반 퍼징과 변형 기반 퍼징을 LLM을 이용하여 적절히 결합한 접근
 - Codex 모델 (Code 생성에 특화된 모델)을 활용해서 초기 시드 생성
 - 시드 생성에 이용하는 이유는 Codex 모델은 GPT 기반의 decoder 특화 모델로 일관된 답변보다는 다양성이 강한 생성에 적합
 - Infilling LLM과 Mutation Operators를 이용해서 Code를 변환 테스트, EVFuzz 기법 활용
 - 이 과정에서 Evaluation Fuzz 기법을 활용해서 Fitness 함수를 구성하여 우선 순위 부여(중복 호출 수, 깊이 등 고려)

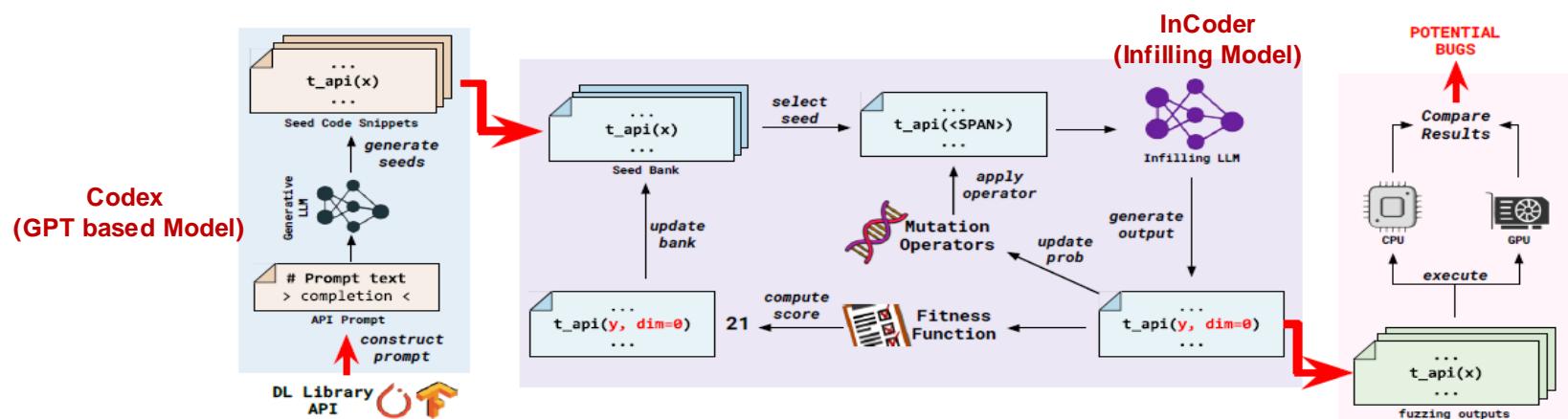


Figure 4: Overview of TITANFUZZ

1. LLM Papers : TitanFuzz

• Mutation Operators

- Incoder(양방향 모델) 모델은 masking 된 코드를 채우는 방식으로 학습 되어 있어 다양한 Test case 생성을 위해 Masking 기법 차용
 - Argument: 특정 호출 인수 마스킹
 - Method: 메서드 이름 마스킹
 - Prefix / Suffix : 호출 앞 뒤에 새로운 코드 추가
 - Keyword Insertion : 호출에 새로운 키워드 추가

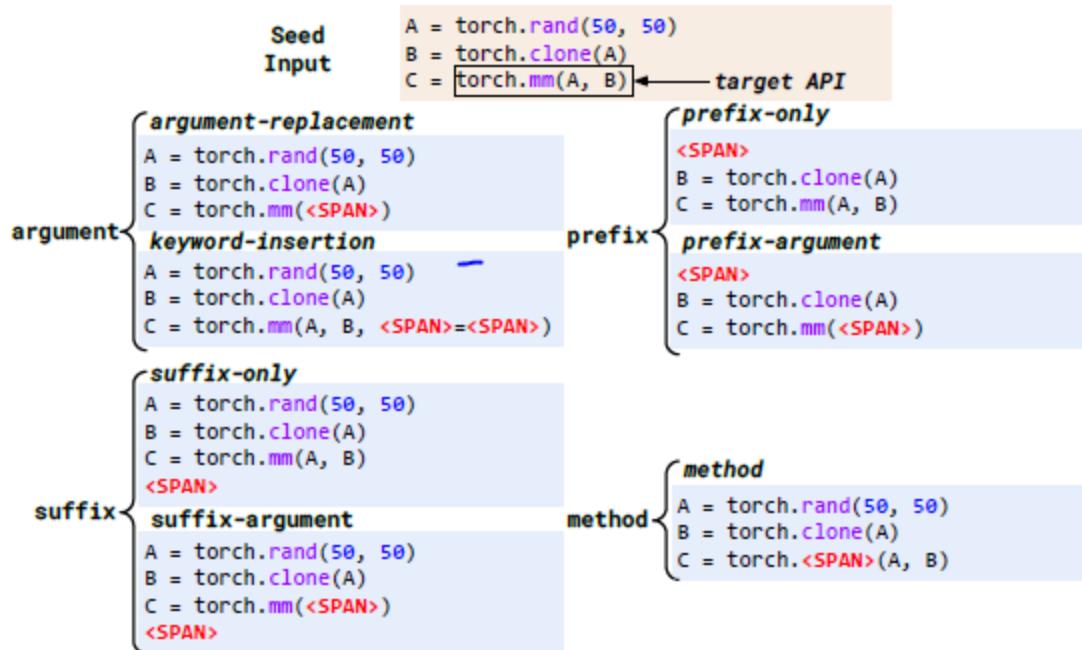


Figure 6: Mutation operators outputs (inputs for the model)

1. LLM Papers : TitanFuzz

- Contribution and Why LLM ?

