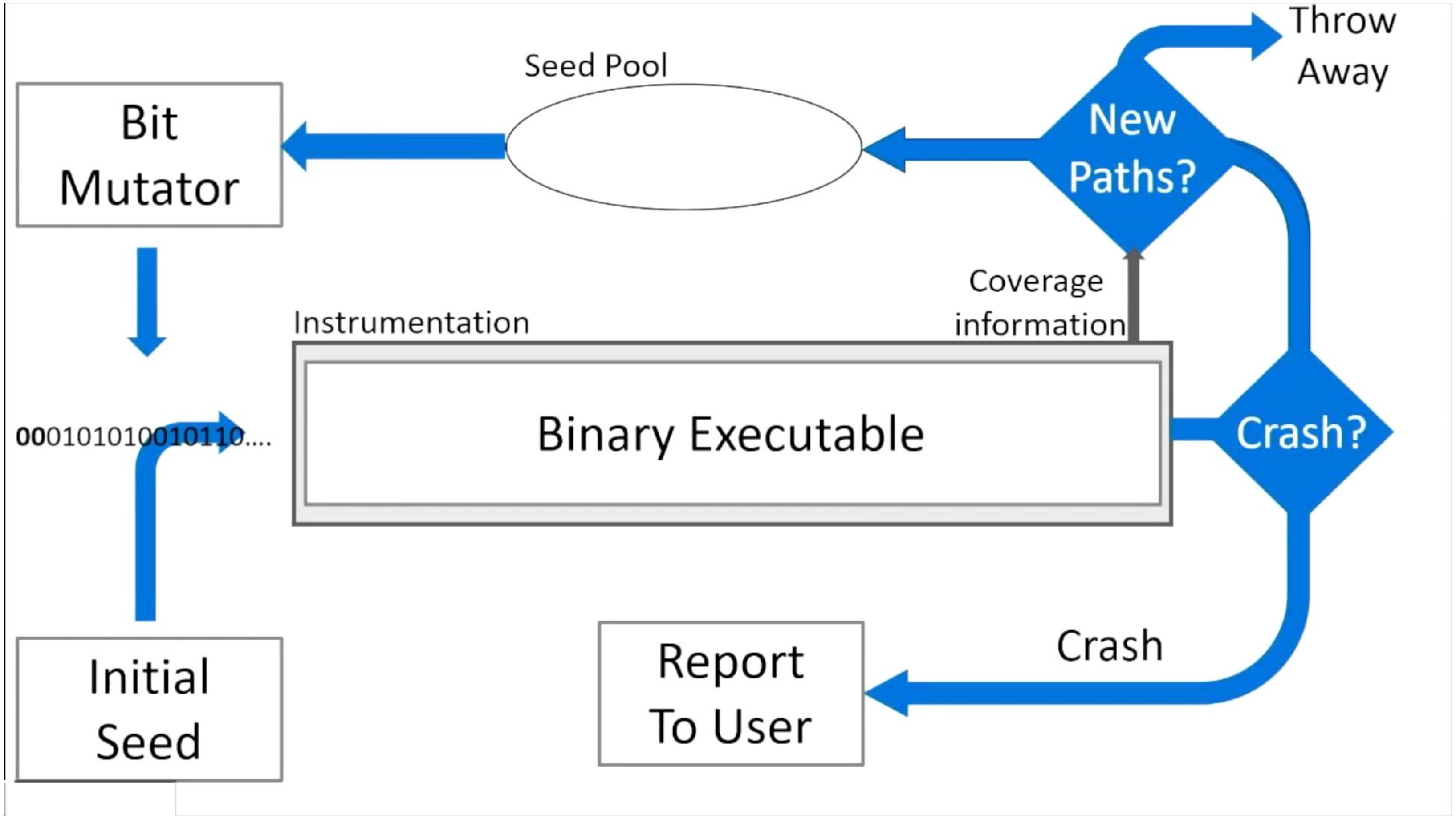
Fuzzing with LLMs

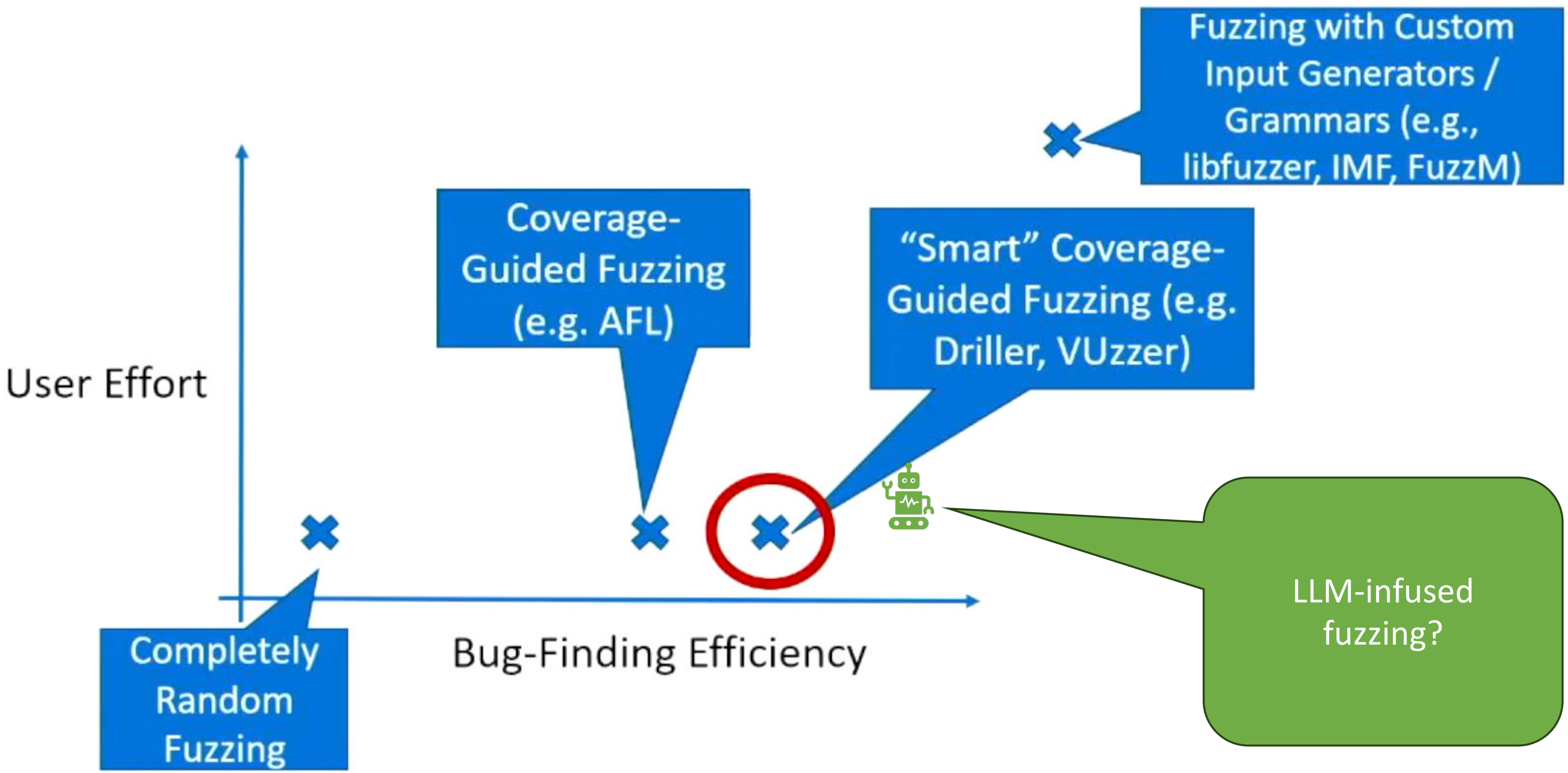
Presented by David Miller

Background: fuzzing



Pilfered from Leo’s colloquium talk “Adventures in Property-Based Testing” Sept. 9, 2022, 17:00

Tl;dr



Adapted from Leo's colloquium talk "Adventures in Property-Based Testing " Sept. 9, 2022, 32:40

Large Language Models are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, Lingming Zhang

UIUC and USTC
ISSTA 2023: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, July 2023

