White-box Compiler Fuzzing Empowered by Large Language Models

CHENYUAN YANG, University of Illinois Urbana-Champaign, USA
YINLIN DENG, University of Illinois Urbana-Champaign, USA
RUNYU LU, Huazhong University of Science and Technology, China
JIAYI YAO, The Chinese University of Hong Kong, Shenzhen, China
JIAWEI LIU, University of Illinois Urbana-Champaign, USA
REYHANEH JABBARVAND, University of Illinois Urbana-Champaign, USA
LINGMING ZHANG, University of Illinois Urbana-Champaign, USA

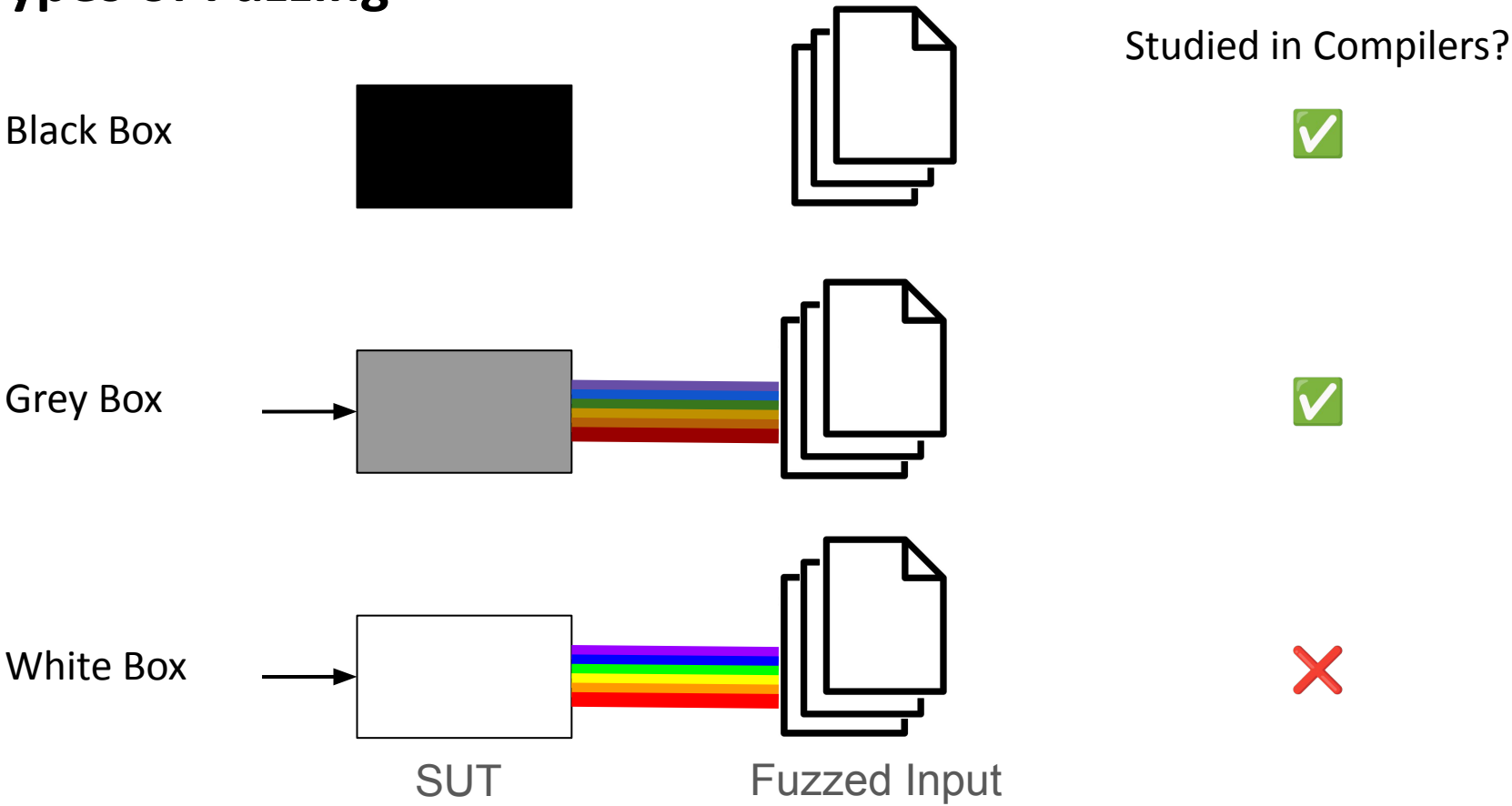
Presented by Group 2:
Advait Iyer, Sam Jaehnig, Leah MacKay, Julianne Shah & Daphne Tsai

Motivation

Compilers are huge, complex software systems

- Optimizations triggered by small edge cases
- LLVM 14M Lines of Code
- Needs to be tested, but how?