- (Zero-Shot Fuzzing) LLM이 Zero-shot Task를 처리하는데 적합하다는 특성을 기반으로 추가 학습 없이 프롬프트 엔지니어링만으로 퍼징 결과를 개선할 수 있어 Zero-Shot Fuzzer로써의 가능성 시사
- (Context-aware) 복잡한 입력 제약을 가진 도메인(Deep Learning Lib)에서 입력 제약을 지킨 효과적인 test case 생성 방법을 제시
- (Fully Automatic Fuzz) 사용자의 수작업을 최소화한 완전 자동화된 퍼징 프로세스를 제시하여 기존의 규칙기반이나 입력 기반보다 효율적이고 효과적인 퍼저 제안

- Performance

- Pytorch와 Tensorflow에서 각각 1329개, 2215개의 API를 커버하며, 기존 퍼저 대비 최대 91.11% 더 많은 API 탐지
- 총 65개의 버그를 발견하였고, 이중 44개가 신규 버그 발견

1. LLM Papers : Fuzz4All

- Fuzz4All : Universal Fuzzing with Large Language Models (ISEC 2024 | Citation: 96 | UIUC | Chunqiu Steven Xia 외 5명)
- Overview
 - LLM을 활용하여 다목적 범용 퍼징을 수행할 수 있는 접근 방법 Fuzz4All 을 제안
 - LLM을 Mutation 엔진으로 활용해서 다양한 Target의 프로그래밍 언어와 시스템 테스트 대상 (SUT)에 적용한 사례로 auto-prompting 기법과 LLM based Fuzz loop를 활용해 입력 다양성을 향상
- Limitation of Traditional Fuzzers (e.g. Csmith ...)
 - L1. 타겟 시스템 및 언어에 대한 높은 의존성
 - 기존 퍼저는 특정 언어 또는 SUT에 맞게 설계되어 타 시스템이나 언어에 쉽게 재사용할 수 없다
 - E.g. Csmith는 C/C++ 컴파일러를 대상으로 하여 확장성이 낮음
 - L2. 진화에 대한 부족한 지원
 - 기존 퍼저는 새로운 기능이나 언어 업데이트에 대해 비효율적이고, 새로운 버전에서 사용 불가하거나 효과가 감소 (e.g. Csmith는 C11이전 버전만 지원하고 최신 컴파일러에서는 새로운 버그를 거의 발견x)
 - L3. 제안된 입력 생성 능력
 - 생성 기반 퍼저는 주로 특정 언어 문법에 의존하며, 변형 기반 퍼저는 고급 시드 프로그램 요구로 인해 테스트 범위가 제한적

1. LLM Papers : Fuzz4All

- Main Idea
 - 범용 퍼징 (Universal Fuzzing)
 - LLM의 범용적인 이해 능력 (즉, Zero-shot Task에 대한 성능)을 활용해 다양한 언어와 SUT에 유의미한 입력 생성을 범용적으로 접근할 수 있게 방법론을 제시
 - 자동 프롬프트 생성 (Autoprompting)
 - 사용자 입력 (e.g. 기술문서, 코드 예제) 등을 요약하고 참조하여 효율적으로 퍼징하기 위한 입력 프롬프트를 자동으로 생성
 - Fuzzing Loop
 - LLM에 의해 생성된 코드에서 예제를 선택하고 세 전략(generate-new, Mutate-existing, Semantic-equiv)을 결합하여 프롬프트를 지속적으로 업데이트하여 입력이 반복되지 않도록 제어
 - SUT에 전달된 입력을 사용자 정의된 Oracle을 통해 버그 탐지

1. LLM Papers : Fuzz4All

- Fuzz4All Workflow

- Distillation LLM

- 사용자 입력(기술문서, 예제 등)을 기반으로 압축된 프롬프트를 생성하는 요소로, GPT-4 기반의 초고성능 모델로 퍼징에 필요한 기술적 사항 자체를 이해하는데 초점을 맞춰 프롬프트 생성