Prior fuzzing for deep learning code

- ML code is hard!
  - Python is dynamically typed
  - Shape errors prevent more interesting tests
- API-based vs. model-based fuzzers
- API-based:
  - Targets individual APIs, perhaps one line of code
- Model-based
  - Create a larger model (using common APIs)
  - Compare results across different backends, e.g., of Keras (here, CPU/GPU)

Method

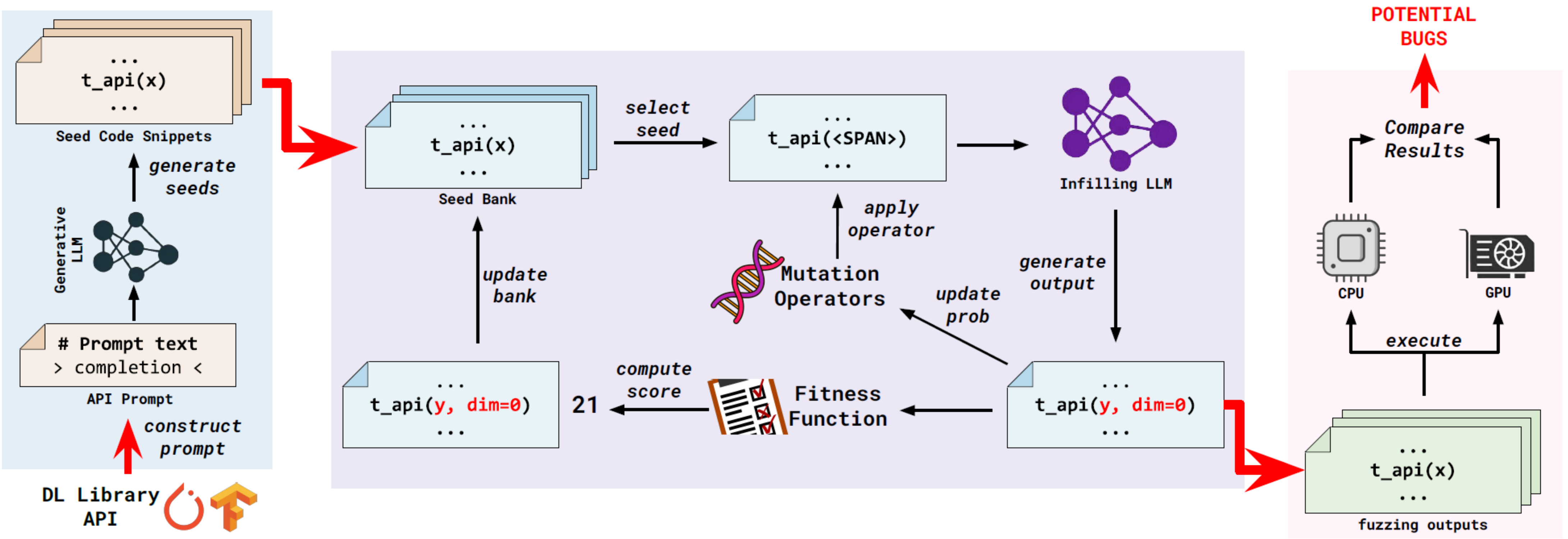


Figure 4: Overview of TitanFuzz

Prompting Codex (generator)

