Northeastern University
Khoury College of
Computer Sciences

MS Thesis Approval

Thesis Title: Leveraging Type Annotations For Effective Fuzzing of Python Programs

Author: Samuel Xifaras

MS Thesis Approved as an elective towards the Master of Science in Computer Science.

____________________	8/14/2024
Thesis Advisor	*Date*
William Robertson	8/14/2024
Thesis Reader	*Date*
____________________	____________
Thesis Reader	*Date*
____________________	____________
Thesis Reader	*Date*
____________________	____________
Thesis Reader	*Date*

KHOURY COLLEGE APPROVAL:

____________________	12/10/2024
Program Director	*Date*
____________________	____________
Program Director (if applicable)	*Date*
____________________	12/10/2024
Associate Dean for Graduate Programs	*Date*

COPY RECEIVED BY GRADUATE STUDENT SERVICES:

____________________	12/11/24
Recipient's Signature	*Date*

Distribution: *Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI Website.*

NORTHEASTERN UNIVERSITY

MASTER'S THESIS

Leveraging Type Annotations for Effective Fuzzing of Python Programs

Author:
Samuel Xifaras

Advisor:
Dr. Panagiotis Manolios

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Computer Science

in the

Khoury College of Computer Sciences

December 6, 2024

Declaration of Authorship

I, Samuel Xifaras, declare that this thesis titled, "Leveraging Type Annotations for Effective Fuzzing of Python Programs" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: December 6, 2024

Abstract

Samuel Xifaras

Leveraging Type Annotations for Effective Fuzzing of Python Programs

Python is among the most popular programming languages, and it powers software systems across diverse domains. Ensuring Python-language systems can grow sustainably is a pressing challenge, complicated by Python's lack of static typing. Optional type annotations have been added to Python to address this, and static type checkers have emerged to offer some compile-time guarantees, but these tools often emit false positives, and there is evidence that developers have been slow to adopt them. To complement these existing Python code analysis tools, we propose and implement a novel approach, TYPE HINT FUZZING. TYPE HINT FUZZING leverages type annotations to automatically perform coverage-guided mutational fuzzing, an extremely successful software testing technique, in a target codebase. Because TYPE HINT FUZZING executes the code, every issue found has an associated counterexample that triggers it, precluding false positives.

Three key developments enable the TYPE HINT FUZZING tool's implementation: 1) a model of Python's type system in the ACL2 Sedan theorem prover, with custom enumerators for primitive types 2) a type checker plugin that extracts type annotation information, and 3) a custom input encoding that is amenable to mutational fuzzing while preserving decodability. In this thesis, we elaborate on these developments and perform a rigorous evaluation to assess the performance of the tool. In our evaluation, we first compare the custom encoding to *pickle*, the *de facto* object serializer in Python, and find that the custom encoding is unequivocally better for fuzzing. Secondly, we explore various tool configurations to understand the effect of hyperparameters on fuzzing performance. We draw some conclusions, and find that more work is needed to fully understand these effects. Finally, we report issues that the tool found in the set of target repositories. Three bug reports have been accepted as legitimate by open-source maintainers, and two of them have been patched. We conclude with extensive discussion of the results and avenues of future work to build on this promising approach.

Acknowledgements

Completing this thesis is the hardest thing I have done yet in my life. I have been challenged and stretched in incredible ways, and I have learned a great deal about how to (and how not to) build good software, how to execute the scientific method, and how to write clearly and concisely. I am still no expert, however. I have a lot to learn. Above all, this experience has left me utterly humbled by the amount of energy and persistence it takes to produce high-quality, impactful research. I am astounded by the fact that I have produced a written treatment in excess of 100 pages, and yet I feel that there is so much left unanswered and uncovered. I am exceedingly grateful for this experience, and for all those who made it enjoyable and achievable for me.

First and foremost, I would like to thank my thesis advisor, Pete Manolios. Pete, without your steadfast support, great ideas, and forthrightness in giving feedback, I would not have been able to develop into a researcher capable of completing this work, so thank you. It has truly been a pleasure getting to know you over the past two years. I am consistently blown away by how much knowledge you have stored in your mind, and I have truly relished our fun and dynamic conversations in our weekly group meetings and other encounters. Thank you also for being attentive to my personal needs as well–when I raised concerns about having taken on too much scope for the project given the amount of time we had, you understood and worked with me to make adjustments.

Next, I would like to thank the other reader on the committee, William Robertson. Thank you, Wil, for agreeing to be a committee member on this project, and for giving your time to sitting in on the weekly meetings throughout the course of this project. You have been a consistent source of great ideas, feedback, and insights related to fuzzing and other areas of your expertise.

I would also like to thank the other members of the Computer-Aided Reasoning group who participated in the weekly meetings for this project: Drew Walter, Cassidy Waldrip, and Daniel Yu. Drew, thank you for your inquisitive mind, offering your time to provide feedback on this writeup, and for creating the type server that I was able to build on. It has been great getting to know you, and I wish you all the best on your future endeavours. Cassidy, it has been a pleasure getting to know you as well, and being able to collaborate with you on the code for this project while you started working on an extension of it. I have never before authored a piece of software from the ground up and had someone else start to extend it, so that has been an amazing learning experience for me. I wish you the best in your future endeavors as a PhD student and beyond. And Daniel, I would like to thank you also for your help and companionship during this project. The automation scripts you wrote were very helpful, and I hope they provided you with some learning as well. I wish you all the best in your future endeavours in wherever your career may take you. I thank you all again and I hope to stay in touch!

One of the aspects of this experience that made it challenging was the fact that I was doing it part-time alongside a full-time software engineering job at Paperless Parts. I thank my managers William, Ariella, and Jon, for giving me the support and space I needed to succeed in completing this thesis while employed. Thank you!

Lastly, but certainly not least, I would like to thank those closest to me. Mom, Dad, Paulina, and Abby, thank you all so much for your unwavering love and support throughout this endeavor, and forgive me for all the time I could not spend with you; I hope to make it up to you now. I love you all.

Thank you again to all, named and unnamed, who influenced me and guided me through this process. Although this thesis credits me as the sole author, this was a team effort in many ways.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Glossary

corpus is a set of valid inputs to the system-under-test when fuzzing, that the fuzzer uses as starting points for mutation into new inputs (pl. **corpora**). 22, 68, 73, 74, 80, 83, 84, 89, 91, 95, 98–101

function candidate is a function in a codebase under test whose types are fully understood, and is therefore a "candidate" for fuzzing with the tool. 13, 14, 37, 39, 40, 48, 72, 80, 91, 99, 101, 104, 112, 142, 145, 146, 150

function contract is the property that a function will return a value of its return type when arguments of the correct parameter types are supplied. 13, 43

fuzzing point (also, **point**) is an input-output pair observed during a fuzzing campaign. 48, 68, 69

harness (also, **fuzzing harness**) is the piece of code that takes a raw bitvector that is produced by a fuzzer and uses it to execute meaningful behavior in the code under test. 3, 15, 21, 22, 42

Acronyms

CAK Coverage At Knee. x, xi, xvi, 67, 71, 73, 74, 91, 98, 99, 101, 105, 106, 109–112, 142, 145

CUT code under test. 13, 21, 69

FUT function under test. 22, 44, 48, 79, 153

RMST Restricted Mean Survival Time. xi, xvi, 77, 99, 106, 115, 118–120

TTK Time-to-Knee. x, xi, xvi, 67, 71, 73, 74, 91, 98, 99, 101, 105, 106, 109–112, 142, 145

1 Introduction

In the ever-changing landscape of software, Python has managed to emerge as one of the most popular languages in the world, named second only after Javascript [126]. Python's meteoric rise in popularity was, according to many, driven by its versatility, ease of use, and the growth of an active and supportive community [27, 112]. Python has become the *de facto* language for machine learning, and it is now used in many SaaS products as well, due to the emergence of web frameworks like Django and Flask [27, 112].

Given that Python now powers so much of our world, extra care should be taken to ensure that Python-language software is easy to understand, easy to modify, stable, and robust against untrusted inputs. The degree of our dependence on Python software, and therefore the importance of investing in its safety and security, is underscored by incidents like the discovery of a platform-specific variation in behavior in one of Python's standard library calls, which cast doubt on the results of dozens of research papers [5]. To achieve the goals of ease of understanding and ease of modification, the Python community has adopted a system of optional type annotations (often called *type hints*) [108]. Static typing in code has a range of benefits, from improved modularity and earlier catching of simple bugs [117], to reduced development time for certain categories of tasks [53]. A key difference, however, between the system of type hints in Python and the strong typed-ness of languages like Java and C# is that types in Python are not enforced statically. The type annotations are essentially ignored when the code is run, meaning that at runtime, values of completely different types than those which are indicated by the annotations could be flowing through one's code. Static type checkers, and other static analysis tools, such as mypy [125] and pytype [99] have emerged as a solution to this problem. Although these tools do often help developers actualize the intent of including type annotations in their code, they fall short, mostly through no fault of their own: Python's semantics are very complex, and there is no evident consensus on what the type system of Python actually is. Rak-amnouykit et al. [101] identify that *mypy* and *pytype* essentially represent two different type systems, and they catch "largely disjoint sets of errors." Yang, Milanova, and Hirzel [143] highlight the prevalence of various semantically complicated "dynamic" Python features that confound static analysis. Perhaps as a consequence of these factors, these static tools are known to emit many false positive errors–i.e., errors that do not indicate actual runtime problems–which detract from their usefulness to developers [101]. According to Rak-amnouykit et al. [101], the proportions of false positives emitted by *mypy* and *pytype* may be as high as 49% and 44%, respectively.

To achieve the second two of the aforementioned requirements, stability and robustness, strong software testing techniques are needed. Static typing discipline and static analysis tooling can help here as well, but given the complex semantics of Python and ensuing difficulty of static analysis, dynamic approaches are promising because the source of truth is the reference interpreter on which all code already runs. Unit testing is common practice in software engineering, but there is clear evidence that the overhead in creating unit tests that deeply exercise code deters developers [147]. Fuzzing, on the other hand, is an automated testing approach dating back to the 1980s [39], and it has been wildly successful in finding many bugs, particularly in C/C++ programs, which tend to be especially buggy due to the burden of manual memory management on the programmer (see, for instance, the list of trophies in [74]).

At a high level, fuzzing is the process of bombarding a system with random inputs and seeing how it responds. In practice, there are many variants of fuzzing that make different trade-offs between randomness and structure [146, 145, 139]. Fuzzing in Python, to our knowledge, has only recently started to become explored [72, 73], enabled primarily by the development of Atheris [45], a Python-language fuzzer built on libfuzzer [74]. The challenge with using Atheris on Python code, however, is that the tester must understand the code they wish to test well enough to write a *harness* (i.e. a small program that takes a stream of bytes as input and executes the code-under-test in some meaningful way) for it. The challenge of developing harnesses is not trivial, and there is work on automating this process for C programs [61].

In light of all this, we identify an opportunity to leverage type annotations in Python code, which are steadily becoming more popular [28], to automatically fuzz a codebase-under-test at various entry points for which there is type annotation coverage. With this approach, we solve the problem of harness generation, and we enable more robust testing of software with minimal added human effort (i.e. the effort required to annotate code, which, while not insignificant, is likely far less than the effort required to write comprehensive unit tests, especially if a project has checks in place to ensure that newly added code is annotated). Because fuzzing tests a wider range of inputs than a typical unit test suite might, we can produce crashes that otherwise might have gone unnoticed. Furthermore, we can also perform dynamic type checking with this approach, and ensure that type annotations on the output of functions are actually correct at runtime. Above all, there is by definition no possibility of false positives since we are running the code; if a type mismatch is detected, we would easily be able to produce a counterexample. This makes what we report immediately actionable and interesting for developers.

We call this approach TYPE HINT FUZZING. In this thesis, we introduce this approach in greater detail and describe its implementation. We also include a detailed evaluation of the tool's effectiveness, from which we also conclude how to optimally configure the tool for best results. We also report several real-world bugs found by the tool.

The main contributions of this work are summarized as follows:

- Modular and extensible design and implementation of a TYPE HINT FUZZING tool (Chapter 5)
- Design and implementation of a custom input encoding (Chapter 6)
- Evaluation of the effectiveness of the custom input encoding (Chapter 7)
- Exploration of the optimal configuration of the tool (Chapter 7)
- Issue reports for bugs found by the tool submitted back to the open-source community (Chapter 7)
- Case study of triaging of an issue found by the tool with generative AI (Chapter 7)
- Recommendations for enhancements to the Python type hint system (Chapter 8)

The remainder of this thesis is organized as follows: Chapter 2 covers background information and related work; Chapters 3 and 4 cover the core technologies the tool integrates with, ACL2s and libFuzzer, respectively; Chapter 5 covers the architecture and implementation of the tool; Chapter 6 covers the design and implementation of the custom input encoding; Chapter 7 contains the evaluation of the three research questions that we pose; Chapter 8 includes a discussion of the results, threats to their validity, recommendations for enhancements to the Python type hint system in light of what we observed, and directions for future work; finally, we conclude in Chapter 9.

2 Background and Motivation

In this chapter, we cover background information pertaining to Python, its system of optional type annotations, and static type checkers that are on the market for checking annotations. We then reveal deficiencies in commonly used static type checkers for Python, and motivate our work in light of these. Finally, we include a walk through related work, illuminating how our work fits into the broader landscape.

2.1 Python and its Type Annotations

Python has become an industry-standard programming language and is the second most used language, behind JavaScript [126]. Although both of these languages are considered dynamic because variables can take values of any type at runtime, they both have some support for optional annotations, which allow types to be notated in the code. Python had type annotations officially added to the language's standard in 2014 [108], while JavaScript's support is typically provided by a third party, TypeScript [129], and there is an open proposal to add similar type annotations to the language itself [121]. The advantages of static typing include improved modularity and catching simple bugs earlier [117], and there is evidence that it can reduce development time for certain categories of tasks [53]. Type annotations also serve as a form of in-code documentation, aiding in code readability.

Despite these benefits, type annotations in Python remain somewhat unpopular.