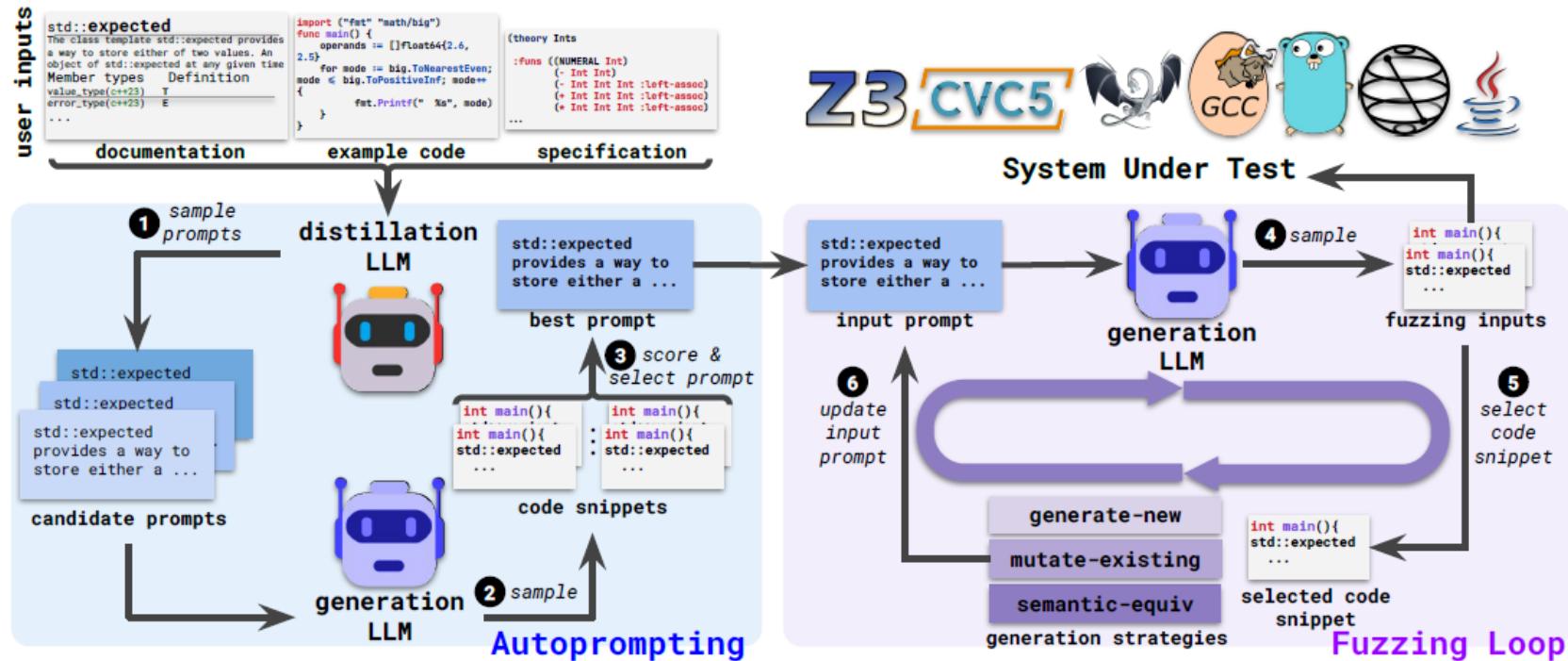


Figure 1: Overview of Fuzz4All.

1. LLM Papers : Fuzz4All

$$\text{Score}(p) = \sum_{c \in \mathcal{M}_G(p)} [\text{isValid}(c, \text{SUT})]$$

Score 함수

Fuzz4All Workflow

Generation LLM (StarCoder: Code 최적화 모델)

- Prompt를 입력받아 실제로 SUT에 적합한 코드 조각을 생성하는 LLM, 여기서 생성된 코드 조각을 기반으로 Score 함수에 의해 점수를 매겨 best prompt 선별 (p : 후보 프롬프트, $M(p)$ 주어진 프롬프트를 사용해 LLM이 생성한 입력들, $\text{isValid}(c, \text{SUT})$ 생성된 SUT가 유효한지 반한 (1 or 0))
- Fuzz Loop에서는 최종 선정된 프롬프트를 기반으로 코드 샘플을 만들어내는 요소