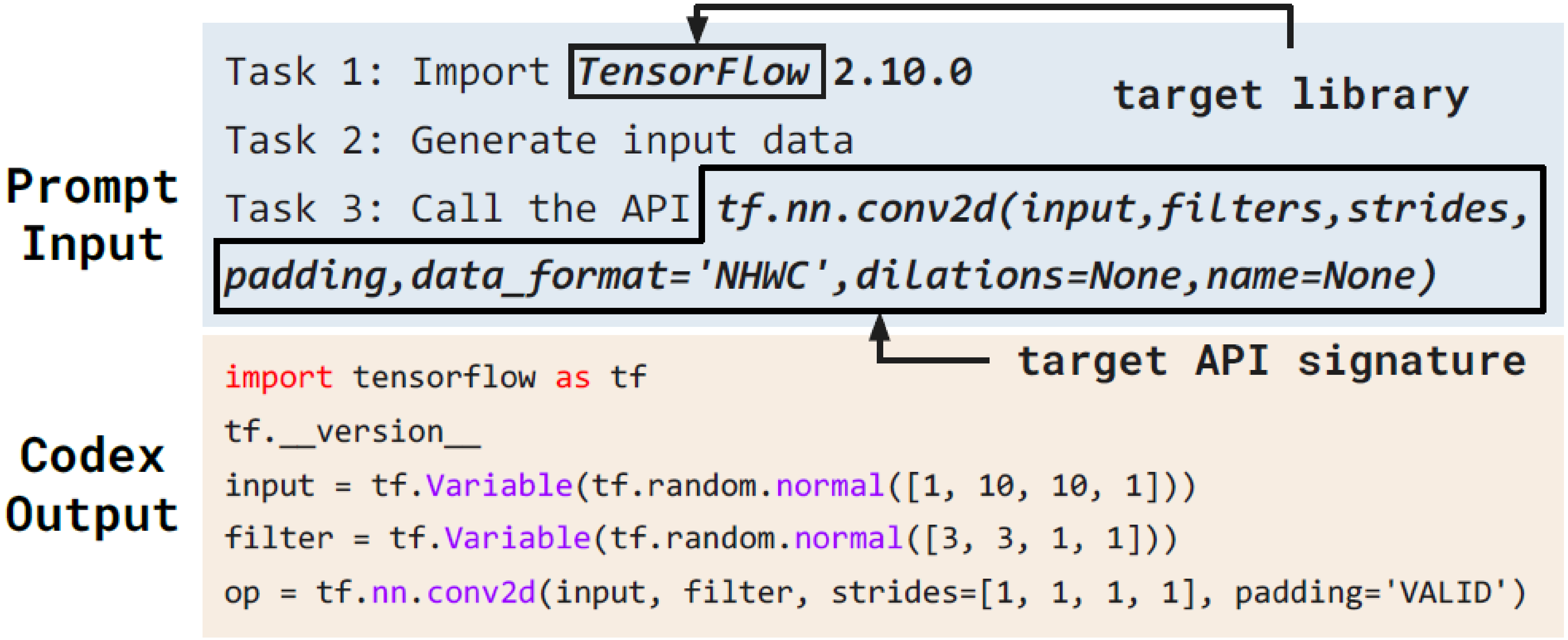


Figure 5: Example generation from the Codex model.

Mutation operators (for infilling with InCoder)

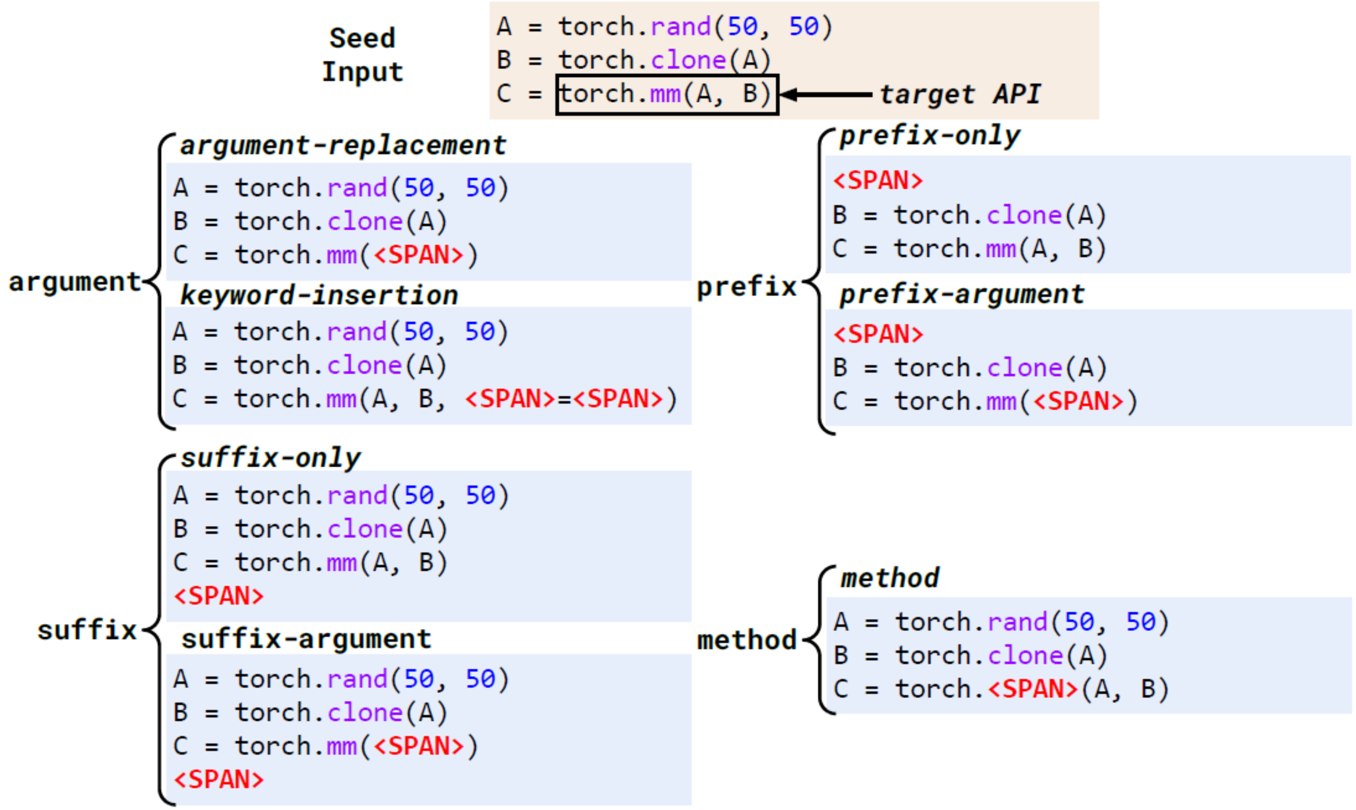


Figure 6: Mutation operators outputs (inputs for the model)

Algorithm 1: Evolutionary fuzzing algorithm

```
Function EvoFuzz(API, Seeds, T_Budget):
    Input  : The test target API, the seed programs Seeds,
             the time budget T_Budget
    Output : The generated programs

    SeedBank ← Seeds
    InitializeMPrior ()
    while T_Elapsed ≤ T_Budget do
        CurrentSeed ← SelectSeed (SeedBank)
        MutationOp ← SelectMutationOp ()
        MaskedInput ← Mask (CurrentSeed, MutationOp)
        Samples ← InCoder (MaskedInput)
        ValidSamples, InvalidSamples ← Evaluate
          (Samples)
        UpdateMPosterior (MutationOp, Count
          (ValidSamples), Count (InvalidSamples))
        FitnessScore ← FitnessFunction (ValidSamples)
        SeedBank ← SeedBank ∪ ValidSamples
    return SeedBank
```

Depth of dataflow graph + # API calls - # repeated calls

D + U + R

Algorithm 2: Mutation operator selection algorithm

```
Function InitializeMPrior():
    for m ∈ MutationOps do
        m.S, m.F ← 1, 1

Function SelectMutationOp():
    Output: The chosen mutation operator m
    for m ∈ MutationOps do
        Sample θ_m ~ Beta(m.S, m.F)
    m* = argmax_m θ_m
    return m*

Function UpdateMPosterior(m, NumValid, NumInvalid):
    m.S ← m.S + NumValid
    m.F ← m.F + NumInvalid
```

Successes/ failures for that API/op

Evaluation metrics

- Coverage
  - APIs
  - Lines of code
- Number of “unique” valid programs
- Execution time
- Bugs detected