A recent empirical study found that less than 10% of all annotatable Python code elements are currently annotated [28]. However, the same study also found that there is a clear upward trend in growth in type annotations across open source projects in the Python ecosystem, increasing from just 15 annotations per 1,000 lines of code at the end of 2017 to 50.1 annotations per 1,000 lines of code in 2021 [28]. Di Grazia and Pradel [28] also identify three clear type annotation usage patterns across the Python open-source projects they studied: "regular," "sprinter," and "occasional." "Regular" projects have type annotations committed steadily alongside new code in the repository. "Sprinter" projects have many commits without type annotations, and there will be type annotation "sprints," consisting of commits that consist mostly or entirely of type annotations that backfill existing code. "Occasional" projects have a small number of annotations across a limited number of files. Projects which are classified as regular type annotators have a greater number of contributors, on average, while occasional type annotation users have far fewer contributors [28]. This suggests that adding type annotations is a marker of the maturity of a project.

Type annotations may seem to have unequivocal benefits, but there are a number of challenges posed by the Python language. Python has many features, many of which complicate its semantics. These features therefore also complicate static analysis [143]. Furthermore, static analysis tools disagree on how Python's type hints should be used and checked. *mypy* [125], which is essentially an implementation of PEP-484 [108], the original type hint standard, assumes a strict notion of static typing discipline, similar to what one might encounter in Java, while another tool, *pytype* offers a looser interpretation [101]. Beyond differences in interpretation of what the type judgement rules are, there is also the reality that type checking in Python is not decidable [109]. Correct and useful static analysis of Python code is difficult under these conditions, and indeed, in the following section, we cover instances where static type checking in

Python fails to produce results that would be desirable for developers.

2.2 Static Type Checkers

With the advent of type annotations in Python, several static type checkers have been created to enforce the static typing discipline that these annotations specify at compile-time. In spite of advances in tooling support for Python's type annotations, previous work has found that *mypy* reports errors in a majority of studied repositories [28, 101]. However, this may not necessarily indicate poor code quality. Rak-amnouykit et al. [101] offer two hypotheses for this phenomenon: first, that *mypy*'s type system generates false-positive warnings more than it catches actual runtime errors which discourages the use of the tool, and second, that *mypy* is not the tool of choice for developers, and they are either not using *mypy* or using other tools with different behaviors. In support of the first hypothesis, they found that 49% of the errors *mypy* generated were false positives, and they also found that the false positive rate of a less strict tool, *pytype*, was not much lower [101]. It is clear that developers are often not integrating *mypy* into their workflows, since these errors were detected statically at the current versions of the studied repositories. This suggests that the prevalence of false positive errors may impede adoption of these tools, since developers may not be motivated to make the necessary code changes to resolve the errors. This is rather speculative, however, and there is no discussion in the literature of what developers actually expect from type checkers in Python, but considering the point of view of a developer, and in light of the evidence from Rak-amnouykit et al. [101], it may be reasonable to expect 1) low false-positive rate, 2) clear, actionable error messages, and 3) ability to infer obvious facts. As we have discussed, and as we will see in the examples that follow in this section, static type checkers on the market for Python do not meet these expectations.

In addition to the false positives generated by *mypy* and *pytype*, another deterrent to using these tools are their semantics, which may be seen as overly strict because they do not account for the use of semantically complex dynamic features that allow programmers to perform object introspection and object modification on the fly. These features are not uncommon in Python code. Yang, Milanova, and Hirzel [143] study the usage patterns of complex Python features, and the authors find that, in a set of hundreds of thousands of repositories from 2021, 16% of files use some dynamic feature (getattr, setattr, hasattr, delattr, eval, exec) of Python. Although the authors find that the majority of dynamic feature occurrences are uses of getattr, setattr still makes almost 200,000 appearances, and as shown in Listing 2.1, both *mypy* and *pytype* fail to account for attributes set through setattr. The code shown in the listing runs successfully on Python 3.8, and the false positive type error was reported with the latest version of mypy at the time of this writing (version 1.8.0), as well as the latest version of pytype at the time of this writing (version 2024.02.27).

```python
class A:
  def __init__(self, x: int, y: str):
    self.x = x
    self.y = y

def test(a: A):
  setattr(a, 'y', 6)
  return a.x + a.y

a = A(5, 'str')

test(a)
```

LISTING 2.1: Motivating example: A snippet of code where *mypy* and *pytype* both report an error when there is no error at runtime.

Although the example in Listing 2.1 is technically a false positive, there is discussion to be had on whether *mypy* reporting an error in this case is unreasonable. The usage of `setattr` in this example seems to be violating the original type annotation of the `y` field in class A. However, from the point of view of a programmer, the error that is raised, which is that the expression `a.x + a.y` is incompatible with the types of `a.x` and `a.y`, may be unclear (the real problem is the perverse use of `setattr`), and demonstrates these tools' inability to infer seemingly obvious facts. In reality, such facts are not obvious, especially in Python. Consider the fact that in Python, function symbols can be overridden arbitrarily at any point. How does a static analysis tool know with certainty, without executing the code, that `setattr` actually does what it is assumed to do by default? In other static analysis settings, such as "linting" in JavaScript, tools on the market make more conservative tradeoffs to avoid harming the user experience with false positives [43].

Hypothetical examples aside, there are plenty of real-world examples of false positives in *mypy*. For instance, issue #3004 on *mypy*'s issue tracker, which remains open at the time of this writing, is an instance of *mypy*'s type checking preventing Python developers from fully leveraging novel language features [115]. Listing 2.2 contains an excerpt from one of the comments on this issue, illustrating a legitimate use case where having different types for a property's getter and the input of a property's setter would be useful. The benefit of this is, as the commenter explains it, that class `Foo` can expose a general interface for setting the `foo` property that does not make the API user have to do extra work to conform to a specific underlying representation of `_foo` while also providing additional guarantees when accessing `foo` (i.e. that it is a set) [115]. Further comments down the chain show that many developers in the community have run into the same issue, including the *mypy* team itself. This is an example of how tools like `mypy` make unrealistic assumptions about code, to the detriment of many users.

```
from typing import Set, Iterable

class Foo:
  def __init__(self) -> None:
    self._foo = set()  # type: Set[int]

  @property
  def foo(self) -> Set[int]:
    return self._foo

  @foo.setter
  def foo(self, v: Iterable[int]) -> None:
    self._foo = set(v)

Foo().foo = [1, 2, 3]
```

LISTING 2.2: Motivating example: A false positive report taken directly from mypy's issue tracker (#3004). *mypy* currently does not support a property's setter having a different type than the property itself.

```
from typing import List

def sum_list(x: List[int]) -> int:
  first = x[0]
  rest = x[1:]
  return first + sum_list(rest)

print(sum_list([]))
```

LISTING 2.3: Motivating example: A snippet of code where *mypy* and *pytype* both report an error when there is no error at runtime.

There are also a number of simple type errors that are commonly encountered during Python development that these tools do not catch. These are *false negatives*, which are particularly alarming, as this means that runtime crashes can go unnoticed until they hit a production environment if proper test coverage and other measures are not in place. Listing 2.3 shows an example of a false negative: a list indexing error caused by the implicit expectation that the given list is non-empty. If code documentation is poor, then a consumer of this API would have to read through the body of the function in order to understand the intended type of `x` (i.e. a *non-empty* list of integers). The PEP-484 standard does not provide for a way to express a non-empty list [108]. If code analysis tools were able to report such issues, developers would likely be very interested in remediating them because they can have catastrophic consequences if the errors occur in production settings. In fact, we have observed many instances where Python developers make up for the deficiencies of the Python type system and its tooling by using the `assert` keyword to enforce such additional properties of types. Listing 2.4 contains an explicit use of `assert` in this way. This strategy enables potentially unexpected denial-of-service, however, as an `assert` statement will cause a runtime crash if the assertion is ever violated. This is especially problematic in light of the discussion of deficiencies of static type checkers: if subtly invalid data is flowing at runtime through complex business logic that passed type checking, then it is perfectly reasonable to expect violations of these assertions in certain edge cases. We expand more on this in the Discussion, where we study cases in the selected repositories where developers used `asserts` in this way and propose a set of enhancements to the Python type hint specification that programmers and static analysis tools can use to strengthen compile-time guarantees about Python code.

```
def as_required_block(
  self,
  stmts: list[ast3.stmt],
  *,
  con_strip: bool = False,
  is_coroutine: bool = False,
) -> Block:
  assert stmts  # must be non-empty
  b = Block(
    self.fix_function_overloads(
      self.translate_stmt_list(
        stmts,
        con_strip=con_strip,
        is_coroutine=is_coroutine,
      )
    )
  )
  self.set_block_lines(b, stmts)
  return b
```

LISTING 2.4: Example, excerpted from mypy, of a developer explicitly using an `assert` statement to enforce an additional constraint of the parameter type that is not specified in the annotation.

The story is not all bad, however. Previous work by Khan et al. [69] has shown that mypy does detect some defects in Python code that was unnanotated and had been found and fixed by other means. Since mypy does not check untyped definitions by default, the authors manually annotated the parts of the code which contained each defect in order to test whether mypy could indeed detect it.

The strengths and weaknesses of static type checkers highlight the need for a testing tool in the Python ecosystem that *complements* static type checkers–one that allows type checkers to continue playing to their strengths, while making up for their deficiencies.

2.3 Problem and Solution

In light of the issues affecting static type checking in Python, we identify the following problems:

- P1: Static analysis in Python is *hard*, due to lack of required static typing, complex semantics , and programmers' liberal use of semantically complicated language features.
- P2: Users of type annotations are not getting as much out of them as they should for the effort they are putting in. Current tools beleaguer them with false positives that are uninteresting and false negatives that are dangerous.

To solve these problems, we propose TYPE HINT FUZZING. We base our approach on fuzzing because it is dynamic (i.e. it executes code), and it has a rich history of being highly effective at finding real-world bugs [74, 139, 138, 6]. Since fuzzing executes the code under test (CUT), we need not worry about the complex of semantics of Python; they are provided by the implementation of the language. This solves P1. To facilitate fuzzing, TYPE HINT FUZZING utilizes type annotation information from the CUT to automatically identify functions that are fuzzable. A function is fuzzable if and only if all of its parameter types and return type are understood under a model of the Python type system that we construct. When the set of fuzzable functions, which we call the *function candidates*, is obtained, the tool automatically fuzzes the functions using well-formed inputs according to their parameter type annotations. During fuzzing, the tool collects information about any crashes and *function contract violations* that are detected. The "function contract" is the property that a function should return a value of the type specified by its return type annotation when correctly typed parameters are given. In this iteration of the implementation, we detect a function contract violation if a function returns a value that does not match its return type annotation. P2 is solved by this automatic function candidate discovery based on type annotations and the information we collect about fuzzed functions; developers who annotate their code reap the value provided by TYPE HINT FUZZING with minimal manual effort.

To implement this tool, we use Atheris [93], a Python-language greybox mutational fuzzer implemented on top of libFuzzer [74]. More information on Atheris and libFuzzer is given in Chapter 4. The model of the Python type system and its implementation are explained in Sections 5.1.1 and 5.2.1. To implement the model of the Python type system, we use the ACL2 Sedan, ACL2s [30, 13] and we implement a client-server architecture to interface with the ACL2s model before and during the fuzzing campaign. More background on ACL2s, and how we utilize it, is given in Chapter 3. In Chapter 5, we elaborate further on the design and implementation details of the tool.

2.4 Related Work

In this section, we take a brief walk over the literature that is related to the background concepts that motivate this research. Our approach depends on other third-party tools, and these will be introduced in subsequent chapters. Each of these chapters will have its own section for related work, and the related work will not be repeated here.

To the best of our knowledge, this thesis represents the first application of fuzzing to Python code in this way, but we drew inspiration from many works on fuzzing and static analysis in the literature.

2.4.1 Fuzzing Python Code

PyRTFuzz [72] is a recent paper that utilizes Atheris [93] to fuzz the Python interpreter. This tool utilizes a two-pronged fuzzing approach. The first phase is generational fuzzing, where sample Python programs that exercise the various standard library

APIs are generated. These Python programs are used as fuzzing harnesses in the second phase, mutational fuzzing, which is performed by Atheris. Similar to this work, PyRTFuzz also had to deal with how to properly call functions which expect certain types of values. This led them to devise an encoding to pack values for the various arguments together into a single stream of bytes that can be mutated [72]. Their custom mutation strategy is different from ours, however: instead of allowing Atheris to mutate using its mutation strategies, they have custom methods for generating random examples of the types that they model. In our work, we use ACL2s [30], a theorem prover which is extended with a data-definition framework and example-generation capabilities, to generate these examples (see Chapter 3 for more information). The authors also only model the most common 20 Python types, whereas our type system is fully extensible, and can extend dynamically to include structural types which consist of types that have previously been admitted to the model.

Györgyi, Laki, and Schmid [50] verify P4 code generation for P4 programmable dataplanes by using Atheris to perform differential fuzzing of the source Python simulation code, which gets translated into lower-level P4 code, and the P4 code itself. This is an intriguing application of Atheris [93], but this work does not deal with the Python type system, as they are working with a limited subset of Python that is a high-level simulation language for P4 programmable dataplanes.

PolyFuzz [73] is a paper that presents a framework for fuzzing of multilingual systems that is aware of some data flow across language boundaries through the use of a language-agnostic intermediate representation. Their tool is based on AFL [1], a state-of-the-art fuzzer, and they compare fuzzing performance when fuzzing multilingual programs whose APIs are expressed in Python against that of Atheris [93]. The authors found that PolyFuzz performs better in terms of coverage, and equivalently in terms of number of bugs found, when compared to Atheris when fuzzing only single-language

Python code [73]. The authors also found that, when used on Python-C multilingual programs, PolyFuzz performs significantly better in terms of code coverage and bugs found when compared to Atheris, and Atheris with a C coverage instrumentation extension to allow it to receive coverage feedback from the C portion of the multilingual code [73]. The work in this thesis focuses on fuzzing Python-only programs to evaluate how the tool performs without a confounding influence from low-level code that cannot be instrumented, so the performance gains that PolyFuzz makes in multilingual fuzzing are not important for the evaluation presented here. However, the possibility of substituting Atheris for PolyFuzz in future work to extend our framework to multilingual programs–especially Python-C multilingual programs, of which there are many–is intriguing.