Types of Fuzzing



Studied in Compilers?

✅

✅

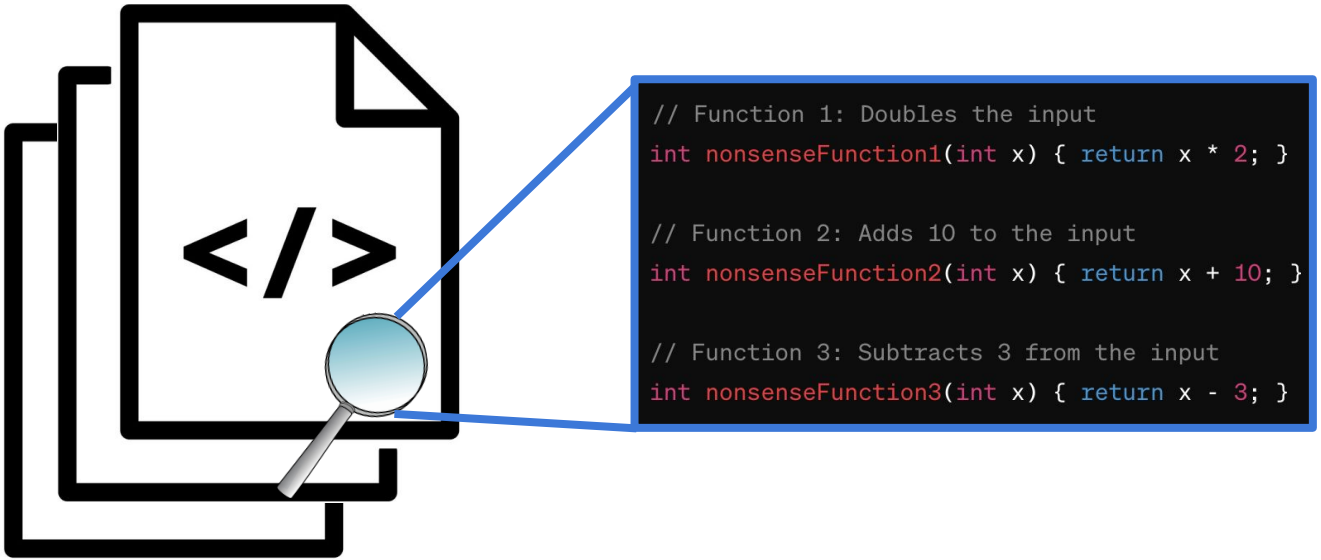
❌

LLMs for Generating Software Tests

System Fuzzers: ✅

Module Unit Testers: ✅

Compiler Fuzzers: ❌



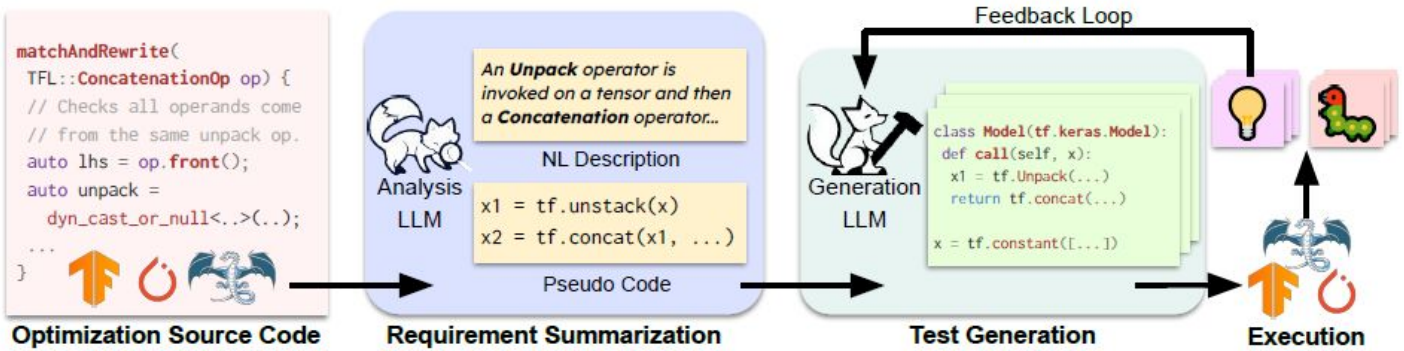
```
// Function 1: Doubles the input
int nonsenseFunction1(int x) { return x * 2; }

// Function 2: Adds 10 to the input
int nonsenseFunction2(int x) { return x + 10; }

// Function 3: Subtracts 3 from the input
int nonsenseFunction3(int x) { return x - 3; }
```

WhiteFox and Dual Model Framework

- **WhiteFox**: test compiler optimizations by leveraging LLMs
- Dual-Model Approach:
  (1) *Analysis* LLM
  (2) *Generation* LLM
- **Feedback loop**: incorporating successful test cases back into the model



Overview of WhiteFox.