Coverage vs. time comparison

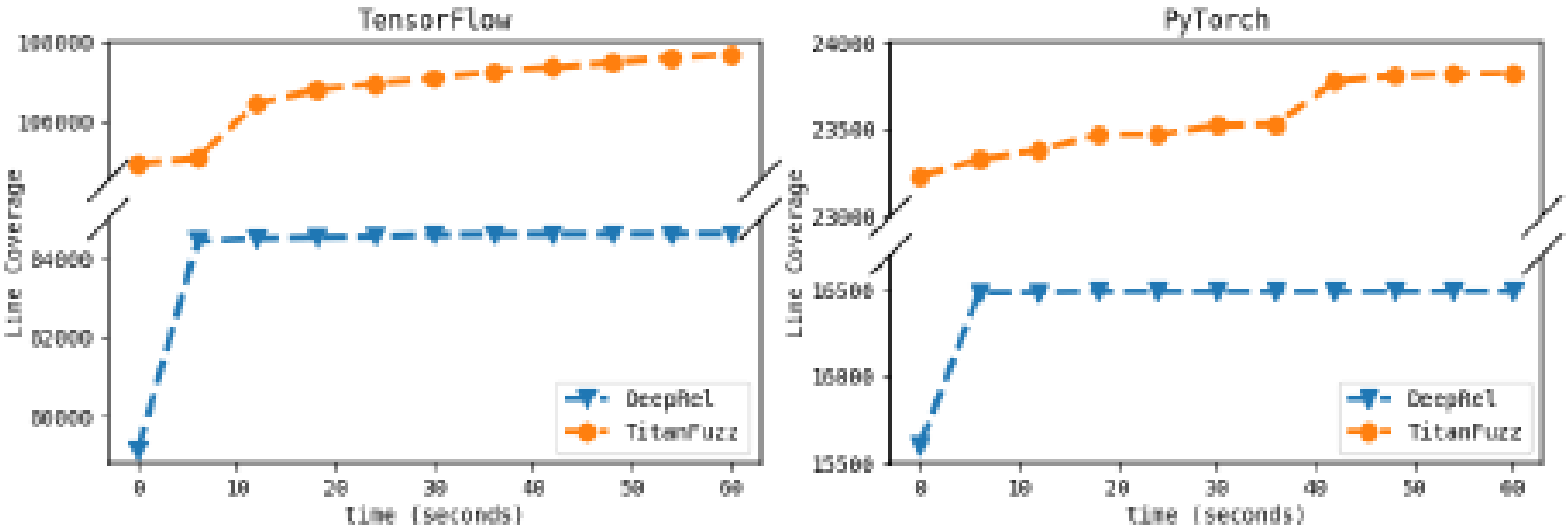


Figure 7: Coverage trend against DeepREL

Coverage vs. time comparison (cont.)

Table 2: Comparison with the best existing techniques

	PyTorch		TensorFlow	
	Coverage	Time	Coverage	Time
DeepREL	15794 (13.91%)	5.1h	82592 (30.65%)	9.3h
Muffin	-	-	79283 (29.42%)	6.8h
TitanFuzz-seed-only (w/ DeepREL APIs)	18447 (16.25%)	3.4h	89048 (33.05%)	4.9h
TitanFuzz-seed-only (w/ all APIs)	22584 (19.89%)	5.1h	103054 (38.35%)	11.9h
TitanFuzz	23823 (20.98%)	9.9h	107685 (39.97%)	21.1h

RQ2: Ablations

- Temperature
- Evolutionary algorithm
  - Mutation operators allowed
  - Fitness function/operator selection
  - InCoder vs. Codex

Temperature

Is this only among valid programs?