2.4.2 Fuzzing Other Dynamic Languages

In the realm of fuzzing other dynamic languages, such as JavaScript, Fuzzilli [47] performs targeted fuzzing of JavaScript's just-in-time (JIT) compiler, as opposed to previous work on fuzzing the JavaScript interpreter that does not exercise JIT semantics. Ruzzy [113] is a coverage-guided fuzzer for the high-level interpreted language Ruby, inspired by Atheris. Ruzzy also is built on libFuzzer, and it supports fuzzing Ruby code as well as Ruby C extensions. CL-FUZZ [25] is a fuzzer for Common Lisp. PHUZZ [88] is a coverage-guided fuzzer for web applications written in PHP, another interpreted language. This work is particularly relevant in the modern age as the internet is tightly integrated into our lives and all sorts of vulnerabilities abound in web applications. The authors of PHUZZ find 20 security issues with 2 new CVEs assigned in their evaluation. Atropos [48], is another PHP fuzzer, which shares many similarities with PHUZZ, but key differences include its use of FastCGI instead of a heavier-weight web server to improve fuzzing performance, and its ability to restore snapshots of various states of the virtual machine on which the application-under-test is running to account for stateful effects of fuzzing. Fuzzing dynamic languages has become highly relevant as dynamic languages have become popular, and this is shown by the academic and industry interest in developing such fuzzing tools and frameworks. Our work on fuzzing Python aligns well with this trend.

2.4.3 Leveraging Type Information in Fuzzing

As for leveraging type information in the fuzzing process, TIFF [62] presents an approach of using type inference to earmark offsets into a fuzzing input as containing specific types (such as int32, int64, array, etc.). In our tool, we supply encoded versions of Python objects (see Chapter 6), so our fuzzing inputs are essentially already earmarked. TIFF, however, applies specific mutation rules to the marked sections of the input that are likely to trigger problematic behavior [62]. We do not customize the mutation procedure for different subsections of the input bitvector, but this is an interesting idea for expansion of the underlying fuzzer technology that our tool operates on. Another key component of the TIFF algorithm is the use of dynamic taint analysis to learn the associations between input offsets and flow of data and control. We do not leverage dynamic taint analysis, but the use case of learning associations between function arguments and program decision points is clearly applicable in our case. Integrating with DynaPyt [34], a dynamic taint analysis framework for Python, could be explored to this end in future work.

SoFi [54] is a JavaScript engine fuzzer that uses object reflection to derive new ways to mutate program inputs. The authors also introduce a repair strategy to fix instances where mutation leads to an input that is not syntactically or semantically correct. In the present work, we only leverage type annotations that are given explicitly in code, but utilizing additional information to automatically learn how to fuzz more functions is another promising area of future work. Our tool also currently only extracts the attributes of classes, but utilizing a similar reflection-based approach to use the methods of classes to generate more realistic states of instance objects, for instance, is something also worth exploring in future work.

JSTIFuzz [141] similarly uses type inference to perform informed mutation, and they implement a corpus pretreatment step which maximizes the information in the corpus. In the present work, due to time constraints, we did not implement corpus minimization for the evaluation, but this is an essential part of future work.

Finally, DIFUZE [21] is an interface-aware kernel fuzzer, motivated by the complicated data structures that are often passed around in Unix-like systems. This motivation mirrors our own–fuzzing Python programs is complicated because of the potential for programs to manipulate complex data.

2.4.4 Bugginess of Python and Software Testing Deficiencies

Previous work has identified many bugs in both Python code and its interpreter, and there is empirical evidence suggesting that Python is a difficult language for developers to test. PyRTFuzz [72], for instance, finds many vulnerabilities in the interpreter and standard library of Python, let alone all the bugs that exist in developers' Python code. Software testing is a frequently applied method to enforce a specification on code, and to prevent regressions in behavior, but such test suites have gaps. Marques, Laranjeiro, and Bernardino [85] bring to light a case study of OpenStack, a Python software stack that is used in business-critical settings. The authors apply a fault-injection tool to the Python code and find many instances where the manual test suite did not detect the injected faults. Zhai, Casalnuovo, and Devanbu [147] find that Python test suites in commonly used projects often do not cover deeply nested code; they find that "each increase in the level of control flow nesting reduces the probability of being tested by about 19%." Manual effort is required to construct a comprehensive, high-quality test suite, and if organizations lack strict standards around code-coverage or test coverage in place, the quality of the test suite can suffer. Our approach focuses on minimizing manual effort, and using the type information that programmers have already invested manual effort into creating. We do not claim that our tool is a replacement for unit test suites, but we rather envision it as a complement to them, providing an additional layer of checks for unexpected behavior without additional human intervention. In fact, "fuzz testing" has been recognized as a key component in ensuring the security of web applications by government agencies [118]. This further demonstrates the relevance of our approach.

Other automated approaches to test generation have also been developed in response to the difficulties of software testing. Fraser and Arcuri [37], for example, present an evoluationary approach entitled EVOSUITE, with the goal of evolving entire test suites that meet sets of coverage goals, while minimizing the size of the test suite. Pacheco et al. [95], on the other hand, introduce a feedback-directed approach, where feedback from execution of a test input informs either the exclusion of the input from the test suite, or the generation of further inputs. Both of these works, like all test generation works, encounter the *oracle problem*: how do we know the expected behavior of a program in response to a test input? Fraser and Arcuri [37], in their whole test suite generation approach, reduce the burden of this problem by applying test set minimization techniques to minimize the amount of effort that needs to be invested by human oracles. Pacheco et al. [95] create oracles from simple heuristic-based approaches, related to API contracts, and they construct *regression oracles* by recording the runtime behavior of code-under-test in response to each input. This is useful for differential testing, and the authors apply this in the paper to detect inconsistencies between two versions of the Java JDK. As for test generation that focuses on Python, Pynguin [77, 76] is a framework that draws from feedback-directed test generation [95], in its approach. The authors also create a simple model of Python's type system, based on *mypy*'s internal representation of types. They also observe that type information is crucial to the test generation process. While our work is not a test generation tool, the approaches are very similar in regards to how they both leverage type information. Both works could benefit from integration with type inference as well, since type annotations in Python are optional.

2.4.5 Deficiencies of Static Tools

A common theme among studies of static tools in the literature is that they frequently emit errors when there is no problem in the underlying code. This is known as a false positive, or occasionally by the conceptually friendlier term "false alarm." There is extensive previous work on attempting to detect "false alarms" raised by static tools and filter them out [52, 55, 56]. Furthermore, Park, Lim, and Ryu [96] describe their "battles" with false alarms in static analysis of JavaScript web applications. These authors take a different perspective, where they do not strive for perfectly sound analysis with the risk of making unrealistic assumptions about applications, but to rather make fewer assumptions and allow unsoundness, while mitigating false alarms wherever possible. Kang, Aw, and Lo [63] highlight problems in machine-learning based approaches to false alarm detection, and identify issues in previous work and data used to train models. This suggests that there is much work left to do in making static tools viable.

Gong et al. [43] further show that similar deficiencies that we have identified in static type checkers for Python also exist in static linters for JavaScript (a linter is a tool that enforces certain rules about code style and the use of certain idioms). In response, the authors developed a dynamic tool called DLINT. This tool differs from the one we have built, however, because they do not leverage fuzzing to find faults. Instead, they develop a ruleset of runtime checks, and directly instrument the code-under-test with additional statements that verify these rules. Instrumenting the code-under-test in order to augment the fuzzing process with additional information is, however, a direction we are considering for future work. Additionally, the motivation behind DLint again mirrors our own: we apply a dynamic approach to make up for the deficiencies in static type checkers with the aim of creating a tool that is readily adoptable.

2.4.6 General Fuzzing-Related Works

Turning to other works in the area of fuzzing, Superion [139] is a grammar-aware fuzzing approach that includes an abstract syntax tree trimming component and abstract syntax tree mutation. DatAFLow [57] presents a data-flow guided fuzzer based on AFL, though the authors conclude that control-flow guided fuzzing approaches are more performant, and therefore effective at generating larger sets of test inputs to feed to the code under test. FuzzGen [61] proposes a method for automatically generating fuzzing harnesses by inferring valid uses of the API of the code-under-test. Skyfire [138] presents an approach to generate well-distributed complex fuzzing inputs through analyzing large collections of existing valid inputs. However, these preexisting sets of inputs may not always be available in practice. These works, along with others mentioned in this section, exhibit thematic interest in two key problem spaces related to fuzzing performance: generating high quality inputs, and generating high quality fuzzing harnesses. Although our approach does not directly compete with any of the aforementioned fuzzers (most of them are geared specifically towards C/C++ programs, anyway), we offer a solution to the input generation problem with our model of Python's type system and automatic extraction of type annotations. We create a basic test harness that supplies inputs of the expected types to the function under test with no extra human effort as well. Integrating ideas for improving the effectiveness of the harnesses we generate, like those discussed in PyRTFuzz [72] or FuzzGen [61], is a promising direction of future work.

"Evaluating Fuzz Testing" [70] is a metastudy of many fuzzing papers which scrutinizes their evaluation methodologies and discovers several papers that are irreproducible, or are reproducible only with specific cherry-picked inputs. Similarly, "Seed Selection for Successful Fuzzing" [59] reports similar findings that the initial seeds have an avalanching effect on fuzzer performance, and that seed selection should be top-of-mind when evaluating fuzzers. We are confident that our work does not fall into these same pitfalls, because by definition, our approach removes all human effort in generating initial seed inputs, and as suggested by these papers, we have evaluated our fuzzer using multiple initial random seeds which are used in generation of input corpora for fuzzing.

Böhme, Pham, and Roychoudhury [6] present "AFLFast," which is a coverage-guided greybox mutational fuzzer that is augmented by a Markov chain model to prevent the repeated exploration of the same parts of code by many fuzzing inputs. The authors find that intentionally choosing inputs that exercise "low-frequency" paths improve fuzzing performance significantly. As we are likely falling into the same pitfall of many inputs repeatedly exercising the same code, the ideas here could be of use in future work. Shastry et al. [116] present "Orthrus," which uses static program analysis to inform fuzzer input generation, and they also glean positive results. We have also considered creating deeper integrations with static tools, in particular type inference tools, in future work.

A number of previous papers also focus on constructing hybrid fuzzing approaches that are augmented by symbolic or concolic execution. Driller [119] is a fuzzer which is augmented with symbolic execution in order to pass complex conditionals that block code regions. Similarly, SAFL [140] is another hybrid algorithm that uses symbolic execution to learn high-quality initial seeds, which are paramount to fuzzing performance, as previously discussed. QSYM [144] and FUZZOLIC [7] are two more examples of fuzzing integrated with concolic execution. Concolic execution works well when combined with fuzzing because it provides an additional aid when the fuzzer gets stuck on navigating complex program boundaries. While we do not apply symbolic or concolic execution in our current approach, we have identified this as a promising future direction as well.

3 Overview of ACL2s

ACL2 [66, 67, 68] is a programming environment that provides a Common Lisp-like language, an automated theorem prover, and an extensible theory in first-order logic. The ACL2 Sedan, ACL2s [30, 13], is an extension to the ACL2 theorem prover that includes a data definition framework [15], a counterexample generation framework [19, 18, 16, 14], termination analysis with calling-context graphs [82] and ordinals [79, 80, 81], as well as a property-based modeling and analysis framework. ACL2s also contains a systems programming framework [137] that provides an API allowing queries to ACL2s using external programming languages.

3.1 Data Definition

In this tool, the type information service is written using ACL2s. We use the data definition framework to represent Python types in ACL2s.

3.1.1 Primitive Types

The primitive types in ACL2s roughly correspond to the primitive types in Python, so no substantial transformation is needed. The main difference occurs with floating point numbers. ACL2s has a `rational` type, and we use this type to represent floating point numbers. The rational numbers are a superset of floating point numbers, so this provides the functionality we need for our purposes. We also add the special values of `inf`, `-inf`, and `nan` to the floating point type.

3.1.2 Complex types

Lists are represented with the "true list" type in ACL2s. The ACL2s data definition framework allows for specifying an arbitrary type as the inner type of the list using the `listof` construct [15]. Dictionaries are represented with the `map` type in the ACL2s data defintion framework [15].

Record types are represented with the `record` construct of `defdata`, where the class's field names are mapped to values of the appropriate types [15].

Union types are represented using `defdata`'s `or` construct [15].

3.2 Custom Enumerators

In ACL2s, data types are *enumerative*, meaning that each type is associate with an enumerator function that maps the set of natural numbers to objects of the type [15]. We leverage this property of ACL2s data definitions to create a model of the Python type system from which representative examples can be extracted for fuzzing. When creating this model, we found that the default enumerators for some primitive types in ACL2s have limited ranges that do not represent their Python counterparts. The default enumerator for strings, for example, emits only alphanumeric strings, but Python strings can have Unicode characters. We include special "edge case" values of each type in the enumerators in an attempt to maximize the chances of fuzzing eliciting problematic behaviors in code. The definitions of these custom enumerators are given in this section.

3.2.1 Integers

To exercise code that works well with integers, we have created a custom enumerator for integers based on the heuristic that powers of 2 and values around them tend to be interesting and may be more likely to trigger failure cases in code. The custom enumerator generates integers from the following cases with the following probabilities. For convenience of notation, where $l, i, h \in \mathbb{Z}$, let $P_2^+(l,h) = \{2^i \mid l \leq i \leq h\}$, $P_2^-(l,h) = \{-2^i \mid l \leq i \leq h\}$, and $P_2^\pm(l,h) = P_2^+(l,h) \cup P_2^-(l,h)$.

- **Sum of powers of two:** 85% chance of generating an integer from the set: $\{a + b \mid a \in P_2^\pm(0,64) \wedge b \in P_2^\pm(0,16)\}$
- **Powers of 2, with off by one:** 6% chance of generating an integer from the set: $UnionAll(\{\{a, a-1, a+1\} \mid a \in P_2^\pm(0,65)\})$
- **65-bit integers:** 6% chance of generating an integer from the set: $UnionAll(\{\{a, -a\} \mid 2 \leq a \leq 2^{65}\})$
- **One:** 1% chance of generating 1.
- **Zero:** 1% chance of generating 0.
- **Negative one:** 1% chance of generating -1.