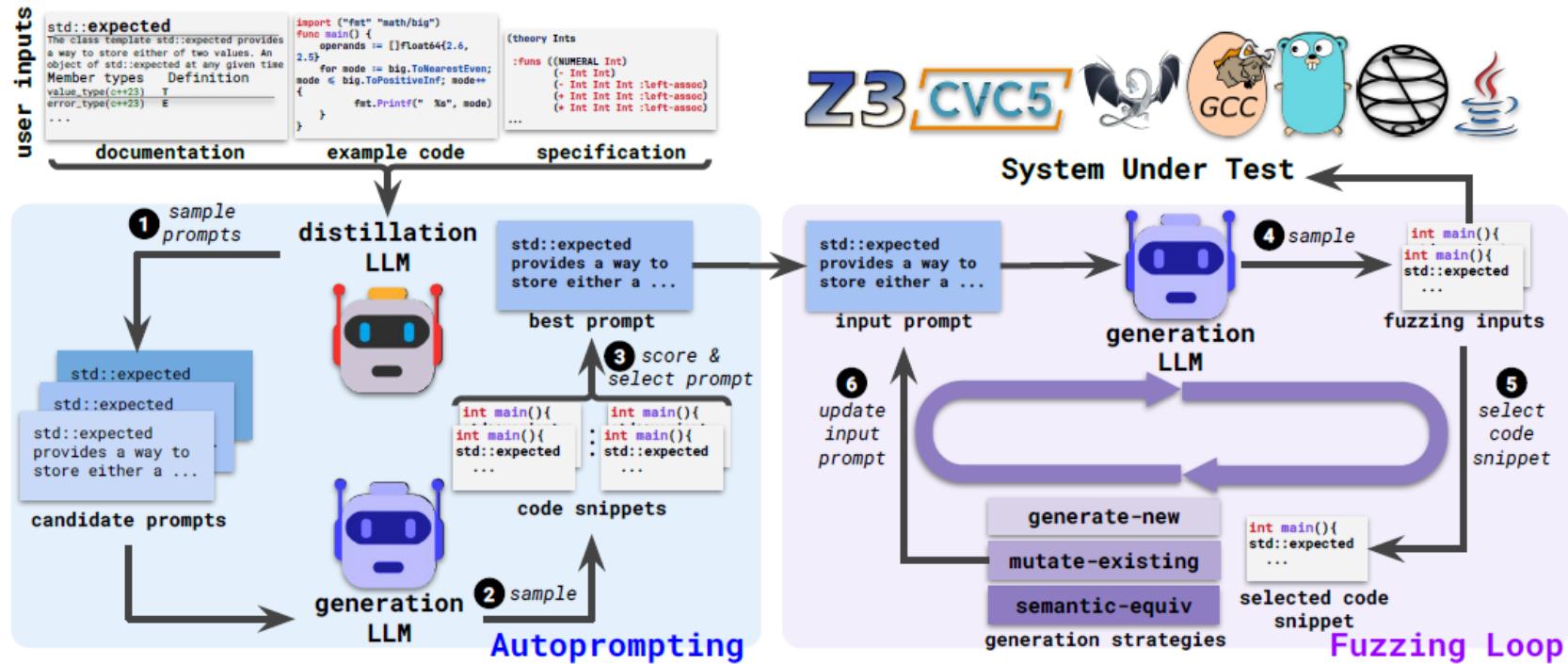


Figure 1: Overview of Fuzz4All.

1. LLM Papers : Fuzz4All

- **Fuzz4All Workflow**

- **Generation Strategies**

- 아래 세 전략을 무작위로 적용하여 입력 공간 탐색, 각 지침은 프롬프트로 생성되어 랜덤으로 LLM에게 전달
 - **Generate-new**
 - 완전히 새로운 코드 생성 전략으로 모델이 초기값 프롬프트를 기반으로 생성
 - 탐색 중심 전략으로 커버리지 확보 목적
 - **Mutate-existing**
 - 변형 기반으로 새로운 입력 생성
 - 기존 입력의 변형을 통해 edge case와 미세 변경으로 인한 버그 탐지
 - **Semantic-equiv**
 - 의미론적으로 동등한 입력을 생성
 - 기존의 입력의 논리적인 기능은 보존되지만 실제 구문이나 표현 방식을 변경
 - 시스템이 의미론적 동등성을 잘 처리하는지 테스트하는 목적
 - E.g. SMT Solver에서 $x + 1 > 3$ 대신 $1 + x > 3$ 과 같이 의미는 같지만 구문이 다른 식을 생성.

1. LLM Papers : Fuzz4All

• Contribution and Why LLM ?

- (Autoprompting for input Optimization)

Autoprompting 기법을 통해 사용자 입력을 LLM이 처리가능한 프롬프트로 요약하여 자동화된 입력을 LLM이 생성할 수 있게 유도

- (Dynamic Fuzzing Loop with LLM)

Fuzzing Loop에서 LLM 기반 생성과 변형 전략을 반복적으로 활용하여 커버리지 증가와 새로운 버그 탐지 => Coverage Plateau 문제 해결

- (Universal LLM Fuzz Process)

LLM을 이용한 퍼징 프로세스를 범용적으로 제시

• Performance

- 총 98개의 zero-day bug 확인 (GCC, Clang, SMT Solver, Qiskit[양자화 lib] ...)

- 평균적으로 기존 퍼저 대비 36.8% 높은 커버리지 달성 (Qiskit: MorphQ 대비 75.6% 증가)