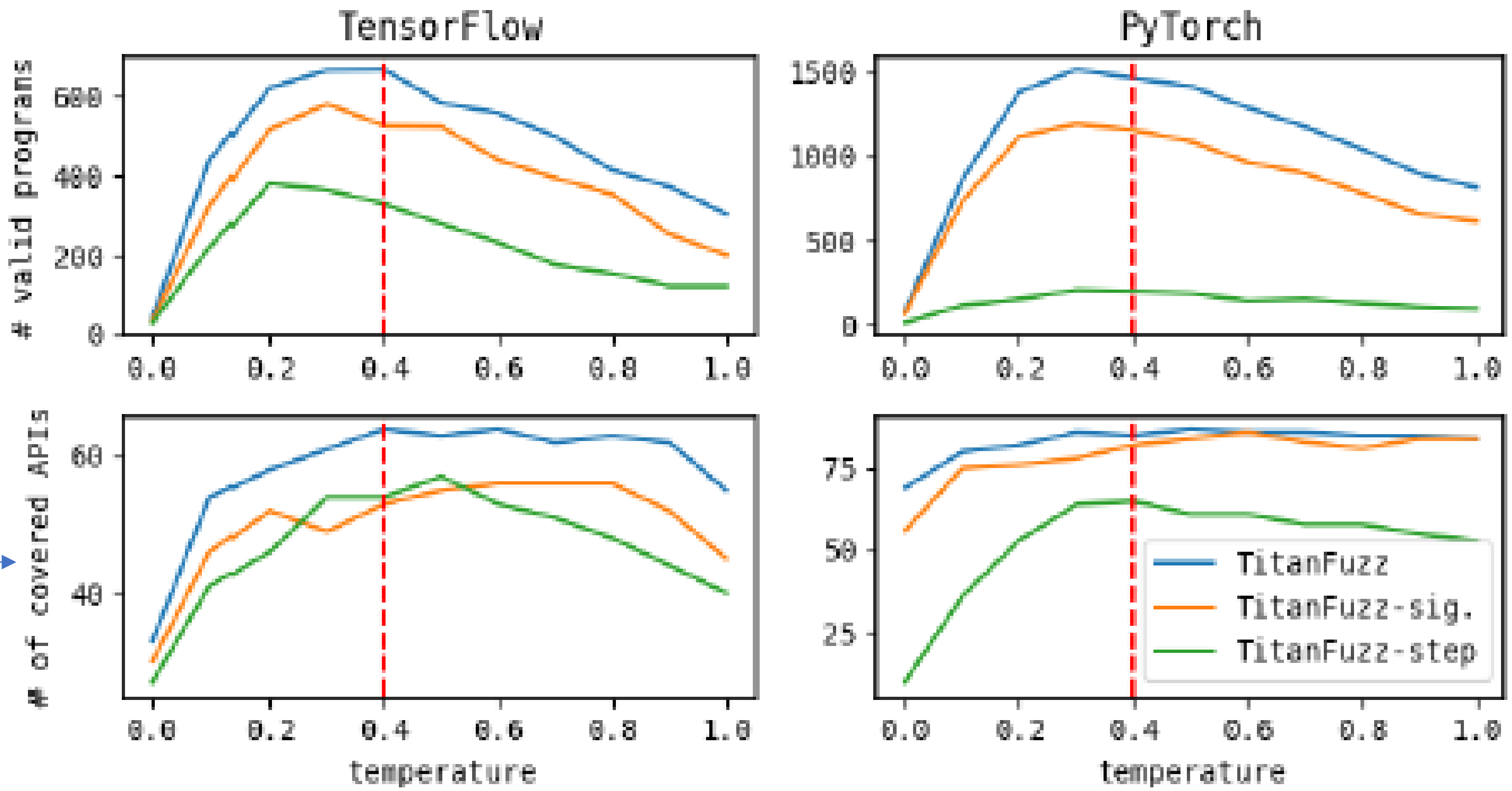


Figure 8: Codex seed generation trend

Ablating operators

- Each seems useful

Table 3: Ablation study of operators

Variants	PyTorch				TensorFlow			
	# Unique Prog.		Coverage		# Unique Prog.		Coverage	
	Valid	All	Valid	All	Valid	All	Valid	All
TitanFuzz	**6969**	**18245**	**17411**	**17957**	**5173**	**16865**	**84447**	**86536**
-Suffix	5770	15813	16709	17691	4642	14501	81145	85294
-Method	6239	16943	16886	17615	3492	12519	83405	85454
-Prefix	6211	17082	17075	17797	3359	12345	83435	85645

Ablating fitness function

Table 4: Ablation study of fitness function

Variants	PyTorch				Tensorflow			
	# Unique Prog.		Coverage		# Unique Prog.		Coverage	
	Valid	All	Valid	All	Valid	All	Valid	All
D+U-R	**6960**	**18245**	17411	17957	**5173**	**16865**	**84447**	**86536**
D+U	5817	15609	**17725**	**18415**	2993	11253	82963	85455
D-R	5872	16916	17229	18046	2876	11861	83563	85599
U-R	6234	17321	16894	17820	4315	15495	84057	86286
Random	7288	20720	16674	17586	3274	13237	83440	85045
Coverage	5098	15300	16715	17617	3210	12880	83030	84194

Ablating operator selection algorithm

Table 5: Evaluation of operator selection algorithms

Library	Algorithm	#Unique programs		Coverage	
		Valid	All	Valid	All
PyTorch	TS	**6960**	18245	**17411**	**17957**
	Random	6185	**18504**	17003	17683
TensorFlow	TS	**5173**	**16865**	**84447**	**86536**
	Random	2612	11816	83238	85469

RQ3: actually finding bugs?

```
input_file = ['https://.../iris_training.csv',
              'https://.../iris_test.csv']
training_dataset = tf.data.experimental.
                   CsvDataset(input_file[0], ..., header=True)
for e in range(10):
    # The following operation is causing Check Fail
    training_dataset = training_dataset.shuffle(1000).repeat().batch(512)
```

Target API: tf.data.experimental.CsvDataset
Catch: Check failed: 0 <= new_num_elements ... (core dumped)

a)

```
x = torch.randn(10, 10).log() # x contains NaN
y = torch.histc(x, bins=10, min=0, max=1)
# On CPU: [48, ...] counts all NaN
# On GPU: [2,  ...] does not count any NaN
```

High Priority

Target API: torch.histc
Catch: Inconsistency between GPU and CPU

b)

```
indices = tf.constant([1, 2, 3, 4])
data = [1.0, 2.0, 3.0, 4.0]
output = tf.raw_ops.ParallelDynamicStitch(indices=indices, data=data)
# On CPU: [7.6904807, ...] out-of-bound read
# On GPU: [0, ...]
```

Security Vulnerability

Target API: tf.raw_ops.ParallelDynamicStitch
Catch: Inconsistency between GPU and CPU

c)

```
X = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.int32)
Z = tf.bitwise.right_shift(X, -1)
# On CPU: [[1, 2, 3], [4, 5, 6]]
# On GPU: [[0, 0, 0], [0, 0, 0]]
```

Implementation -defined

Target API: tf.bitwise.right_shift
Catch: Inconsistency between GPU and CPU

d)

Figure 9: Bugs detected by TitanFuzz

- Their fuzzer tries wacky cases!
  - Obscure APIs wouldn't be used by model-based fuzzers
  - More complex Python scaffolds
- 9/53 confirmed bugs could be found by API-level fuzzing, none by model-level