Implementation Errors

Due to an implementation error, the second and third cases were merged into a single case as follows. These could not be corrected for this work because of time constraints– the entire evaluation would have needed to be redone with the corrected implementation. This will be fixed in future work.

- **Powers of 2, with off by one:** 12% chance of generating an integer from the set: $UnionAll(\{\{a, a-1, a+1\} \mid a \in P_2^\pm(0,65)\})$

3.2.2 Strings

For strings, we generate many different varieties of Unicode strings. The probabilities are broken down as follows:

- **ASCII strings:** 50% chance to generate an ASCII-only string
- **Mixed strings:** 40% chance to generate a "mixed" string, which contains characters from the ASCII character set, as well as characters from all of the following sets
- **Emoji strings:** 2% chance to generate an emoji-only string
- **Greek-letter strings:** 2% chance to generate a string with Greek-language characters only
- **Mathematical symbols:** 2% chance to generate a string with mathematical and logic symbols only
- **Latin diacritics:** 2% chance to generate a string with Latin diacritics only
- **Compound emojis:** 2% chance to generate a string with "compound emojis," which are emoji characters that span two or more codepoints

3.2.3 Floats

ACL2s does not have a true notion of a "float"–rather, ACL2s performs arithmetic on rational numbers. In order to define a floating point type that is representative of Python's float type, we define a custom enumerator that produces rational numbers in pre-defined interesting categories, as well as special case values (note that all numbers that can be expressed as a floating point value are rational numbers). We use the same $P_2^+ / P_2^- / P_2^\pm$ notation from the previous integer enumerator definition. There is focus placed on generating "edge case" values that may be likely to trigger interesting behavior.

- **Rational Numbers:** 75% chance to generate a rational number, n/k, where n, k are produced using the enumerator for integers described previously in this section
- **Powers of 2 with small-magnitude exponents:** 4% chance of generating a number from the set:
  $UnionAll(\{\{a, a-1, a+1\} \mid a \in P_2^{\pm}(-64, 64)\})$
- **Powers of 2 with large-magnitude exponents:** 4% chance of generating a number from the set:
  $UnionAll(\{\{a, a-1, a+1\} \mid a \in P_2^{\pm}(65, 1024) \cup P_2^{\pm}(-1024, -65)\})$
- **Min and max normal 32-bit floats:** 3% chance of generating a number from the set:

$$
\begin{aligned}
&\{2^{-126}, \\
&2^{-126} + 1, \\
&2^{-126} - 1, \\
&2^{127}(2^{-23} - 2), \\
&2^{127}(2^{-23} - 2) - 1, \\
&2^{127}(2^{-23} - 2) + 1\}
\end{aligned}
$$

- **Min and max normal 64-bit floats, with off by one:** 3% chance of generating a number from the set:

$$\begin{aligned}
&\{2^{-1022},\\
&2^{-1022}-1,\\
&2^{-1022}+1,\\
&2^{1023}(2^{-52}-2),\\
&2^{1023}(2^{-52}-2)-1,\\
&2^{1023}(2^{-52}-2)+1\}
\end{aligned}$$

- **Max integer representible as a 32 or 64 bit floating point number:** 2% chance of generating a number from the set:
  $\{2^{24}, -2^{24}, 2^{53}, -2^{53}\}$

- **Min and max subnormal 32 and 64 bit floats:** 2% chance of generating a number from the set:

$$\begin{aligned}
&\{2^{-149},\\
&-2^{-149},\\
&2^{-126}(1-2^{-23}),\\
&-2^{-126}(1-2^{-23}),\\
&2^{-1074},\\
&-2^{-1074},\\
&2^{-1022}(1-2^{-52}),\\
&-2^{-1022}(1-2^{-52})\}
\end{aligned}$$

- **Not-a-number:** 1% chance of generating `nan`

- **Positive Infinity:** 1% chance of generating `inf`

- **Negative Infinity:** 1% chance of generating `-inf`

Implementation Errors

The following cases of the enumerator were implemented incorrectly in the initial version of the implementation. These could not be corrected for this work because of time constraints–the entire evaluation would have needed to be redone with the corrected implementation. This will be fixed in future work. The actual behavior of the implementation is as follows.

- **Min and max normal 32-bit floats:** 3% chance of generating a number from the set:
  $\{2^{-126}, 2^{-126} + 1, 2^{-126} - 1, 2^{104}, 2^{104} - 1, 2^{104} + 1\}$
- **Min and max normal 64-bit floats:** 3% chance of generating a number from the set:
  $\{2^{-1022}, 2^{-1022} - 1, 2^{-1022} + 1, 2^{972}, 2^{972} + 1, 2^{972} - 1\}$
- **Min and max subnormal 32 and 64 bit floats:** 2% chance of generating a number from the set:

$$
\begin{aligned}
&\{2^{-149},\\
&-2^{-149},\\
&2^{-126}(2^{-23}-1),\\
&-2^{-126}(2^{-23}-1),\\
&2^{-1074},\\
&-2^{-1074},\\
&2^{-1022}(2^{-52}-1),\\
&-2^{-1022}(2^{-52}-1)\}
\end{aligned}
$$

3.3 Related Work

The ACL2 Sedan [30], abbreviated ACL2s, was introduced as a beginner-friendly interface to the ACL2 [68, 66] automated theorem proving system. It includes features such as termination analysis with calling-context graphs [82] and the `defdata` data definition framework [15], making it easy to model and prove theorems about arbitrary data. The data types in ACL2s are *enumerative*, meaning that they have enumerators attached which act as generators of examples of the types. This is the foundation of the counterexample generation system in ACL2s [20, 17, 14].

Building on the foundation of enumerative data types in ACL2s, Walter, Greve, and Manolios [135] introduce constraints on the types, and use them to generate sample input data to test the 802.11 Wi-Fi protocol. Generating data that satisfies the constraints of the 802.11 specification is difficult for modern SMT solvers, but a two-stage approach involving parameterized enumerators and SMT solvers performs well in generating many correct examples of 802.11 protocol frames. This work on dependent types may become useful to TYPE HINT FUZZING in the future, as the set of types in Python's type system grows.

ACL2s has also become useful as a pedagogical tool. Walter, Kumar, and Manolios [136] introduce an Eclipse IDE plugin which presents an interactive proof checking interface with the goal of effectively teaching proof-writing skills to undergraduate students in formal logic courses.

3.4 Acknowledgements

I would like to thank Andrew Walter and Pete Manolios for their contributions to the design of these custom enumerators. I would further like to thank Andrew Walter for his assistance in implementing these custom enumerators.

4 Overview of Atheris/libFuzzer

4.1 libFuzzer

libFuzzer [74] is a state-of-the-art coverage-guided mutational fuzzer that is included as a part of the LLVM compiler infrastructure [124]. It targets C/C++ code, and it supports a wide variety of configuration options. It also integrates with various *sanitizers*, such as AddressSanitizer (ASAN), UndefinedBehaviorSanitizer (UBSAN), and MemorySanitizer, to catch various classes of bugs that can lead to security vulnerabilities such as buffer overflows and out-of-bounds reads. Coverage instrumentation is provided by SanitizerCoverage, which instruments code at function, basic block, and edge levels [123].

libFuzzer has been recognized for discovering many bugs in programs, listed in the "Trophies" section of its documentation [74].

4.1.1 Fuzzing Loop

Algorithm 1 gives the high-level fuzzing loop. INITIALIZE performs the initialization, which consists of running every input in *corpus* through the function-under-test (FN) [74]. PICKUNIT picks a unit to mutate. Which unit is picked can be customized based on command-line options, but the default setting is to use *entropic seed scheduling* [84]. After a unit has been picked, it is mutated. libFuzzer has various built-in mutation procedures, such as CROSSOVER, ERASEBYTES, and SHUFFLEBYTES, to name a few. libFuzzer will apply up to `mutation_depth` mutations in a row, sequentially, where `mutation_depth` is a configuration parameter that defaults to 5 (in our experiments, we left this at its default value).

Algorithm 1 Core Fuzzing Loop

```
function FuzzLoop(corpus, numRuns, FN)
    succeeded ← Initialize(corpus, FN)
    if not succeeded then
        error
    end if
    repeat
        unit ← PickUnit(corpus)
        mutatedUnit ← Mutate(unit)
        FN(mutatedUnit)
    until numRuns times, or crash
end function
```

4.1.2 Command Line Options

We have found two command-line options useful in influencing the behavior of libFuzzer in a way that seems to increase the amount of unique fuzzing activity. The first is `-keep_seed`. libFuzzer maintains an internal in-memory corpus that is different from the on-disk corpus directory supplied at fuzzer startup. During fuzzing, libFuzzer selects a next element to mutate from this internal corpus. By default, libFuzzer will only add elements to this internal corpus if they trigger new coverage. `-keep_seed` instructs libFuzzer to always add newly mutated elements to the internal corpus, even if they do not trigger new coverage. This means that there are more units available for *crossover*, the mutation strategy where libFuzzer tries to mix the contents of two units together to produce a new input. Given that our encoding will always successfully decode by design (see Chapter 6), we chose to use this option because it increases the space of possible mutations, which may decrease the chance that the fuzzer will reach a point of stalling progress. The second command-line option is `-cross_over_uniform_dist`, which instructs the crossover operation to select the corpus element to cross over with the current input uniformly, instead of the entropic seed scheduling procedure that is enabled by default. This again is intended to increase the variety of inputs generated through cross over. Through informal observation, we have found these command line options to increase the variety of inputs that the fuzzed function receives, especially with limited type information. Follow-up work may address the degree to which these command line options help or harm the fuzzing process.

4.1.3 Reduction

An important feature of libFuzzer is that it will attempt to reduce the size of inputs that trigger some coverage pattern. When libFuzzer finds an input that is shorter than a previous one but produces the same coverage, it adds the input to the corpus.

4.2 Atheris

We integrate with Atheris, which is a Python-language fuzzer that is built on top of libFuzzer. Atheris was introduced in 2020 by Google [45]. Atheris is not highly popular, but it has begun to be used in the literature as fuzzing Python code is becoming more of a topic of interest [72, 50, 73]. According to the Atheris authors, it is useful for testing a program where there is a well-defined notion of what "good" and "bad" behavior are. For many Python programs, "bad" behavior is as simple as throwing an unexpected exception. For example, the authors note that Atheris was able to successfully disprove a claim from a YAML parsing library that only exceptions of type `YAMLError` would be thrown. Atheris found that other types of exceptions could be thrown, which "indicates flaws in the parser" [45].

Atheris hooks into libFuzzer's code coverage system by simulating the loading of shared libraries, and tricking libFuzzer into thinking Python code is dynamically loaded shared libraries [93]. However, the benefits of using sanitizers such as AddressSanitizer and UndefinedBehaviorSanitizer can only be reaped when fuzzing native extensions to Python. In our evaluation, we focus primarily on Python-first repositories, so we do not consider bugs in native C code, though this may be an interesting avenue for future work.

4.3 Related Work

Manès, Böhme, and Cha [84] contribute the "entropic power scheduling" behavior to libFuzzer, which has since been merged into libFuzzer and is used as the default seed scheduling algorithm. It utilizes information theory to select seed inputs from the supplied corpus to mutate and supply to code under test, maximizing the information gained about behaviors of the code.

Vishnyakov et al. [134] present Sydr-Fuzz, which is a hybrid fuzzing framework that integrates with libFuzzer and AFL. The paper compared Sydr-Fuzz to libFuzzer and AFL individually, and found that their hybrid approach is comparable and can generate many interesting inputs.

Böhme and Falk [12] reveal counterintuitive results about fuzzing: discovering new bugs linearly requires computational resources exponential in the distance from previously covered code. They demonstrate these findings through experiments with libFuzzer and AFL, the two state-of-the-art coverage-guided fuzzers.

5 Tool Design and Implementation

In this chapter, we cover the architecture of the tool and various implementation details. We have endeavored to make the design and implementation of the TYPE HINT FUZZING tool as modular and extensible as possible, to enable future work.

5.1 High-level Architecture

A high-level architecture diagram is given in Figure 5.1. Components in orange rectangles are external, pluggable components, and all the components in white represent parts of the tool that were programmed as part of this work.

Phase 1 is the *Type Information Extraction* phase. As the type information extractor traverses the codebase-under-test (CUT), it extracts information about attribute accesses on user-defined types (classes), as well as function and method signatures. From this information, a *type set* and a set of *function candidates* are extracted. The type set is the subset of all Python types that comprise the model of Python's type system that underpins the functioning of the tool. This type set, along with the signature information from the extracted type information, are used to generate the function candidates, which is the set of top-level functions defined in the codebase whose parameter types and return type are fully within the set of types that the model understands.

Phase 2 is where fuzzing occurs, and is the phase in which the tool spends the longest. The tool iterates through each function candidate, generates a corpus, and fuzzes the function-under-test (FUT) until the time budget expires. If the tool is configured to do so, it will reach out to the type modeling service for new examples at a configurable frequency, with the goal of periodically exposing the fuzzer to new parts of the search space. This idea is partially inspired by Klees et al. [70], who recommend that fuzzing tools borrow an idea from SAT solvers: randomly resetting the solving process in order to prevent falling into a local minimum that is hard to escape. As fuzzing progresses, the results are collected and transferred to Phase 3 for processing.

Phase 3 is where the results of the fuzzing are processed into data that was used to generate the results for this thesis. For industrial use, this processing step can be disabled entirely, and run separately or not run at all if the user prefers to simply analyze the raw output of Phase 2. We include Phase 3 here nonetheless to clarify how we obtained the results herein and to inform attempts at reproduction of the results. In the "Coverage Replay" process, we "replay" the inputs that were supplied to each FUT, in order, to measure how much of the FUT and the repository as a whole was covered. In the "Store Results" process, we store the results in a multiset-like data structure to maximize space efficiency, as the results can grow quite large. Finally, we check the types of all of the return values for non-exceptional fuzzing iterations, and flag any of them that do not match the return type annotation of the FUT.

In the following subsections, we cover each of these Phases in greater detail, and in the following section, we cover concrete implementation details that the reader might find interesting.