- 퍼즈 입력 수 대비 다양하고 유효한 입력으로 커버리지와 탐지율 개선

- SMT2는 TypeFuzz 대비 24.9% 높은 커버리지

- Go-Library는 Go-Fuzz 대비 13.7 높은 커버리지

1. LLM Papers : ChatAFL

- Large Language Model guided Protocol Fuzzing (NDSS 2024 | Citation: 96 | National University of Singapore, Singapore | Ruijie Meng 외 4인)
- Overview
 - 프로토콜의 자연어 사양(RFC) 기반으로 기계 판독 가능한 정보를 추출하여 이를 기반으로 새로운 상태 전이를 유도하거나 초기 입력을 다양화해서 퍼징 성능을 극대화하는 ChatAFL을 제시하여 기존 퍼저들이 해결하지 못했던, 한계(L1 ~ L3)를 해결
- Limitation of Traditional Fuzzers (e.g. AFLNET, NSFUZZ ...)
 - L1. 초기 시드 입력의 제한성 (Dependence on initial seed)
 - 기존 퍼저들은 사전에 제공된 입력 시드의 품질과 다양성에 의존. 그러나 시드는 프로토콜의 상태나 입력 구조의 다양성에 대해 충분히 반영하지 못하여 탐색 가능한 범위를 제한하는 문제가 발생
 - L2. 메시지 구조 정보 부족 (Unknown message structure)
 - 기계가 읽을 수 있는 입력형식이나 메시지 구조 정보가 없는 경우, 기존 퍼저는 mutation fuzzing에 의존하게 되며 이로 인해 완전히 새로운 메시지 구조 생성에 어려움 발생
 - L3. 상태 공간 탐색의 비효율성 (Unknown state space)
 - 현재 상태를 정확히 식별하거나 새로운 상태를 효과적으로 탐색하기 위한 충분한 피드백 정보를 제공하지 않아 복잡한 구조에 대해 프로토콜은 많은 상태를 탐색치 못한채로 퍼징이 이뤄진다.

1. LLM Papers : ChatAFL

- Main Idea

- 초기 시드 다양화 (Seed Enrichment)

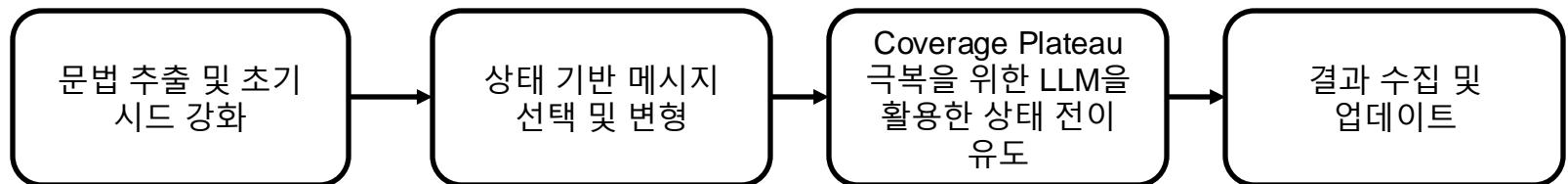
- LLM을 통해 초기 시드에 부족한 메시지 유형을 보완하여 기존 시드의 다양성과 품질 향상
 - 초기 시드가 특정 메시지에 제한적이여도 LLM은 기존의 학습 데이터를 기반과 문서를 기반으로 새로운 메시지 구조를 생성하여 상태 공간 확장 => L1을 해결

- LLM 기반 문법 추출 및 구조적 변형 (grammar-Aware Mutation)

- LLM에게 자연어로 기술된 RFC의 문서를 참조시켜 메시지 구조(Grammar)를 추출, 기계 판독이 가능한 형식으로 변환 이를 활용하여 메시지 구조적 안정성을 유지하면서 변형이 되게하여 새로운 상태 탐지 가능 => L2 해결

- 상태 전이를 유도하는 입력 생성 (State-Inference & Coverage Plateau Handling)

- 클라이언트와 서버 간의 통신 기록을 바탕으로 새로운 상태 전이를 유도할 수 있는 다음 메시지 생성
 - 기존 퍼저의 문제인 코드 및 상태 커버리지 정체 문제를 해결하여 새로운 상태 탐지 유도 => L3 해결



1. LLM Papers : ChatAFL

• Contribution and Why LLM ?

- (LLM-Guided Protocol Fuzzing)

LLM은 인터넷에 공개된 RFC 문서를 사전에 추가 학습 없이도 효과적으로 자연어 기반으로 기계 판독 가능한 문법으로 변환이 가능한 특성을 이용해 기존 퍼저의 문제들을 해결 (C1~3), 이는 추가 학습 없이도 대응 능력이 뛰어난 LLM의 zero-shot task에 대한 특성

- (Surpassing Coverage Plateau)

기존 프로토콜 퍼저들의 커버리지 정체 문제를 LLM을 이용한 새로운 상태 전이 유도 입력 생성으로 해결하는 전략을 제시

- (Context-aware Mutation)

문법적 구조를 유지하면서 유효한 변형 수행 가능, 이를 통해 메시지 유효성에 대한 불필요한 에러 감소

- (Automated fuzzing Process with LLM)

추가적인 규칙 설계나 수작업 없이 다양한 프로토콜에 적용 가능 (실제로 RTSP(Live555)를 대상으로 설계되었으나, 다른 프로토콜(ProFTPD, Kamailio ...)에서도 제로데이 취약점 발견)

• Performance

- 9개의 Zero-day 취약점 발견, AFLNET은 3개 NSFUZZ는 4개를 발견
- 발견 취약점 중 다수는 Remote Code Execution 취약점으로 CVE에 보고
- AFLNET에 비해 47.6% NSFUZZ 대비 42.69% 높은 상태 전이 탐색
- AFLNET 대비 5.81%, NSFUZZ 대비 6.74% 높은 브랜치 커버리지 달성

1. LLM Papers : ProphetFuzz

```
ffmpeg -copyts -start_at_zero -y -itsoffset offset \
-itsscale scale -ss position -sseof position -i url
ffmpeg option example
```