Augmenting Greybox Fuzzing with Generative AI

Jie Hu, Qian Zhang, Heng Yin

UC Riverside

arXiV

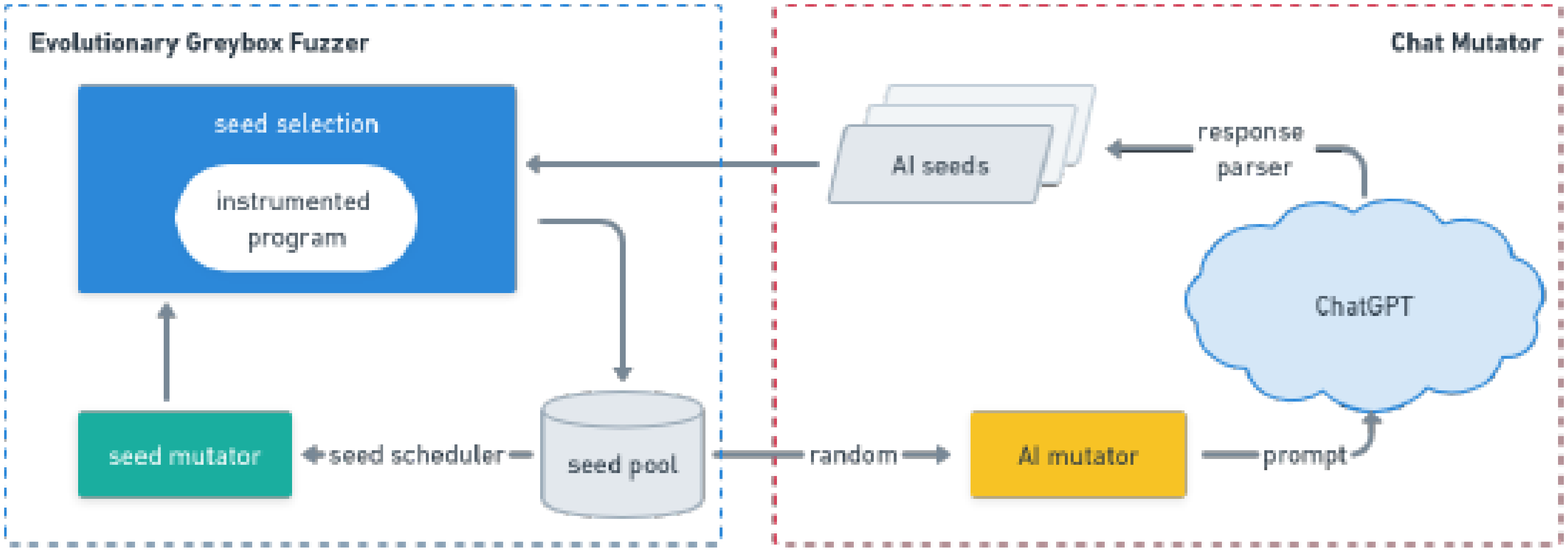


Figure 1: CHATFUZZ Overview

Hyper-parameters

- Model endpoint
- Prompt style
- max_tokens
- n (# completions)
- Temperature

Table 2: Prompt Templates

Config	Prompt Info.		Model	Prompt Template
	Sample Input	**Format**		
AI	✓	✓	CT	System: "You are a <format>file generator" User: "Here is an example <format>file, generate another one." + <sample input>
			CP	<sample input>+ "And here is another <format>file like above: "
AI_noINPUT	✗	✓	CT	System: "You are a <format>file generator" User: "Generate a <format>file."
			CP	"Here is a <format>file: "
AI_noFORM	✓	✗	CT	System: "You are a file generator" User: "Here is an example file, generate another one." + <sample input>
			CP	<sample input>+ "And here is another one like above: "

Max_tokens

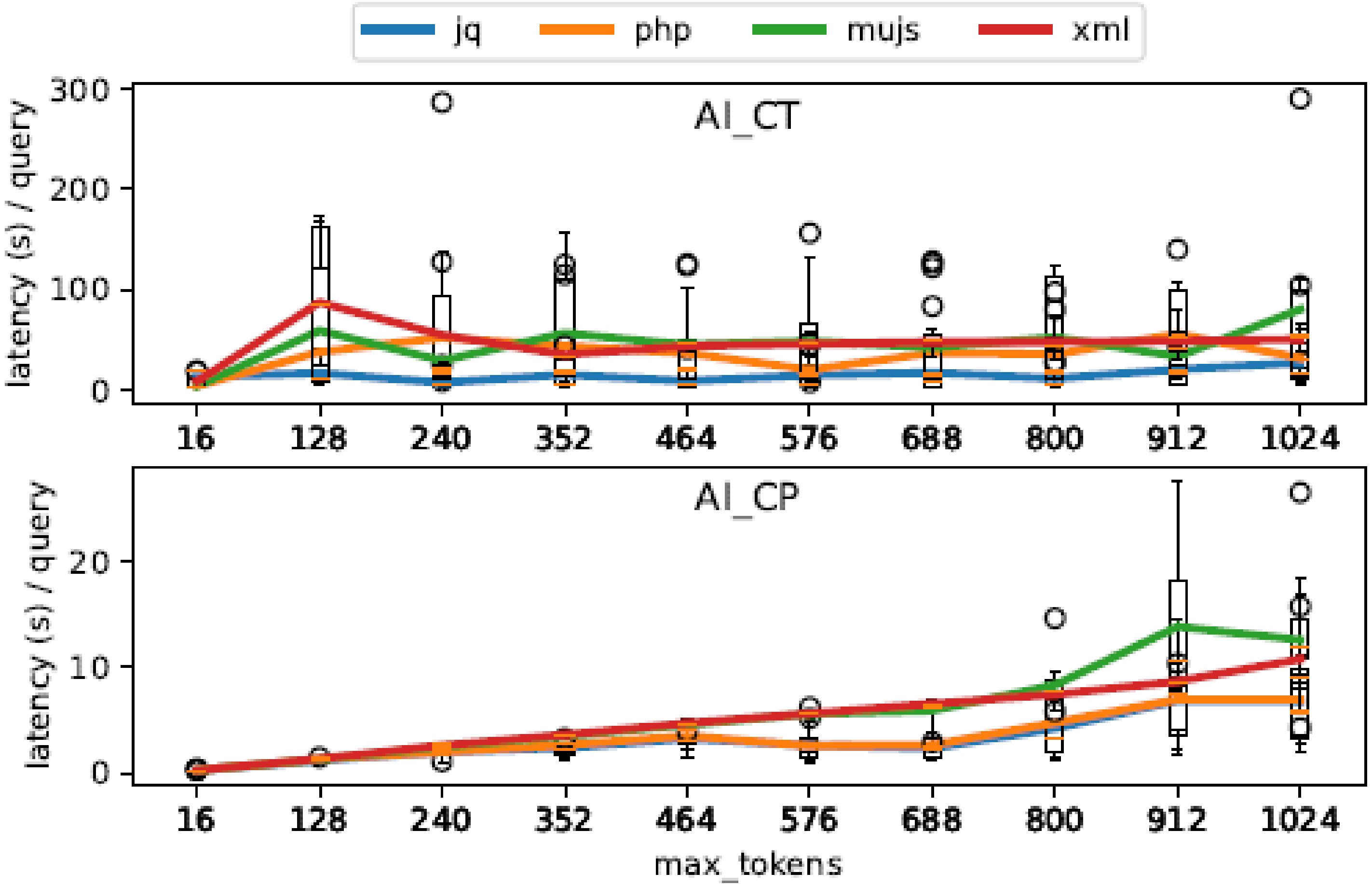


Figure 2: Model Latency and max_tokens

Temperature

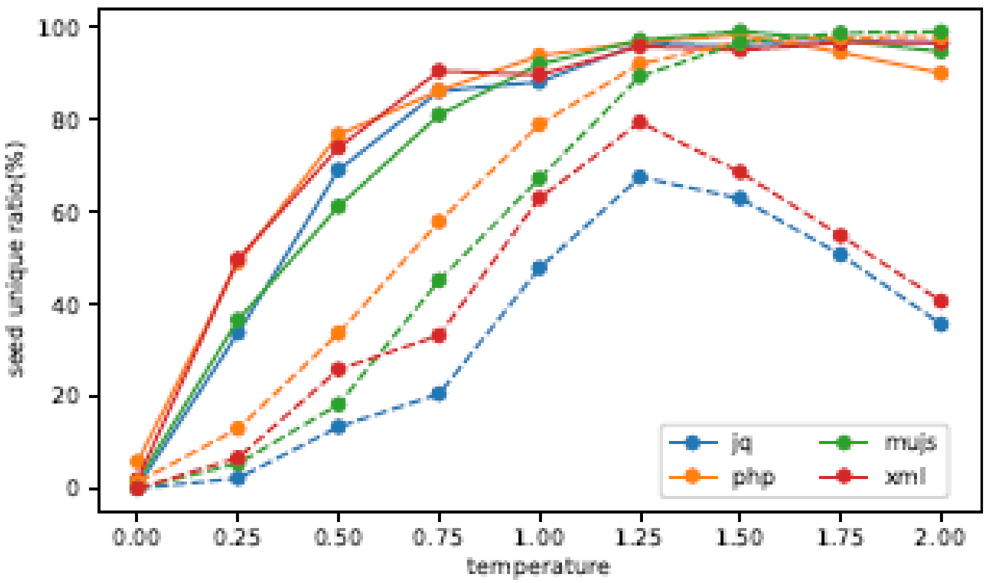


Figure 5: Seed unique ratio of all generated seeds. Note that the result of *AI_CT* is in a solid line while that of *AI_CP* is in a dashed line.