5.1.1 Type Information Extraction

Type Information Extraction happens in two subphases: an *extraction* phase, and a *registration* phase. *Extraction* is a static analysis process that combs over the CUT, seeking information about 1) top-level function signatures, 2) user-defined class attribute

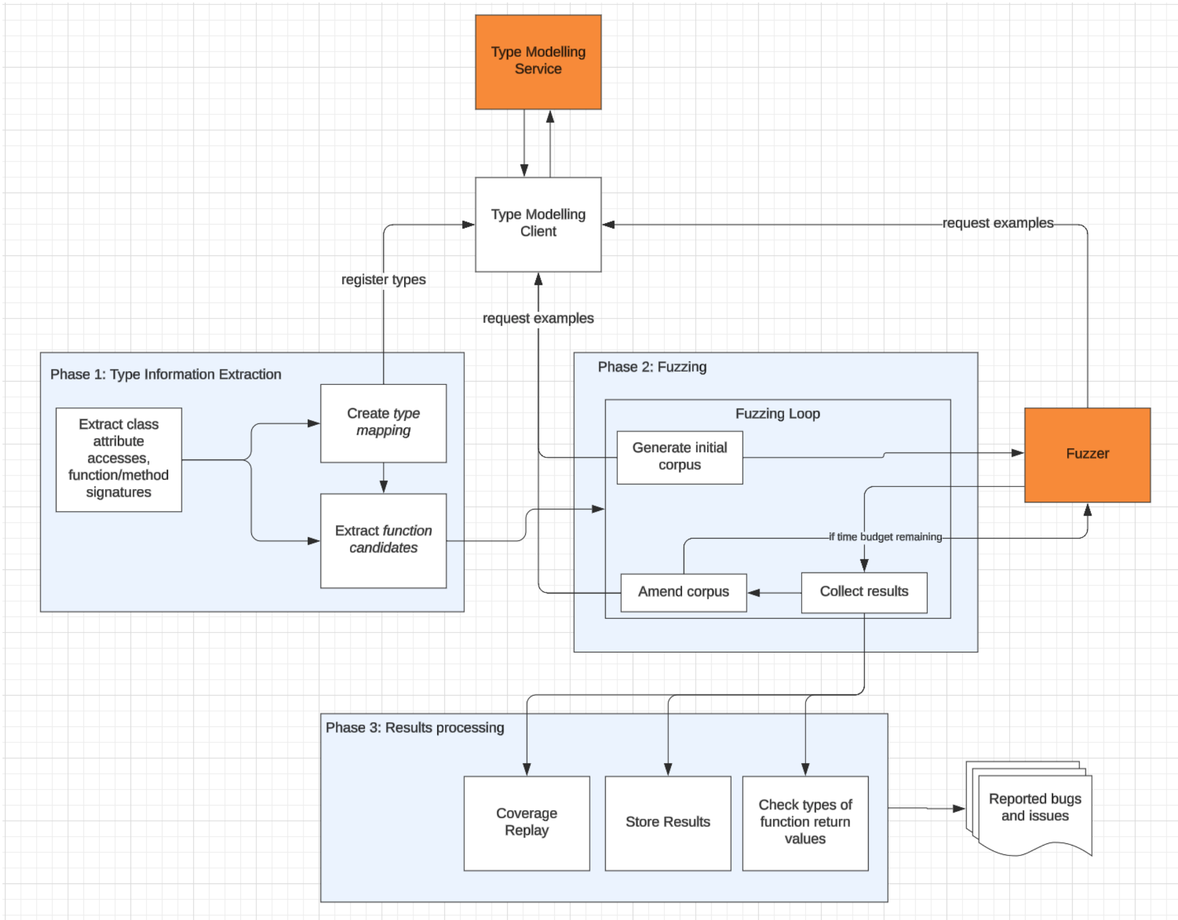

FIGURE 5.1: The architecture of the tool

types, 3) and user-defined class method signatures. *Registration* in turn has two sub-phases: *type set construction* and *function candidate resolution*. The purpose of *type set construction* is to capture any user-defined types that can be expressed in terms of types that have previously been registered as understood. The procedure repeats until a maximum number of iterations is executed or a fixed point is reached, whichever occurs first. After the type set has been constructed, *function candidate resolution* takes place, where the set of top-level functions whose signatures are fully understood under the subset of Python types that we have modeled is extracted.

Pseudocode for *Type Set Construction* is given in Algorithm 2. Lines 2-6 set up the initial state. `INITIAL_TYPE_SET` (line 2) is predefined to include a number of common Python types, inspired by the set of most commonly used types in type annotations from Rak-amnouykit et al. [101]. `maxIters` (line 3) sets the maximum number of iterations of the main loop of the algorithm. The termination condition of the algorithm (line 7) is whether we have executed `maxIters` iterations, or whether we have hit a fixed point (i.e. the type set has not changed since the last iteration). The main loop (lines 8-17) iterates through all the classes that were extracted in the *Extraction* process, and admits a class to the type system model if all of its attribute types and method signatures are understood under the current model. When checking whether all attributes and methods are understood on lines 12-13, we use expressions of the form "$\forall a \in \text{set} :: P(a)$", which evaluate to a boolean value that indicates whether all elements a in "set" satisfy a predicate P. The procedure is iterative to account for situations like the following: suppose `class B` has a field of type `class A`, and `A` has all primitive-typed fields. The first iteration will be guaranteed to admit `A`, and `B` will be admitted in the second iteration, now that `A` is admitted.

Pseudocode for *Function Candidate Resolution* is given in Algorithm 3. Since we have the type set from the previous step, we need only iterate through all of the top-level function signatures extracted as part of the *Extraction* phase and use the IsSigUnderstood helper function (definition given in Algorithm 4), which returns whether a given function signature is *fully understood* under a type set. We say that a function signature is *fully understood* if and only if all of the types of its positional arguments (args), keyword arguments (kwargs), and its return type are members of the type set.

5.1.2 Fuzzing

After the type information is extracted and the set of function candidates has been determined, fuzzing can begin. Our tool is designed to be as agnostic as possible to

Algorithm 2 Type Set Construction

```
function TypeSetConstruction(signatureInfo, attribInfo, methodInfo)
    typeSet ← INITIAL_TYPE_SET  ▷ This is a predefined constant. See implementation details for the value.
    maxIters ← 5
    classes ← attribInfo.keys() ∪ methodInfo.keys()
    iter ← 0
    prevSet ← null
    while iter < maxIters and (prevSet is null or prevSet ≠ typeSet) do
        prevSet ← copy(typeSet)
        for class in classes do
            attributes ← classes[class] or empty mapping
            methods ← classes[class] or empty mapping
            understandAllAttrs ← ∀a ∈ attributes.values() :: a ∈ typeSet
            understandAllMethods ← ∀m ∈ methods.values() :: IsSigUnderstood(m, typeSet)
            if understandAllAttrs and understandAllMethods then
                typeSet ← typeSet ∪ {class}
            end if
        end for
    end while
    return typeSet
end function
```

Algorithm 3 Function Candidate Resolution

```
function FunctionCandidateResolution(signatures, typeSet)
    functionCandidates ← ∅
    for signature in signatures do
        if IsSigUnderstood(signature, typeSet) then
            functionCandidates ← functionCandidates ∪ {function}
        end if
    end for
    return functionCandidates
end function
```

Algorithm 4 IsSigUnderstood Helper Function

```
function IsSigUnderstood(signature, typeSet)
    for arg in signature.argTypes do
        if arg ∉ typeSet then
            return false
        end if
    end for
    for arg in signature.kwargTypes do
        if arg ∉ typeSet then
            return false
        end if
    end for
    if signature.returnType ∉ typeSet then
        return false
    end if
    return true
end function
```

the actual implementation of the fuzzer. We rely on a small number of key details: 1) the fuzzer accepts a *corpus* (C) of seed inputs (the *corpus size* is the number of elements in the corpus, hereinafter occasionally notated as $|C|$), 2) the fuzzer accepts as input a function pointer to a harness which accepts a single bitvector as input, and 3) the fuzzer allows the consumer to hook into the mutation process (this is how we facilitate occasional fetching of additional seed examples from the Type Modeling Service).

The definition of the main fuzzing loop is given in Algorithm 5. The arguments (line 1) are the function-under-test (FUT), the types of its positional arguments, the types of its keyword arguments, and its return type, respectively. Lines 2-4 define local variables. *knownExceptions* is the set of exceptions that have been encountered so far during fuzzing. The *corpus* is generated in CreateCorpus, given in Algorithm 6. `client.ExampleGenerator` (line 2 of Alg. 6) produces a generator object, which can be thought of as an infinite list of random examples of assignments to all of the positional and keyword arguments to the FUT. *corpusSize* elements are extracted from the list and the corpus is returned. On line 4 in Algorithm 5, a higher-order function, CreateHarness, is called to generate the harness. The definition of CreateHarness is given in Algorithm 6.

Algorithm 5 Phase 2 Fuzzing Loop

```
function FuzzLoop(fn, argTypes, kwargTypes, returnType, mutatorHook)
    knownExceptions ← ∅
    corpus ← CreateCorpus(argTypes, kwargTypes, corpusSize)
    harness ← CreateHarness(fn, argTypes, kwargTypes, returnType)
    while time budget remaining do
        corpus ← AmendCorpus(fn, argTypes, kwargTypes, knownExceptions, corpus)
        okResults, failResults, crashResults ← Fuzzer(harness, corpus, mutatorHook)
        crashes ← { c.exception for c in crashResults }
        knownExceptions ← knownExceptions ∪ crashes
    end while
end function
```

Algorithm 6 Phase 2 Helper Functions (Part 1)

```
function CreateCorpus(argTypes, kwargTypes, corpusSize)
    exampleGen ← client.ExampleGenerator(argTypes, kwargTypes, batchSize)
    corpus ← []
    while corpus.size < corpusSize do
        example ← Next(exampleGen)
        encodedExample ← Encode(example, argTypes, kwargTypes)
        corpus.Append(encodedExample)
    end while
    return corpus
end function

function CreateHarness(fn, argTypes, kwargTypes, returnType)
    function Harness(data)
        args, kwargs ← Decode(data, argTypes, kwargTypes)
        result ← fn(∗args, ∗ ∗ kwargs)
        if result is exception then
            record exception
        else if result is ok then
            if returnType.CheckType(result) then
                record successful trial
            else
                record function contract violation
            end if
        end if
    end function
    return Harness          ▷ Note that this function is returning another function: the harness
end function
```

To understand lines 5-10 of Algorithm 5, it is helpful to note that an additional implementation detail of the fuzzer is implicitly assumed: the fuzzing process is expected to occasionally terminate (the reason for the termination is left up to the implementer), acting as a "checkpoint" where results can be collected in batches and exceptions can be processed into the *knownExceptions* (line 9). This behavior also allows for an opportunity to amend the corpus with AMENDCORPUS (line 6). Atheris [93] exhibits this behavior by terminating upon the first exception in the function-under-test that it encounters. We have found this corpus amendment behavior important in situations where all elements of the initial corpus trigger exceptions. This is because Atheris, in its initialization phase, will try to send all elements of the corpus unchanged through the FUT.

AMENDCORPUS is defined in Algorithm 7. AMENDCORPUS filters out corpus elements that cause known exceptions (lines 3-10), and (depending on the tool's configuration parameters) will optionally "top off" the corpus, adding new elements to it. This "top-off" behavior has two modes. The first mode tops the corpus off if the corpus has dipped below full at all (lines 12-17), and the second mode only adds one element to the corpus if it has become empty, to make it nonempty (lines 18-22). We use the second mode in our experimental configurations, because it has proven to be better at allowing the fuzzing process to start up successfully (for functions where many possible inputs trigger exceptions, it is easier to generate a single non-exceptional input than, say, 10 at a time).

Finally, the *mutatorHook* argument to FUZZLOOP that is passed to FUZZER is a callback function that is invoked at each round of mutation (this function is known as a "custom mutator" in libFuzzer terminology). In this function, there is the opportunity to customize the mutation behavior of the fuzzer arbitrarily, however in this tool it is used to fetch fresh examples from the type modeling service in a controlled way.

Algorithm 7 Phase 2 Helper Functions (Part 2)

```
function AMENDCORPUS(FN, argTypes, kwargTypes, knownExceptions, corpus, originalCorpusSize)
    exampleGen ← client.EXAMPLEGENERATOR(argTypes, kwargTypes, batchSize)
    for c in corpus do
        args, kwargs ← DECODE(c, argTypes, kwargTypes)
        result ← FN(*args, **kwargs)
        if result is exception then
            if result ∈ knownExceptions then
                corpus ← corpus \ {c}
            end if
        end if
    end for
    if should top off corpus then
        while corpus.size < originalCorpusSize do
            example ← NEXT(exampleGen)
            encodedExample ← ENCODE(example, argTypes, kwargTypes)
            corpus.APPEND(encodedExample)
        end while
    else if corpus is empty and should add to corpus if empty then
        example ← NEXT(exampleGen)
        encodedExample ← ENCODE(example, argTypes, kwargTypes)
        corpus.APPEND(encodedExample)
    end if
    return corpus
end function
```

5.1.3 Results Processing

The coverage replay algorithm is trivial. We simply reproduce the sequence of inputs that were given to a function-under-test, in order, and replay them into the function until a predetermined limit, if one is set. While the inputs are getting replayed, code coverage is collected.

5.2 Implementation Details

The tool is implemented in Python 3.8, in about 13K lines of code. In development, we emphasized strong object-oriented design and composability of the various components. We are hopeful that this property of the tool will increase its reusability by the community. In addition to the Python component, we also integrate with ACL2s [30] (see Chapter 3) as the Type Modeling Service, and Atheris [93] as the Fuzzer, as displayed in Figure 5.1.

5.2.1 Type Information Extraction

The model of the type system is implemented as a Python class hierarchy. The root of the class hierarchy is `PyCgenType`. Every type extends this root type. The "type set" referred to in Algorithm 2 and Section 5.1.1 is actually implemented as a type *dictionary*, that maps names of types to functions that construct `PyCgenType` instances. Each `PyCgenType` instance knows how to communicate with the Type Modeling Service, and it knows how to translate responses from the Type Modeling Service into actual Python objects.