- ProphetFuzz: Fully Automated Prediction and Fuzzing of High-Risk Option Combinations with Only Documentation via Large Language Mode (CCS 2024 | Citation: 0 | Zhongguangcun Laboratory, Beijing, China | Dawei Wang 외 5명)
- Overview
 - 소프트웨어 보안 테스트에서 옵션 조합에 의한 취약점을 LLM을 활용해서 자동으로 예측하고 퍼즈 테스트를 수행하는 ProphetFuzz 제안
 - 사양 문서의 입력만을 이용하여 옵션 간 제약을 정밀하게 추출하여 고위험 옵션 조합을 예측하고 이걸 테스트하여 기존의 퍼징 기법의 제약(L1~L3)을 해결
- Limitation of Traditional Fuzzers (e.g. AFLargv, Tofu, CarpetFuzz...)
 - 옵션 테스트는 다양한 동작을 정의, 그러나 조합 복잡도가 높아 실행 경로를 급격히 증가, 테스트의 효율성 감소 (e.g ImageMagick은 305개의 옵션으로 2^{305} 개의 조합 발생)
 - L1. 우선순위 부재 (Lack of Prioritization)
 - 모든 조합을 동일한 우선순위로 취급하여 테스트의 효율성 저하
 - L2. 의미론적 불일치 (Semantic Mismatch)
 - 생성된 명령어와 입력 파일 간의 비 일관성으로 실행 오류 발생
 - L3. 전문가 의존성 (Reliance on Expertise)
 - 보안 전문가에 대한 높은 의존도가 요구되어 확장성, 자동화 가능성에 제약

1. LLM Papers : ProphetFuzz

- Main Idea

- 완전 자동화 퍼즈 프로세스 (Fully Automated Fuzz Process)
 - 소프트웨어의 공식 문서만을 입력으로 사용하여 옵션 간 제약조건을 추출하고 고 위험 옵션 조합을 예측하게 설계한 완전 자동화 도구 (Reliance on Expertise 문제 해결)
- LLM 기반 제약 조건 추출 (LLM-based Constraint Extraction)
 - 문서에서 옵션 간 상호 종속성 및 충돌 등의 요소를 LLM을 통해 자동 추출하고, Self-Check 기법을 통해 높은 정확도(94.00%)를 확보하여 오류를 제거
- 고 위험 옵션 조합 예측 (Prediction of High-Risk Combinations)
 - Few-shot 학습과 자동 생성(Auto-CoT)된 사례를 통해 과거의 패턴을 학습해 취약점 발생 가능성이 높은 조합 예측
 - Few-shot Learning : LLM의 추론에 몇몇 예제를 추가로 제공하여 구체적인 패턴을 학습하고 결과를 도출 할 수 있게 유도하는 방식으로 Zero-shot보다 높은 정확도를 보장

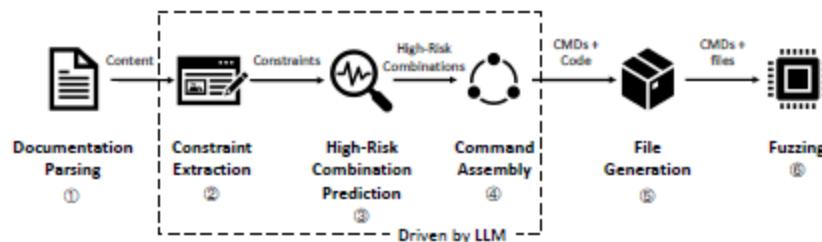


Figure 1: Overview of ProphetFuzz.

1. LLM Papers : ProphetFuzz

① Documentation Parsing

키워드 매칭을 통한 파싱

```

NAME      ffmpeg -ffmpeg video converter
DESCRIPTION ffmpeg is a very fast video and audio converter ...
SYNOPSIS ffmpeg [global_options] [[input_file_options] -i ...
Options
  -url (input)
    input file url
  -sseof position (input)
    Like the "-ss" option but relative to the "end of ...
  -copyts
    Do not process input timestamps, but keep ...
  -start_at_zero
    When used with copyts, shift input timestamps ...
  -y (global)
    Overwrite output files without asking.
  ...

```

Parsing document content into structured data

JSON 형식으로 추출

```
{
  "name": "ffmpeg",
  "description": "ffmpeg is a ....",
  "synopsis": "ffmpeg [global_options] ....",
  "options": [
    {"url (input)": "input file url",
     "sseof position (input)": "Like the \"-ss\" ...",
     "-copyts": "Do not process input timestamps ...",
     "-start_at_zero": "When used with copyts, shift ...",
     "-y (global)": "Overwrite output files without ...",
     ...
   ]
}
```

② Constraint Extraction

Guide LLM to extract potential conflict/dependency

Conflict:

Dependency:

Conflict:

Dependency:

③ High-Risk Combination Prediction

Predict high-risk combinations while adhering to identified constraints

Combinations:

Few-shot과 Auto CoT 기법을 이용해서 고위험 조합 예측

Figure 2: Example of ProphetFuzz.