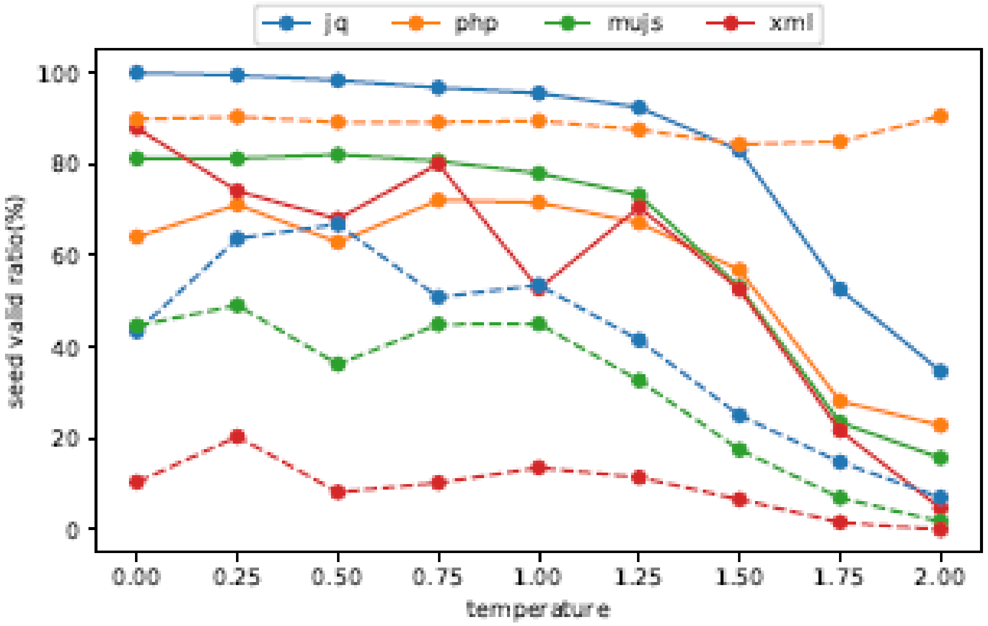


Figure 6: Seed valid ratio of all generated seeds. Note that result of *AI_CT* is in solid line while that of *AI_CP* is in dash line.

Temperature (cont.)

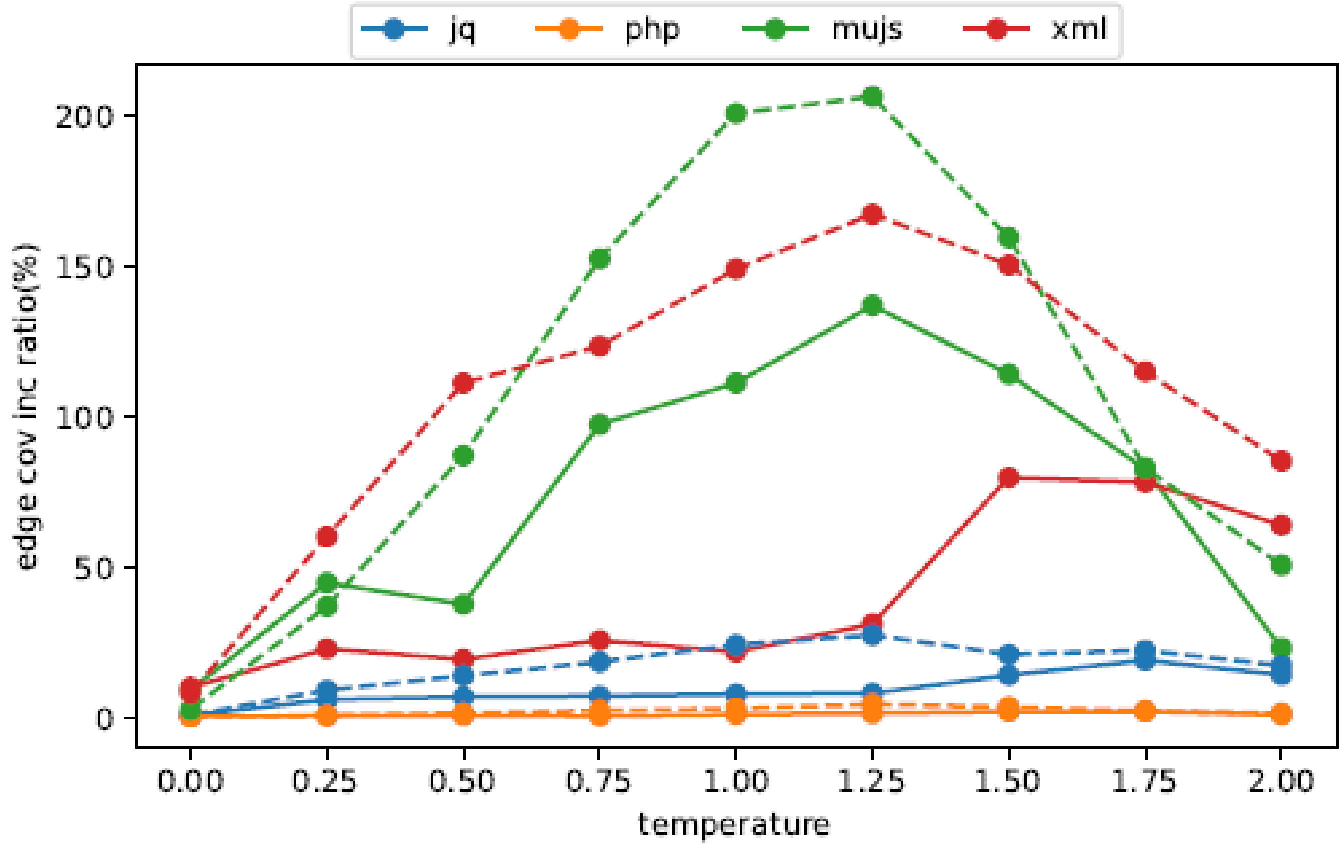


Figure 7: Code Coverage Improvement Over Initial Corpus

Something is funky in the study of prompt ablation

Table 5: Prompt Ablation Study

Program	CT endpoint					CP endpoint				
	AI	AI_noFORM		AI_noINPUT		AI	AI_noFORM		AI_noINPUT	
	cov	cov	vs. AI	cov	vs. AI	cov	cov	vs. AI	cov	vs. AI
jq	3837	4015	+4.64%	3555	-7.35%	4043	4015	-0.69%	3555	-12.07%
php	18995	19609	+3.23%	18364	-3.32%	20021	19609	-2.06%	18364	-8.28%
mujs	11233	10924	-2.75%	6819	-39.29%	13763	10924	-20.63%	6819	-50.45%
xml	7217	6988	-3.17%	6209	-13.97%	7832	6988	-10.78%	6209	-20.72%
Average			**+0.49%**		**-15.98%**			**-8.54%**		**-22.88%**