Type Modeling Service

The Type Modeling service exposes the following APIs:

- `request_examples`: request examples of types from the service
- `register_record`: submit a record type (i.e. a class) for registration; all types of all fields must already exist/be registered
- `register_union`: submit a union type for registration; all types of the union must already exist/be registered
- `alias`: create a type alias for an existing type
- `undef`: unregister a type
- `set_seed`: sets the random seed used to generate examples

The `client` module in the tool's code exposes these APIs to the rest of the program. The `TypeInfoService` interface in the `tis` module encapsulates managing this service's startup and teardown.

Type Mapping

In Section 5.1.1, we introduced the concept of the *type set*, which is the set of all types that have been admitted to the model, and we discussed an iterative procedure that admits user-defined types to the model based on the types from the last iteration. This type set is a implemented as a mepping from type names to callable objects that produce a valid corresponding `PyCgenType` instance. Table 5.1 provides the initial set of type mapping elements. This set of types essentially represents the subset of the Python type system that we natively understand. This is by no means complete, and we hope to continue to expand this moving forward. Notably, it does not contain many of the typing constructs from Python's `typing` module [130].

TABLE 5.1: Initial state of type mapping

Type Identifier	**Callable**
builtins.int	PyCgenInteger
builtins.str	PyCgenUnicodeCodepointString
builtins.bool	PyCgenBool
builtins.float	PyCgenFloat
builtins.list	PyCgenList
builtins.bytes	PyCgenBytes
builtins.set	PyCgenSet
builtins.dict	PyCgenDict
None	PyCgenNoneType
Tuple	PyCgenFixedTuple
builtins.tuple	PyCgenVariadicTuple
Union	PyCgenUnionType
Any	PyCgenAnyType
AnyType	PyCgenAnyType

Note that we include an entry for "Any" (and its alias "AnyType"). This represents the *dynamic* type, as it is known in the literature. This type is consistent with every other type and every other type is consistent with it [117]. In order to gather data on how functions annotated with "Any" behave, we represent Any as a large union between many different types, specifically, `Union[int, str, float, bytes, bool, None, List[str], List[int]]`.

Function Candidates

Function candidates end up occasionally not being fuzzed, either due to import errors, or due to various errors during the fuzzing process that render results collection impossible. The counts of function candidates therefore represent current upper bounds on the number of functions we can actually fuzz. We are continuing to investigate sources of these errors and mitigating them.

5.2.2 Results Multiset

A key implementation detail of the tool is the results storage format. We give it the name of *multiset* because it is essentially a multiset of "fuzzing points," which are observations of input-output pairs during fuzzing. It is implemented as a dictionary that maps result "structures" to records describing when the particular point was observed. The result structures contain the positional arguments and keyword arguments supplied to the FUT, and the return value or exception data if an exception was thrown. The observation record stores the number of times the point appeared during fuzzing, the index of the fuzzing iteration in which the point was first observed, and a list of the relative timestamps during the fuzzing campaign when the point was observed.

5.2.3 Monkeypatching

Monkeypatching is a rather humorous term for the practice of dynamically overriding attributes of imported libraries at runtime to change the behavior of code that uses those libraries. Since the code we fuzz may make calls to the filesystem or other operating system calls, we implemented monkeypatches for these libraries to minimize the chance of the operating system either terminating the process, or the process producing unexpected results in the environment. In other words, it is a primitive form of process isolation. We monkeypatch calls from `os`, `shutil`, `pathlib`, `subprocess`, and the `open` call from the standard library.

5.2.4 Custom Mutator

The type hint fuzzer implementation allows for freshly generated inputs to be inserted into the fuzzing process while it is running. The frequency at which this happens is controlled by a parameter called `acl2s_reachout_frequency`, shortened for brevity to `acl2s_reachout_freq`.

5.2.5 Configuration Files

The tool is configured through the use of INI-style configuration files. These configuration files control almost every aspect of how the algorithm runs, such as timeout, encoding, and how to behave when the corpus becomes empty due to filtering, among other options. Here, some important configuration options are listed, and a full reference is given in Appendix A.

- `acl2s_reachout_frequency`: When `backend` is set to `atheris`, this controls the frequency at which a new input from ACL2s will be pulled during the fuzzing campaign, instead of using a mutated seed input from Atheris. 0.7, for example, means that 70% of fuzzing iterations will pull a fresh example from ACL2s.

- `backend`: This is the fuzzing backend to use. Can be either `acl2s` or `atheris`. The `acl2s` backend represents ACL2s-only fuzzing, with no coverage-guided mutation. `atheris` represents fuzzing with Atheris. More backends may be added in the future.

- `corpus_size`: The size of the corpus. This is applicable when `backend` is set to `atheris`.

- `metadata_only`: If set to True, scrambles the data portion of all seed inputs with random bytes, to remove the influence of primitive type information.

- `memory_limit_mb`: The memory limit, in megabytes.

- `random_seed`: The random seed of the fuzzing campaign.

- `style`: Controls the "style" of the fuzzing run: whether it is limited by time, or by number of iterations. Takes two values, `TIMEOUT` and `RUN_LIMIT`.

6 Custom Encoding

In this chapter, we discuss the motivation, design, and implementation of the custom input encoding that we developed to facilitate effective fuzzing.

6.1 Motivation

A challenge we encountered while developing this tool–also encountered in PyRT-Fuzz [72]–was how to marshal and unmarshal data between actual Python objects and a byte sequence that Atheris can effectively mutate. Pickle [98], which is Python's standard serialization module included in its standard library, seemed like an obvious choice. However, we quickly realized that pickle objects contain a large proportion of metadata, even for simple objects (e.g. on Python 3.8.3, pickling the string "hello" takes up 20 bytes, a 1:3 data-metadata ratio) and pickled data does not actually flatly encode whatever data the original object contained. Rather, pickled data is a series of opcodes for a virtual machine, called the "Pickle Machine," that instructs the machine on how to reconstruct the pickled object [120]. It is no surprise then that in our results, we find that pickle only succeeds in decoding inputs mutated by Atheris **8.3%** of the time.

Another strong deterrent to using pickle in our tool is the fact that there are known security risks, including the risk of arbitrary code execution, when unpickling untrusted data [120]. Python acknowledges these security risks by stating that developers should not attempt to unpickle untrusted data [98], but in practice, developers often do. In fact, the practice of sharing pickle files has become commonplace in the machine learning community, and security vulnerabilities in this practice are beginning to be revealed and discussed [65, 127, 33, 120, 107]. If we were to continue using pickle in the tool's implementation, adoption of the tool could be harmed due to these vulnerabilities. All of these considerations motivated us to develop our own encoding scheme.

We set out to meet 3 design goals in our custom encoding:

- G1: the encoding should represent metadata (that is, information about the *shape* of the Python object) separately from the data (the primitive values that make up the object).
- G2: for any metadata, any data section should be decodable into a Python object that fits the shape described by the metadata.
- G3: the encoding should be as versatile as Pickle for Python objects in use cases beyond just fuzzing.

Why G1? This property is particularly useful for fuzzing since, with a small set of modifications to libFuzzer [74], it allows the fuzzer to focus solely on mutating the primitive values of a Python object. Primitive values, such as strings and integers, are comprised of bytes of data, so this aligns with mutational fuzzers' strength in randomly and quickly mutating byte streams. The concern of mutating the data in a way that makes sense in the context of a certain data structure or grammar is handled by the encoding. This implies that the fuzzer will never be able to change a seed input's predefined *shape* (Definition 1), but given that the ACL2s type model provides us with a wide variety of possible input shapes, we believe that we can get away with not relying on the fuzzer to explore the metadata space.

Why G2? This property is also useful for maximizing the use of the fuzzing time budget. Instead of wasting fuzzing cycles on mutated inputs that just get rejected, we can make the best out of any input that is given.

Why G3? Our encoding is useful in any other fuzzing application for Python, but given the security vulnerabilities and obscure storage format of pickle, we are motivated to create an encoding that can serve as a substitute for pickle in general use.

6.2 Design

At the highest level, strings in our encoding contain two components, separated by a *metadata separator*, which is a 4-byte special sequence. The bytes to the left of the separator make up the *data* portion, and the bytes to the right make up the *metadata* portion. There is great flexibility in how the metadata is stored as long as it captures all information about the data's *shape* (Definition 1), but for the data section, we prescribe a rigid format. We define another 4-byte special sequence, the *data separator*, which separates individual components of the data. In summary, a string that is produced by the encoder can be broken down into *metadata* which describes how the various *data components* in the data section can be reconstructed into a Python object.

Definition 1. *A **shape** is the structure of a Python object. It is a tree where each non-leaf vertex is a complex Python object that contains other Python objects. Each leaf vertex represents a primitive type (i.e. a type that is atomic), annotated with the type of value that should be placed there in the structure. For container types such as lists, which may contain arbitrary amounts of objects, the shape of such objects encodes the exact amount of contained objects that are present in a particular instance. Figure 6.1 contains an example of the shape of a list of integers, of length 3.*

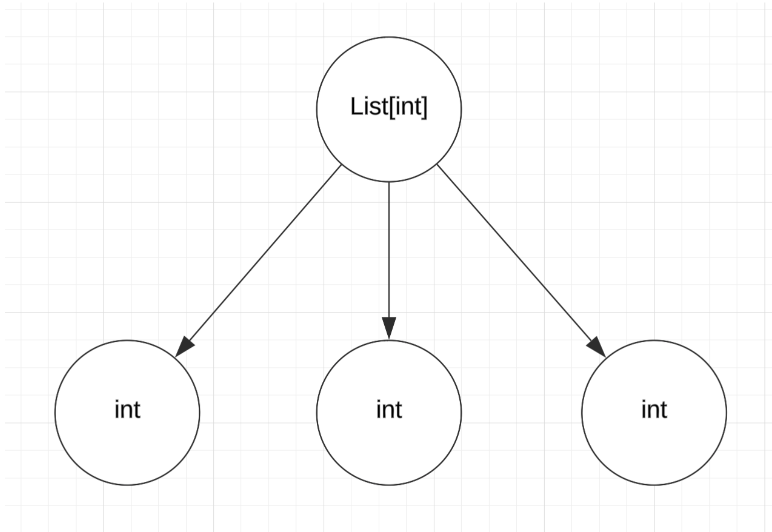

FIGURE 6.1: Conceptual example of a *shape*

We believe this encoding design satisfies G1 and G2, and has excellent potential to satisfy G3 with future work:

- G1: The metadata and data of our encoding are separated by definition.
- G2: As we will demonstrate in the following section, G2 holds by construction of our implementation.
- G3: The encoding design supports this goal, provided that the type system is complete enough to support encoding almost all Python objects that one would want to encode. Additional work will be required to extend our model of the type system before this can be fully realized, but the encoding format will not need to change.

Our encoding design has the added advantage that the length of the data portion will grow independently of the level of nesting in the structure of the Python object (this is not necessarily true of the metadata, and it is not true based on the metadata schema we have chosen, but there are opportunities to improve). A key disadvantage of our encoding is that it is still possible to tamper with the metadata to change the structure of the object that is decoded from a given set of data. However, since the type of the object needs to be known at decode-time and is not stored in the encoded bitvector, there is no opportunity for the actual type of value decoded to change by modifying the encoded data. And most importantly, unlike *pickle*, no part of our encoding exposes an instruction set that can be used to execute arbitrary code.

In the next section, we cover the specification of our implementation of this design.

6.3 Encoding Specification

To encode a Python object, we must separate the data in the primitive "slots" from the shape of the data (Definition 1). The set of primitives in Python are given in Definition 2. To do this, we must construct a *primitive tree*. Thus, the first step in the encoding algorithm is primitive tree creation. This process is given in Algorithm 8.

Definition 2. *We define the* ***primitive types*** *as the following subset of Python primitives:*

$$\{None, str, int, float, bool, bytes\}$$

If the given type is a primitive, we simply create a leaf node (lines 2-3). If the type is a list-like container type (lines 4-6), we create a node that has as children the primitive trees of its elements (in the pseudocode, we use a Python-like list-comprehension syntax). If the type is a dictionary, we create a node whose children are the interleaved primitive trees of its keys and values (lines 7-12). If the type is a class, we obtain a list of pairs of field names and types, sorted alphabetically by field name (lines 13-15). We then produce a node with the primitive trees of the field values as children, sorted by field name (lines 16-17).

Algorithm 8 Primitive Tree Creation

```
1: function CreatePrimitiveTree(value, type)
2:     if type is primitive then
3:         return PTNode(value, type, [])                    ▷ "PTNode" is a primitive tree node
4:     else if type is one of {list, fixedtuple, variadictuple, set} then
5:         children ← [CreatePrimitiveTree(e, type.inner_type) for e in value]
6:         return PTNode(None, type, children)
7:     else if type is dictionary then
8:         pairs ← value.items()
9:         keyChildren ← [CreatePrimitiveTree(k, type.key_type) for k, _ in pairs]
10:        valChildren ← [CreatePrimitiveTree(v, type.value_type) for _, v in pairs]
11:        children ← Interleave(keyChildren, valChildren)
12:        return PTNode(None, type, children)
13:    else if type is class then
14:        field_dict ← GetFieldDictionary(type)
15:        sorted_fields ← Sort(field_dict.items(), pair → pair[0])
16:        children ← [CreatePrimitiveTree(name, t) for name, t in sorted_fields]
17:        return PTNode(None, type, children)
18:    else if type is union then
19:        inner_type_index ← GetUnionArgIndex(value, type)
20:        inner_type ← union.args[inner_type_index]
21:        children ← [CreatePrimitiveTree(value, inner_type)]
22:        return PTNode(inner_type_index, type, children)
23:    else
24:        error
25:    end if
26: end function
```

We define the set of primitive types as shown in Definition 2. Algorithm 9 shows the encoding for primitive types, while Algorithm 10 shows the decoder. The definition of the primitive decoder allows our encoding to satisfy G2, since it handles arbitrary streams of bytes.