예측한 조합을 이용하여 적합한 명령어와 입력 파일을 생성이 가능하게 구성

④ Command Assembly

Generate proper commands and file generation code

Command:

Code:

⑤ File Generation

Execute the code to produce the files in the sandbox

Fuzzing commands:

Seed corpus:

생성 및 퍼징

⑥ Fuzzing

Perform option-aware fuzzing

1. LLM Papers : ProphetFuzz

• Self-Check 기법

- LLM의 hallucination(잘못된 정보 생성) 문제로 인해 고 위험 옵션 조합의 문제 발생 위험성을 고려해 정확도 개선을 위해 고안 및 활용
- 제약 조건을 검증하기 위해 두 가지 질문을 이용하여 검증
 - 검증 질문 : 옵션 A는 반드시 B 없이 사용되어야 하는가 ?
 - 반례 질문 : 옵션 A는 B와 함께 사용될 수 있는가 ?
- 위 두 질문의 답변이 LLM에서 충돌하지 않을 경우 올바른 제약으로 판단
- 이 과정에서 temperature 하이퍼 파라미터를 조정하여 신뢰도 향상
- Temperature는 랜덤성과 정확성 중 LLM이 어디에 비중을 두고 답변할지에 관여하는 파라미터
 - Start_at_zero는 타임 스탬프를 0부터 시작하도록 조정
 - copyts는 기존 타임 스탬프를 유지하는 옵션
- E.g ffmpeg의 -start_at_zero는 -copyts 없이 사용할 수 있는가 ?와 -start_at_zero는 -copyts와 함께 사용할 수 있는가를 이용해 이 제약은 오류임을 확인
- 결과적으로 52개의 문서에서 633개의 제약조건을 추출하고 정확도 94% 달성
- self-check 없이는 23.41%

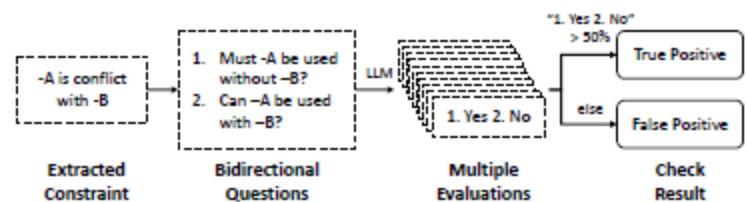


Figure 3: Workflow of the Self-Check approach.

1. LLM Papers : ProphetFuzz

- **Few-Shot Learning**
 - 문서와 과거 고 위험 옵션 조합 간의 관계를 학습하여 LLM이 제약조건을 준수하면서 취약점이 발생할 가능성이 높은 조합을 예측하도록 유도
 - Github 이슈 및 취약점 보고서에서 8개의 프로그램에 대한 고 위험 옵션 조합 29개를 수집하여 few-shot Learning을 위한 Auto-CoT에 제공
 - 이 작업은 수동으로 이뤄지기에 앞의 Fully Automatic에 위배되지만, 저자는 이러한 데이터 수집은 일회성(one-time job) 작업이라고 설명
- **Auto-CoT (Chain-of-Thought)**
 - LLM이 복잡한 옵션 조합 관련 문제를 단계적으로 접근하여 논리적 추론 능력을 강화하여 예측 성공률을 높이는데 사용됨
 - CoT와의 차이로 위에서 수집된 여러 데이터를 기반으로 유사한 few-shot을 자동 생성하여 프롬프트와 통합
 - 작동 방식
 - 프로그램의 기능 이해 (e.g. ffmpeg는 비디오 및 오디오 데이터 처리 도구입니다. 옵션의 예는 이런 게 있습니다.)
 - 각 옵션의 역할 및 효과 분석 (옵션은 copyts 기준 스템프를 유지해야하고 ... 등등)
 - 추출된 제약 조건 상기 (앞의 self-check 기법을 이용해 도출된 옵션간 제약을 꼭 참조하라고 명시)
 - 제한을 준수하는 조합 탐색 (가설에 근거하여(Hypothetically)라는 표현을 사용하여 대담한 추정 유도)
 - 메모리 취약점을 쉽게 유발할 수 있는 추가 옵션 탐색을 지시 (--sseof position 같은 위험 옵션 추가)
 - JSON 형식으로 결과 출력 강제 (실제 구체적 예시를 few-shot으로 제공하여 형태 고정 지시)

1. LLM Papers : ProphetFuzz

• Contribution and Why LLM ?

- **(Documentation-Driven Prediction and Fuzz)**

LLM의 뛰어난 텍스트 이해 및 생성 능력을 기반으로 소프트웨어 문서 입력만으로 옵션 조합과 고 위험 취약점을 예측 및 퍼즈 테스트하는 방법론을 제시

- **(Constraint Extraction and Risk Awareness)**

Self-Check 기반으로 옵션 간 제약조건을 정확히 추출(94%)해서 해당 제약을 기반으로 옵션 조합을 예측하여 의미적으로 일치하는 입력을 보장

- **(Few-shot Learning for High-Risk Prediction)**

고 위험 사례를 Few-Shot 학습 사례로 자동 생성하는 기법을 통해 초기 한번의 생성 되에 전문가의 지속적 개입 없이 고 위험 예측에서 정확도 높은 퍼즈 프로세스를 달성