Compiler White Box Fuzzing through WhiteFox

- Requirement:
  - LLM: generated test cases
- Test Generation:
  - Generation LLM
  - Employs a feedback loop
- Thompson Sampling
  - Test generation
  - Exploitation to optimize test generation

Target Optimization

```
### Please generate different valid [TARGET INPUT] example with
[INPUT SPECIFICATION] meets the specified requirements.

# Description
The [TARGET INPUT] should contain the following pattern:
```
[PSEUDO CODE]
```
This pattern characterizes scenarios where [NL DESCRIPTION].

# [TARGET INPUT]
[EXAMPLE TRIGGERING INPUT #1]

# [TARGET INPUT]
[EXAMPLE TRIGGERING INPUT #2]

# [TARGET INPUT]
[EXAMPLE TRIGGERING INPUT #3]

# [TARGET INPUT] [TO BE GENERATED]
```

Prompting for test generation with feedback.

Evaluation Overview

Evaluate **WhiteFox** from these perspectives:

- How is each component of the architecture relevant? (ablation study)
- How does it perform relative to existing compiler fuzzing technologies?
- Is it useful in detecting “real-world” bugs?

Evaluation (Ablation Study)

Requirement Formatting:

- **WF-mix** = default
- **WF-NL** = natural language
- **WF-Code** = pseudo-code
- **WF-Impl** = source code

Loop Configurations:

- default (Thompson Sampling)
- **Wf-No-Feedback**: no feedback loop
- **Wf-Random**: random feedback

Other:

- **WF-StarCoder**: StarCoder replaces GPT4 in analysis LLM

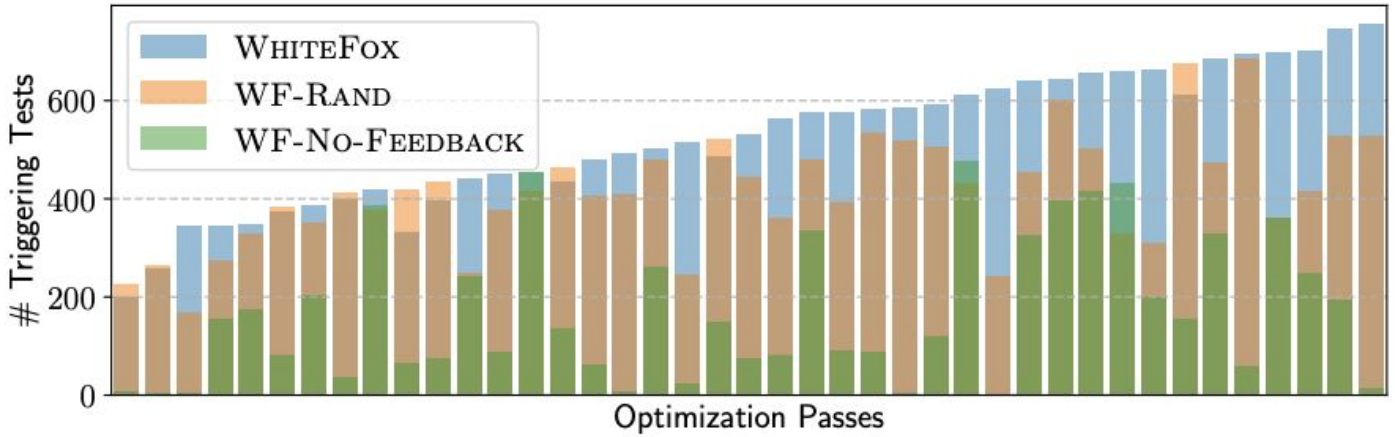
Results (Ablation Study)

- **WF-mix** (natural language & pseudo-code hybrid) tests triggered the most compiler optimizations
- Using GPT4 performs better than StarCoder for the analysis step