In these two algorithms, NoneType is encoded with a placeholder, NoneValue, as the type has a single value. In decoding, `None` is unconditionally returned because the underlying byte sequence does not matter (Algorithm 10, lines 2-3). Integers are arbitrary precision in Python, which means that we can encode and decode arbitrary-length integers and byte sequences. Integers are represented as signed big-endian

Algorithm 9 Primitive Encoder

```
function ENCODEPRIMITIVE(primitive_value, type)
    if type is NoneType then
        return NONEVALUE
    else if type is int then
        return TOBYTES(primitive_value, 'int')
    else if type is float then
        return TOBYTES(primitive_value, 'float')
    else if type is bool then
        return 0xF0 if primitive_value is True else 0xF1
    else if type is str then
        if primitive_value == "" then
            return EMPTY
        end if
        return encoding primitive_value as UTF-8
    else if type is bytes then
        if primitive_value == b"" then
            return EMPTY
        end if
        return primitive_value
    else
        error
    end if
end function
```

Algorithm 10 Primitive Decoder

```
function DECODEPRIMITIVE(byteSequence, type)
    if type is NoneType then
        return None
    else if type is int then
        return FROMBYTES(byteSequence, int)        ▷ ints are arbitrary precision in Python
    else if type is float then
        paddedFloat ← LEFTPAD(byteSequence, 0x00, 8)
        return FROMBYTES(paddedFloat[0:8], float)
    else if type is bool then
        return is POPCOUNT(byteSequence) even?
    else if type is str then
        if primitive_value == EMPTY then
            return ""
        end if
        return decode primitive_value as UTF-8, replacing any errors with a placeholder
    else if type is bytes then
        if primitive_value == EMPTY then
            return b""
        end if
        return byte_sequence
    else
        error
    end if
end function
```

values. The decision of big-endian is arbitrary. Encoding and decoding integers is implemented with Python's built-in marshaling functions, represented in Algorithms 9 and 10 as TOBYTES and FROMBYTES. Floating point values in Python are double precision, so they are stored in eight bytes. To encode them, we emit the eight underlying bytes using Python's built-in marshaling function, represented again as TOBYTES on line 7 of Algorithm 9. Decoding of floating point values takes place on lines 6-8 of Algorithm 10. If the byte sequence is less than eight bytes in length, we first pad it to the left with zeros up to a length of eight, and then we decode the byte sequence as if it were a float. If the byte sequence is greater than eight bytes in length, we clip off the remaining bytes after the eighth, and decode only the first eight bytes.

Algorithm 11 Emit Bytes from PTNode

```
1: function EMITBYTES(node : PTNODE)
2:     metadata_sexpr ← TREETOSEXPR(node)    ▷ Converts the shape information of the tree into an
   s-expression representation
3:     leaves ← COLLECTLEAVES(node)    ▷ Collects leaves from left-to-right
4:     encodedPrimitives ← [ENCODEPRIMITIVE(leaf.value, leaf.type) for leaf in leaves]
5:     dataBytes ← JOIN(encodedPrimitives, DATASEPARATOR)
6:     return JOIN([dataBytes, metadataSexpr], METADATASEPARATOR)
7: end function
```

Now that we can encode and decode primitive types, and we can construct a primitive tree, we need to create the final byte stream. That is covered in Algorithm 11. TREETOSEXPR (line 2) is a function that converts the shape information encoded in the tree into an s-expression. The grammar of these s-expressions is as follows:

$$
\begin{aligned}
METADATA\_SEXPR ::= {} & \texttt{symbol} \\
& \mid (\ INNER\_SEXPR\ ) \\
& \mid (\ \texttt{integer}\ INNER\_SEXPR\ )
\end{aligned} \tag{6.1}
$$

$$
\begin{aligned}
INNER\_SEXPR ::= {} & METADATA\_SEXPR\ \text{" "}\ INNER\_SEXPR \\
& \mid METADATA\_SEXPR
\end{aligned} \tag{6.2}
$$

The first production rule (Equation 6.1) has three branches. The first is a symbol, which is the name of a primitive slot. The symbol PyCgenInteger, for instance, tells the decoder that an integer is supposed to go into this primitive slot. The second branch specifies the list case for an s-expression. The third branch specifies a special case expression which is used to encode union types. The <integer> value represents the index into the array of arguments to the union. The second production rule (Equation 6.2) specifies how the s-expression list's elements are whitespace-separated. Once the s-expression containing the shape information is generated, the COLLECTLEAVES function (line 3) collects the leaves of the PTNode tree, from left to right. Then, on line 4, ENCODEPRIMITIVE (defined in Algorithm 9) is used in a list comprehension over the primitive objects to generate the data stream. Finally, in lines 5-6, the encoded bitvector is assembled.

To begin the decoding process, the given bitvector is split into its data and metadata portions. The data portion is further split by the DATASEPARATOR, and this split array of data segments becomes the *dataArray* provided in Algorithm 12. The metadata is parsed according to the grammar given in Equations 6.1 and 6.2. The parsed metadata becomes a primitive value, or list, depending on the contents of the expression, and it is given as the *metadata* argument in Algorithm 12. The second input is the target type we are decoding into. This is required because the metadata, although it contains the shape information, does not have information about the source type. This feature is useful as an extra validation step when decoding, so that arbitrary information is not decoded. The third input is the metadata segment, after it has been parsed from s-expression form into Python nested lists. GETRANK (line 2) is a function that produces the rank of a given metadata. A shape's rank is the number of primitive "slots" it has available to fill. For instance, a shape that represents a list of integers of length 5 would have a rank of 5. ROUNDROBINEXTEND (line 3) is a function that takes a list,

Algorithm 12 Decode Into PTNode

```
function DecodeIntoPTNode(dataArray, type, metadata)
    rank ← GetRank(metadata)
    primitivesSlots ← RoundRobinExtend(dataArray, rank)
    if type is primitive then
        decodedPrimitive ← DecodePrimitive(dataArray[0], type)
        return PTNode(decodedPrimitive, type, [])
    else if type is list, fixedtuple, variadictuple, or set then
        children ← [], offset ← 0
        for innerShape in metadata do
            innerRank ← Rank(innerShape)
            dataSlice ← dataArray[offset : offset + innerRank]
            children.append(DecodeIntoPTNode(dataSlice, type.innerType, innerShape))
            offset ← offset + innerRank
        end for
        return PTNode(None, type, children)
    else if type is dictionary then
        children ← [], offset ← 0
        for pairShape in metadata do
            keyRank ← Rank(pairShape.key)
            valueRank ← Rank(pairShape.value)
            keyDataSlice ← dataArray[offset : offset + keyRank]
            key ← DecodeIntoPTNode(keyDataSlice, type.keyType, pairShape.key)
            offset ← offset + keyRank
            valueDataSlice ← dataArray[offset : offset + valueRank]
            value ← DecodeIntoPTNode(valueDataSlice, type.valueType, pairShape.value)
            offset ← offset + valueRank
            children.append((key, value))
        end for
        return PTNode(None, type, children)
    else if type is class then
        children ← []
        fieldDict ← GetFieldDictionary(type)
        sortedFields ← Sort(fieldDict.items(), pair → pair[0])
        for (name, fieldType) in sortedFields do
            fieldShape ← metadata[name]
            fieldRank ← GetRank(fieldShape)
            fieldNode ← DecodeIntoPTNode(dataArray[0 : fieldRank], fieldType, fieldShape)
            dataArray ← dataArray[fieldRank :]
            children.append(fieldNode)
        end for
        return PTNode(None, type, children)
    else if type is union then
        innerTypeIndex ← metadata[0]
        innerType ← type.args[innerTypeIndex]
        innerNode ← DecodeIntoPTNode(dataArray, innerType, metadata[1 :])
        return PTNode(innerTypeIndex, type, [innerNode])
    else
        error
    end if
end function
```

and pads it up to the given length by taking elements in a circular fashion starting at the front of the list. For instance, round-robin extending the list $[1,2,3]$ to length 5 would yield $[1,2,3,1,2]$. The purpose of this logic is to allow for separators in the data section of the encoding to be erased with no effect on the decodability of the data. If separators are instead added into the data, either by change or through a deliberate action of the mutation algorithm, the extra primitives will be ignored during decoding. This is necessary so the fuzzer is not interrupted if an input fails to decode; since our encoding never fails by design, our hope is that the fuzzer will have many more chances to mutate something interesting out of an encoded string. The remainder of the decoding function given in Algorithm 12 branches into different cases based on the given type, and the rank information of the various sub-metadata is used to place the primitive slots as children of the proper nodes. Note the use of DECODEPRIMITIVE on line 5 from Algorithm 10.

The final step in the decoding algorithm is value reconstruction. For the sake of brevity, we omit the pseudocode for this step, since it is a straightforward recursive algorithm which takes a tree constructed in Algorithm 12 and produces the corresponding Python object.

6.4 Modifications to libFuzzer

In order to better support fuzzing using our encoding, we modified libFuzzer to treat the metadata section of our encoded bit streams as separate from the data, so that libFuzzer can focus its mutation effort on the data only. Here, we give a summary of the modifications that were made to libFuzzer.

In libFuzzer's codebase, there is a class called `MutationDispatcher` (the definition is given in `FuzzerMutate.h`). We added two new methods to this class: `StripMetadata` and ReplaceMetadata. The definition of StripMetadata is given in Listing 6.1.

StripMetadata locates the aforementioned *metadata separator* in the current unit, and returns a new value for the size of the unit that includes all bytes up to the separator. This essentially hides them from the fuzzing process, since the mutation dispatcher uses this size value to bound the array of bytes that comprise the fuzzing unit. Information about how many bytes were hidden, and whether bytes are currently being hidden in the first place, are stored as private instance variables on the MutationDispatcher class, so the metadata can be restored into the encoded bitvector later. The maximum unit size that was used in all experiments performed in this evaluation was 1 MiB (1,048,576 bytes). The effective size of the fuzzer's mutation surface, therefore, is 1 MiB - size_of_metadata_in_bytes. As explained previously, metadata values are stored as uncompressed s-expressions that exactly mirror the *shape* of the encoded Python object. For very large objects, this metadata can also grow quite large, and there is an opportunity to compress this in future work.

```
/**
 * Strips out metadata from the given data,
 *   and places it into this object for storage
 * Returns the new size of the data buffer.
 *
 * MaxLen - (Size - NewSize) = New Max Mutation Length
 */
int StripMetadata(uint8_t *Data, int Size);
```

LISTING 6.1: StripMetadata definition in FuzzerMutate.h

ReplaceMetadata, as one might expect, *replaces* the metadata by unhiding it after it has been hid with StripMetadata. The declaration of this function is given in 6.2.

```
/**
 * Replaces metadata into the given data
 *   from this object's internal store.
 * Returns the new size of the data buffer
 */
int ReplaceMetadata(uint8_t *Data, int Size, int MaxSize);
```

LISTING 6.2: StripMetadata definition in `FuzzerMutate.h`

6.5 Related Work

6.5.1 Encodings

Li et al. [72], for their fuzzer PyRTFuzz, developed an input encoding to address the problem of fuzzing typed function contracts, similar to our use case. Their goal was to be able to generate type-correct values to be passed into the arguments for each of their generated Python "APPs," or fuzzing harnesses. In their implementation, they use `FuzzedDataProviders` to generate random values of primitive types, such as strings, integers, and floats. The PyRTFuzz encoding also support some complex types, such as lists and dictionaries, but unlike our implementation, they do not support arbitrarily nested complex types. They also do not support automated extension of the type system with user-defined types. Lists can only contain primitive strings, for instance. Then, these values are packed together into a single bitvector, separated by a 4-byte data separator, similar to how we separate the primitive values in the data portion of the encoding. Our implementation supports the encoding of more diverse and complex Python objects, and we also leverage the custom enumerator capabilities of ACL2s, as discussed in Chapter 3, as opposed to the use of `FuzzedDataProviders`.

6.5.2 Vulnerabilities of Pickle

Kathikar et al. [65] perform an assessment of models listed on the Hugging Face platform to identify sources of weakness. They find that some of the most popular parts of the AI software supply chain, such as *transformers* [128], are susceptible to arbitrary code execution due to deserialization of untrusted pickle data, which is a common storage format used in the AI community to perform model exchange. Tidjon and Khomh [127] also flag pickle as a common arbitrary code execution vulnerability. The authors found that a contributor to PyTorch, a popular open-source AI/ML project, regrets introducing pickle into the codebase as a way to exchange models.[1] Sultanik [120] provides a deeper look into why pickle is so vulnerable to arbitrary code execution, and provides a tool to decompile pickle files.

[1] https://github.com/pytorch/pytorch/issues/52181

7 Evaluation

In this chapter, we develop the method of our experimental design, and present the results of our comprehensive evaluation of the type hint fuzzer.

7.1 Experimental Methods

To the best of our knowledge, this tool is the first to bridge the gap between fuzzing and arbitrary (annotated) Python code, similar to how PyRTFuzz [72] is the first fuzzer for the Python interpreter. Like PyRTFuzz, we do not have a direct baseline against which to compare; our focal evaluation question is *"Is this tool effective?"* rather than *"Is this tool more effective than X?"*. Therefore, to evaluate this tool, we focus on understanding 1) how to configure the tool for the best results, and 2) whether issues found by the tool represent feedback that is relevant and actionable for developers.

To understand the best configuration, we first perform a comparative evaluation between the custom encoding developed in Chapter 6 and the baseline of *pickle*, so that we may choose the encoding that better facilitates fuzzing. Then, we evaluate how various configurations of the tool's tunable hyperparameters (other than input encoding) affect fuzzing performance as measured by code coverage and statistics about crashes found. Finally, to evaluate the practical value of the issues this tool finds, we solicit feedback from repository maintainers on a sample of the issues found.

In the evaluation of the various tool configurations, we take inspiration from a history of fuzzing works where the core fuzzer is augmented by another tool (or occasionally, another fuzzer) to enhance progress [72, 139, 138, 134, 36, 140, 23, 119]. A core evaluation goal, then, which goes together with our design goal of extensibility, is to understand how best to configure this tool to integrate well with other third-party tools and analyses. We therefore optimize for quick and comprehensive exploration of the state space, so that when a fuzzing "wall" is hit, the next step in the pipeline will have access to a large amount of information.

In light of the motivation given in Chapter 2, we did also consider comparing the found issues against the issues static type checkers, such as mypy, emit for the repositories-under-test, but we felt that this would be unfair for the following reasons:

1. We do not claim that the type hint fuzzer can replace static type checkers entirely. Rather, we hope to complement static type checkers by addressing the deficiencies outlined in Section 2.2.

2. By definition, the set of issues that the type hint fuzzer finds is largely disjoint from the set of issues that static type checkers can find. The obvious overlap is identification of when a function's return type is incorrectly annotated. There may be other instances where some of the crashes found by the type hint fuzzer are caused by typing issues that would also be found by mypy, but for the rationale given in the previous item, we consider this to be out of the scope of this work.