Prompt ablation (cont.)

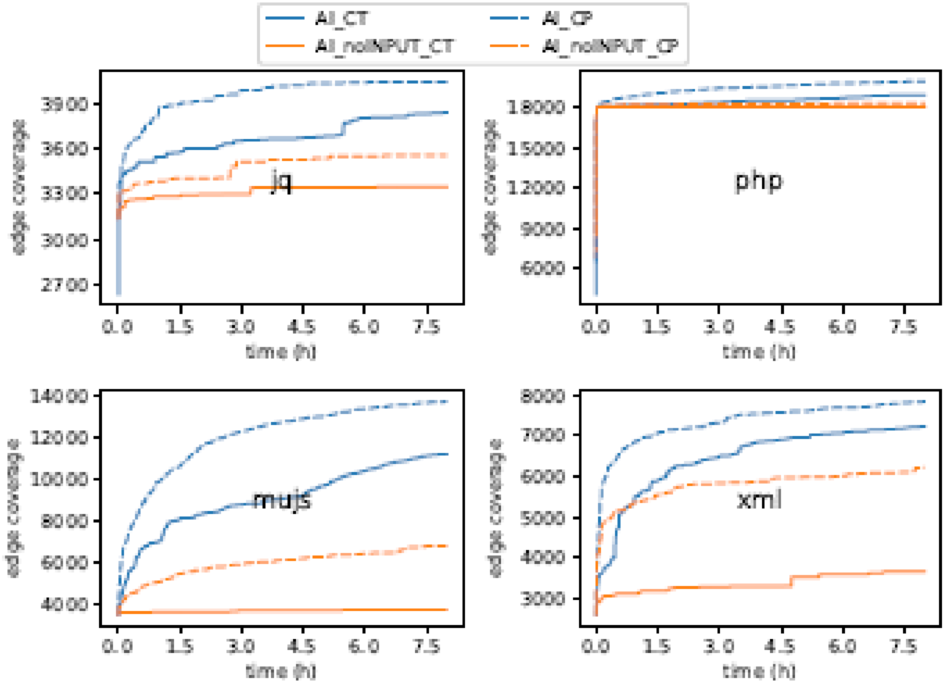


Figure 8: *AI* vs. *AI_noINPUT*

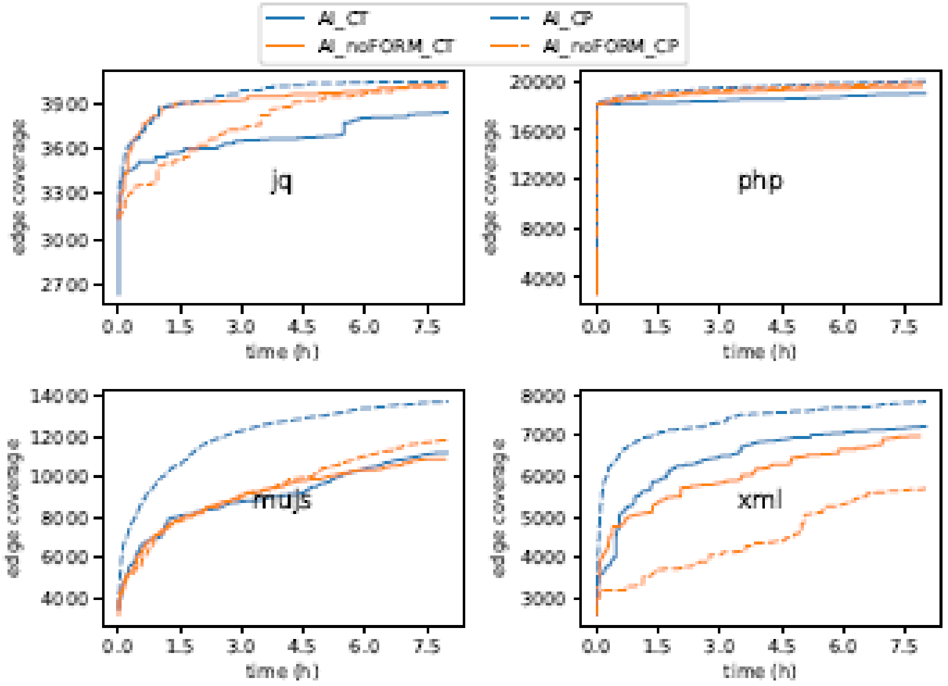


Figure 9: *AI* vs. *AI_noFORM*

Evaluated fuzzers

Table 6: Baselines

Baseline	Model Endpoint	Format Agnostic?
AFL++	-	-
CHATFUZZ	CP	✗
CHATFUZZ-F	CP	✓
CHATFUZZ-C	CT	✗
CHATFUZZ-CF	CT	✓

Evaluation setting

Table 7: Benchmarks

Type	Program	Version	Input Format
data	jq	jq-1.5	json
	php	php-fuzz-parser_0dbedb	PHP
	xml	libxml2-v2.9.2	XML
	jsoncpp_fuzzer	jsoncpp	json
code	mujs	mujs-1.0.2	js
	ossfuzz	sqlite3_c78cbf2	SQL
	cflow	cflow-1.6	C
	lua	lua_dbdc74d	lua
text	curl_fuzzer_http	curl_fuzzer_9a48d43	HTTP response
	openssl_x509	openssl-3.0.7	DER certificate
	base64	LAVA-M	.b64 file
	md5sum	LAVA-M	md5 checksum

Questions for both of them

- What about using LLMs to write generators/mutators?
- What about other baselines?
  - Property-based testing (more manual effort)
  - Leo mentioned Driller, VUzzer
- Can’t one enforce some output constraints at generation time? (with non-API models anyway)
  - Microsoft Guidance, get inspired by JSONFormer, etc.