	# Triggered optim.	% Total optim.	# Triggering tests	% Total tests
WF-Mix	**39**	**60.9%**	**1,113**	**17.4%**
WF-NL	37	57.8%	940	14.7%
WF-Code	32	50.0%	1,055	16.5%
WF-Impl	32	50.0%	638	10.0%
WF-Starcoder	32	50.0%	745	11.6%

Results (Ablation Study)

- A feedback loop (specifically Thompson Sampling) performs best



Evaluation (Relative Performance)

Compiler Frameworks:

- 3 deep learning compilers (PyTorch Inductor, TensorFlow Lite, TensorFlow-XLA) & LLVM for C/C++

Existing Fuzzers:

- **TitanFuzz** (LLM-based) & **NNSmith** (symbolic rule based) DL fuzzers
- **YarpGen** for C/C++ fuzzing & **GrayC** for grey-box C fuzzing

Evaluation (Real-world Applications)

Main Goal:

- # of detected bugs

WhiteFox Sub-goals:

- # of triggered optimizations
- # of optimization-triggering tests

Results (Real-world Applications)

- WhiteFox detects **65 bugs** in PyTorch
  i. 10 detected by TitanFuzz, 3 by NNSmith
- **51 bugs** (of 65) have been fixed
  i. 5 are *high-priority*
  ii. 32 are compiler-opt related; 2 detected by competitors

	Total	Confirmed	New	Won't fix	Pending
PyTorch	65	62	60	3	0
TensorFlow Lite	11	8	8	2	1
TensorFlow-XLA	12	10	10	0	2
LLVM	8	2	2	3	3
Total	96	82	80	8	6

Results (comparison with baselines)

		# Optim.	# Triggered optim.	# Triggering tests	# Tests	Time (hour)
PyTorch Inductor	WhiteFox	61	**41**	**21,469**	61,000	41.1
	WhiteFox-Mini		39	1,737	6,100	4.2
	TitanFuzz		4	5,519	521,251	76.6
	NNSmith		5	47	12,084	4.9
TensorFlow Lite	WhiteFox	13	**12**	2,801	12,000	18.1
	WhiteFox-Mini		10	305	1,200	1.1
	TitanFuzz		8	571	243,288	59.0
	NNSmith		7	**4,666**	117,381	6.8
TensorFlow-XLA	WhiteFox	49	20	12,990	49,000	59.7
	WhiteFox-Mini		19	1,307	4,900	5.3
	TitanFuzz		**22**	45,762	243,288	63.2
	NNSmith		16	**117,006**	117,381	6.0
LLVM	WhiteFox	52	**26**	**25,322**	52,000	30.9
	YARPGen		3	3,352	6,948	28.1
	GrayC		4	8,353	107,234	30.6

Critique: Strengths

- **Overall Merit**: Current compiler fuzzing methods are lacking.
- **Experimental Methodology**: WhiteFox was evaluated against an already existing LLM fuzzer.
- **Experimental Methodology**: The paper features an extensive ablation study.

Critique: Weaknesses

- **Writing Quality**: The paper was written with very large paragraphs.
- **Novelty**: LLMs have already been used for fuzzing software systems.
- **Experimental Methodology**: Only 4 compilers are used for fuzzing evaluation, and 3 of which were deep learning python compilers.
- **Experimental Methodology**: WhiteFox was evaluated against only two non-NN based compiler fuzzers.

Questions?

Thank You!

Potential Papers

1. Model, Design, and Evaluation of a Compiler for a Parallel Processing Environment Very old (1977) and not ML
2. Machine Learning in Compiler Optimization
3. White-box Compiler Fuzzing Empowered by Large Language Models THE WINNER :)
4. https://arxiv.org/abs/2312.04511 github:https://github.com/SqueezeAILab/LLMCompiler

https://www.when2meet.com/?24296983-vtlLt

Guidelines

13 minutes hard limit, then 2 minutes for questions

Max 20 slides, everyone must talk

Submit slides and paper by 9pm on tuesday!!!

- Don’t just lift figures from the pdf (graphs/tables ok to lift)
- Don’t have too many all text slides
- No long sentences on slides, don’t just read the slides, look at audience
- Equations/proofs not very interesting to show, code examples are great points to discuss

Implementation/Set-up Details

1. Optimization Collection
    a. Identify relevant source code

2. Instrumentation
    a. Add logging to identified *compiler optimization* functions

3. Analysis & Generation LLMs
    a. GPT4 for the *analysis* LLM, StarCoder for the *generation* LLM

4. Few-shot Prompting
    a. Initialize with one-shot prompting (optimization, sample requirement description/test input), internally the feedback loop utilizes three-shot prompting