- **(Fully Automated Fuzizng Process with LLM)**

입력 데이터 생성까지 거의 모든 부분을 LLM이 완전 자동화되게 수행하여 사용자의 수작업을 최소화한 퍼지 프로세스 구현

• Performance

- 52개의 프로그램에서 1748개의 고 위험 옵션 조합을 예측하고 7614개의 퍼징 명령어를 실행해서 364개의 고유 취약점 발견 이는 기존의 SOTA(state-of-the-art) 모델(CarpetFuzz) 대비 32.85% 높은 탐지율

- 140개의 제로데이/하프데이 취약점을 발견하고 이중 93개가 개발자에게 확인, 21개가 CVE 번호 부여

2. Summary

퍼저	주요 특징	대상 영역	사용된 접근법	주요 성과
TitanFuzz	<ul style="list-style-type: none"> - LLM을 활용한 딥러닝 lib 퍼저 - API 호출 코드 생성 및 변형 기반 퍼징 수행 - Mutation Operator를 활용한 변형 - 생성-변형을 적절히 도입한 workflow 	딥러닝 라이브러리 (TensorFlow, PyTorch) DB, Language 등 그 외 영 역 확장 가능	<ul style="list-style-type: none"> - LLM을 이용해 API 호출 코드를 생 성 - 입력 제약을 만족하는 생성 및 변 형 - Codex와 Infilling Model, EV-Fuzz 를 활용 - Mutation Operator 	<ul style="list-style-type: none"> - 커버리지 30 ~ 50% 증가 - 65개 버그 발견 (44 개의 zero-day)
Fuzz4All	<ul style="list-style-type: none"> - 다중언어를 지원하는 범용 LLM 퍼저 - Auto Prompting을 통한 프롬프트 개선 과정 추가 - Generation Strategies를 통해 퍼즈 개선 	다양한 시스템 (범용적으로 사용되는 SUT, Language, Compiler ...)	<ul style="list-style-type: none"> - Auto Prompting을 활용한 동적 입력 생성 전략 - LLM 기반의 Mutation 엔진 활용 - Generation Strategies 	<ul style="list-style-type: none"> - 6개 언어에서 기존 퍼저 대비 36.8% 높은 커버리지 달 성 - 99개의 버그 발견 (64개의 zero-day)
ChatAFL	<ul style="list-style-type: none"> - Network Protocol 구현을 위한 상태 기반 퍼징 - 프로토콜 문서 추출 및 Grammer 기반 생성 	Network Protocols	<ul style="list-style-type: none"> - LLM 기반 문법 추출 및 구조적 변 형 - 상태 전이 유도 생성 - 초기 시드 강화 및 Grammar Aware Mutation 	<ul style="list-style-type: none"> - 상태 전이 커버리 지 47.6% 증가 - 9개의 신규 취약점 발견
ProphetFuzz	<ul style="list-style-type: none"> - 고 위험 옵션 조합 예측 및 퍼징 - 문서만으로 완전 자동화된 테스트 수행 - 소프트웨어 문서를 자동화하여 LLM에게 하는 방법을 제시 - Auto-CoT 기법 이용 	- 소프트웨어 옵션 조합 테 스트	<ul style="list-style-type: none"> - LLM 기반 제약 조건 추출 - Few-shot Learning - Auto-CoT를 통한 고 위험 조합 예 측 	<ul style="list-style-type: none"> - 1748개의 고 위험 옵션 조합 예측 - 364개의 고유 취약 점 발견 (93개의 zero-day 취약점, 21개의 CVE)

퍼저	Open Source? Closed Source			
TitanFuzz				
Fuzz4All				
ChatAFL				
ProphetFuzz				

2. Summary

- LLM Mutation을 활용했을 때 장점
 - Fully Automatic Fuzz Process
 - 거의 완전 자동화된 Fuzzing 프로세스를 구축
 - Context Aware Mutation
 - 제약 조건을 학습하여 유효한 입력을 생성하는 입력기를 만들 수 있다.
 - Zero-Shot Task에 대한 높은 성능과 Few-shot의 일부 도입으로 인한 정확도 향상
 - 추가적인 학습 없이 Prompt를 개선한 LLM만으로도 Fuzzing의 다양한 task에 대한 성능을 개선할 수 있다.
 - Few-shot을 일부 도입하여 정확도 향상 가능
 - Coverage Plateau 문제 해결
 - 새로운 입력을 다양한 기법으로 창의적으로 유지하여 커버리지 증가율 정체 문제 해결 가능
 - Universal Fuzz : 높은 확장, 범용성
 - 퍼즈 대상의 버전의 변화나 비슷하지만 다른 Task에 대한 퍼즈 등 특정 Target에 Fit하지 않아도 적절히 동작하는 확장성 높은 퍼저를 구현 가능
 - Seed 공간 다양화
 - LLM을 통해 초기 시드를 다양화하여 입력 공간을 다양하고 넓게 유지

2. Summary

1. Robot Mutation이 필요한 이유 정리
2. Humble 기반의 PX4 Source 기반으로 llama_ros를 이용해 node_graph 생성
3. PX4 phy-cyb 문서 RAG 후 Mutation Message 생성