In the next section, we cover the research questions that we will be asking in order to answer the question of whether this tool is effective. We also rigorously define the metrics and bug detection methodology.

7.2 Research Questions

In evaluating this tool, we set out to answer the following research questions.

7.2.1 RQ1. How does the custom encoding compare to *pickle* in facilitating effective fuzzing?

In Chapter 6, we delve into the custom input encoding that we designed to address shortcomings of Python's pickle object serialization (hereafter denoted *pickle*) [98]. To fulfill the goal of understanding how to best configure the tool for optimal fuzzing performance, we perform a comparative evaluation between the custom encoding (hereafter denoted *custom*) and *pickle*. To ensure fairness, we compare across a suite of metrics, and perform statistical tests where necessary to understand the significance of the results.

The metrics we compare are post-mutation decode rates, unique point and total point counts, coverage growth over time, the Time-to-Knee (TTK) of coverage growth, Coverage At Knee (CAK) compared across tool configurations, the ratio of CAK to TTK, to summarize how much exploration is completed once the first "wall" is hit–that is, the point at which fuzzing progress slows–and how quickly the fuzzer gets there into a single metric. These metrics, and their justifications, are given as follows.

Post-Mutation Decode Rate

In this first comparison of *custom* to *pickle*, we measure the proportion of mutation cycles during fuzzing that are are "wasted" on inputs that do not decode properly. Inputs do not decode properly when libfuzzer's mutation corrupts the formatting of the data, which motivated the design of *custom* in the first place (Chapter 6). We call this proportion the "post-mutation decode rate," which for brevity we occasionally shorten to "decode rate." Decode rate is the ratio of successful decodings of an input after the current unit has been mutated by whatever strategy libfuzzer decides to use, when there is at least one element present in the corpus. We only count when there is a non-empty corpus because when this holds, libfuzzer will pick an element from the corpus to start mutating, rather than attempting to create an input from nothing. This provides a clearer picture of how often a decodability of an input stream is preserved during mutation.

The formal definition is given in Equation 7.1. This equation represents the decode rate, under some random seed, n, for the given configuration C and function F. S_{total} is the number of successful decodings measured in total. $S_{with\ no\ corpus}$ are the number of successful decodings when there are no elements in the corpus. In our observations, this value is typically zero or near zero. T_{total} is the total number of decoding attempts, while $T_{with\ no\ corpus}$ is the total number of decoding attempts when there are no elements in the corpus.

$$DR_n(C, F) = \frac{S_{total} - S_{with\ no\ corpus}}{T_{total} - T_{with\ no\ corpus}} \tag{7.1}$$

Unique Points and Total Points

In the second comparison of *custom* to *pickle*, we measure unique and total points during fuzzing campaigns under each encoding. The definition of fuzzing point and "unique points", and "total points" are given in Definition 3. While there is no precedent for this type of comparison, to our knowledge, in the literature, we find that counting total points is a useful way to understand the amount of fuzzing volume that each encoding facilitates. By comparing the ratio of unique points to total points, the relative proportion of time spent during fuzzing on exploring new parts of the state space is also revealed.

Definition 3. *A* ***fuzzing point*** *is an input-output pair (analogous to a point on an x-y coordinate plane, where y is a function of x). During a fuzzing campaign, points that are emitted need not be unique. The set of all points emitted is referred to as the set of* ***unique points****. The volume of points emitted during a fuzzing campaign is referred to as the quantity of* ***total points****.*

It is worth noting that this metric is not guaranteed to be reproducible when fuzzing on a timeout, since fluctuations in the speed at which the program runs on the testbed machine may affect this value.

Coverage Growth

In previous work related to fuzzing, understanding how a fuzzer's coverage of the code under test (CUT) evolves overtime has proven useful in measuring fuzzer performance [72, 59, 57, 36]. It is useful for visually understanding the relative coverage performance of tool configurations, and for identifying when the coverage "wall" is getting hit, after which adding additional time budget may offer diminishing returns. This metric is defined as the number of statements covered in the union of the sets of statements of the bodies of all functions that were successfully fuzzed, as a function of time. The y-axis of these charts are given in number of statements covered.

In order to construct the graph, we use the data produced during the "coverage replay" phase of the tool's operation, described in Section 5.1.3. This phase produces a set of "coverage points," which are ordered pairs of the form (time, lines covered). To account for variability in independent trials, we incorporate all five trials' results by computing the arithmetic mean at evenly spaced ..., and present 95% confidence interval bounds above and below the main line.

The algorithm to generate the graph is as follows: first, we generate equally spaced values along the time range of the fuzzing campaign, $[0, 440]$. Let T be the set of these values. If there are n bins, then $T = \{0, \frac{440}{n}, 2 * \frac{440}{n}, 3 * \frac{440}{n}, \cdots, (n-1) * \frac{440}{n}, 440\}$.

The definition of $CumCov$ is given in equation 7.2. For a given experimental configuration, c, set of functions F, random seed s, and bin cutoff b, this function produces the ratio of lines covered across function bodies in F under the given experimental configuration and seeds, utilizing all coverage chunks whose timestamps are the greatest that are less than or equal to b. $CovPoints$ returns the set of coverage samples associated with the given function f, configuration c, and random seed n. These chunks are represented as 2-tuples, with the first element, written as t' in Eq. 7.2, being the relative timestamp in the fuzzing campaign at which the coverage sample was obtained, and the second element, written as cov, being the number of statements in the function's body that have been covered covered. By definition of the coverage replay algorithm (Section 5.1.3), coverage increases monotonically with respect to time, which allows us to use max on the set of coverage ratios to get the coverage value that corresponds with the latest bin.

$$CumCov(F, c, t, s) = \sum_{f \in F} \max\{cov \mid (t', cov) \in CovPoints(f, c, s) \wedge t' \leq t\} \quad (7.2)$$

To combine results across independent trials, we use Equation 7.3 to produce the final graph. This function produces the arithmetic mean of cumulative coverage across the independent trials corresponding to random seeds in the set S.

$$CumCovGrowth(F, c, t) = \frac{1}{|S|} \sum_{s \in S} CumCov(F, c, t, s) \quad (7.3)$$

In the visualizations of these coverage curves, following previous work in Grissom and Kim [46] and Herrera et al. [59], we also include 95% confidence intervals, computed with the bootstrap method applied to the sample sets of which we are taking the arithmetic mean in Equation 7.3. In our evaluation, $|S| = 5$, which is not a large sample size, but our goal with this metric is not to perform statistical analysis; we simply use the confidence intervals to visualize variability across the independent trials.

Knee, Time-to-Knee, Coverage At Knee

When two systems or parts of a pipeline need to be integrated, such as the Level-1 and Level-2 fuzzer in PyRTFuzz [72], or AFL and the concolic execution engine in Driller [119], some decision procedure is required in order to determine when the cutover between one part and another should take place. In the case of PyRTFuzz, this cutover is based on a predetermined time budget [72], and in the case of Driller, the cutover to concolic execution occurs when the fuzzer has gone through a predetermined number of mutation cycles without new coverage, proportional to the input length [119]. Inspired by these works, and inspired by the common intuition of the "knee" of a curve, we develop a method of identifying when the type hint fuzzer is stuck. This is important to measure because it allows us to achieve our stated evaluation goal of understanding how to best configure the fuzzer to achieve fast and comprehensive state space exploration, so that integration with other tools and analyses can be effective. To this end, we introduce three metrics, Time-to-Knee (TTK), and Coverage At Knee (CAK), and their ratio, CAK/TTK.

Definitions of knees of curves exist in the literature, such as the one given by Demetris T Christopoulos [26], which is based on the definition of curvature of a function. However, given the problem we are trying to solve by detecting the knee, we saw it fitting to create a new definition for the "first knee" of a curve that is more in line with the notion that a fuzzer is "stuck" when significant progress has stopped for a certain "long" (on the order of seconds-minutes) amount of time. Algorithm 13 gives our definition for how we detect knees in curves (this definition is general, and shared across all types of curves we may examine in this work). $lookBehind$ is the size of the window of samples to evaluate when looking for a knee, in seconds. We choose 60 seconds because in modern computational contexts, a minute is a long time for performance to stall. So that each $lookBehind$ window covers the same number of samples across all functions, we perform the following derivation.

Let s be the number of samples we want to capture in each $lookBehind$ window. Let F be the set of fuzzable functions in some repository; the number of function candidates therefore is $|F|$. The time spent fuzzing each function candidate is $T = 440$ seconds. If we want s samples to be present in each window of size $lookBehind$, then we need $N_s \frac{lookBehind}{T} = s$ to be true, where N_s is the number of samples, or equally spaced points in time at which to compute total coverage based on the chunks that were logged up to that point. By elementary algebra, and accounting for different time units, we derive Equation 7.4.

$$N_s = \frac{440s}{lookBehind} \tag{7.4}$$

We choose $s = 10$, as a balance between too small of an s leading to lack of resolution in the curves, and too high of an s leading to unnecessarily large computation times. $slopeThreshold$ is the threshold below which the slope between two samples is considered to be indicative of a lack of progress. Its unit is *new lines covered / second*. It is set to 0.5. We choose 0.5 because the typical pattern that we observe (see Figures 7.5, 7.6, and 7.7) is that there is a short burst of high coverage gain at the beginning of the fuzzing campaign ($slope \gg 1$), and this slope quickly decreases to < 1. Choosing a slope that is too small would lead to very late detection of this dropoff, and choosing one that is too large (too close to 1) could lead to detection of the knee while it is still forming in some cases, so we choose 0.5 as a reasonable compromise.

Algorithm 13 Knee Detection

function FINDKNEE(X, Y)
  $lookBehind \leftarrow 60$ ▷ detect knees if progress has stalled for 1 minute
  $slopeThreshold \leftarrow 0.5$
  for idx **in** $[0, 1, \cdots, |X| - 1, |X|]$ **do**
    $coords \leftarrow [(x, y)$ **for** x, y **in** $\text{zip}(X, Y)$ **where** $X[idx] - lookBehind \leq x \leq X[idx]]$
    $slopes \leftarrow [\frac{coords[j].y - coords[j-1].y}{coords[j].x - coords[j-1].x}$ **for** j **in** $[1..|coords|)]$
    if $|slopes| > 0$ **and** $X[idx] \geq lookBehind$ **and** all $slopes < slopeThreshold$ **then**
      return idx, x
    end if
  end for
  return no knee
end function

TTK is the first of the metrics that is computed with this algorithm. It is the value of the timestamp at which FINDKNEE identifies a knee. In the results presented later in this chapter, we compare Times to Knee across tool configurations to understand the impact of corpus size (see Section 5.1.2) and input encoding on how quickly the fuzzing campaign hits the knee.

CAK is the amount of code coverage at the time of knee, as identified by measuring TTK according to the definition given previously. It is measured for a given function, f, under configuration c, time-to-knee k, and random seed n, as given in Equation 7.5, which references the definition of $CumCov$ from Equation 7.2.

$$CAK_{f,c,k,n} = CumCov(\{f\}, c, k, n) \tag{7.5}$$

Finally, the **CAK/TTK** ratio is simply computed by dividing the coverage at the knee by the time at which the knee was found. In terms of the equations given previously, this is defined as follows in Equation 7.6, where k is the time-to-knee for function f under configuration c and random seed n.

$$CAK/TTK_{f,c,k,n} = \frac{CAK_{f,c,k,n}}{k} \tag{7.6}$$

When comparing tool configurations, we examine the means of these values across all independent fuzzing campaigns (functions fuzzed across repos * 5 indepedent trials). We also perform hypothesis tests of difference in distribution with the Mann-Whitney U test, as recommended by Klees et al. [70] and Arcuri and Briand [3], and we compute effect size with Vargha and Delaney's $\hat{A}_{12}$ [131].

7.2.2 RQ2. How does the configuration of the tool affect fuzzing performance?

The ratio of use of fresh inputs generated by ACL2s and inputs produced by mutation of old inputs is configurable in the tool. Coverage-guided mutation can also be controlled by adjusting corpus size. For this RQ, we explore how to achieve good fuzzing performance by tuning these parameters.

Inspired by previous fuzzing works, we measure performance in code coverage, crashes found. We also perform a CAK/TTK (see Section 7.2.1) analysis to understand what configuration of the tool best satisfies the evaluation goal of understanding how to configure the tool to quickly and comprehensively explore the state space. Finally, following previous work [3, 59], we perform a survival analysis of crashes found, which will be described later in this section.

For brevity, we define a notation to describe experimental configurations in this RQ. The form <corpus_size>_<acl2s_reachout_freq> describes an experimental configuration with the corpus size, $|C|$ and `acl2s_reachout_freq` as given. Section 5.2.4 contains the definition of `acl2s_reachout_freq`. For example, `100_0.1` describes a configuration with $|C| = 100$, and `acl2s_reachout_freq` $= 0.1$. When the configuration string is preceded by a repository name, e.g. `mypy_100_0.1`, this indicates that the associated results and data pertain only to the fuzzing activity from that repository.

In the following subsections, we introduce our methodology for crash deduplication and filtering, which will inform the discussion of survival analysis methodology afterward.

Crash Deduplication

Klees et al. [70] explore the pitfalls of two common bug deduplication methods: stack hash-based deduplication and coverage profile-based deduplication. Both of these approaches may not accurately reflect the actual "bugs" in a program, and the authors recommend counting "ground truth bugs," which map crashing inputs before some patch P that cease to cause crashes after P. Due to time and resource constraints, however, we do not have access to ground truth bugs, so we instead generate a set of *crash classes*, defined in Definition 4.

Definition 4. *A **crash class** is a family of exceptions or abnormal termination behaviors–e.g. segfault. Membership in the class is determined by the last N stack frames that refer to code locations in the code-under-test, as well as the failure type.*

We make no upfront claim that each crash class corresponds to a bug, unless explicitly indicated as such by a developer. To produce a best-effort count of crash classes, we use a stack trace-based deduplication, with some additional filtering, where we index a crash using N of the last traceback elements along with the type of exception that was thrown (we did not encounter any abnormal program terminations in this evaluation). In this evaluation, we choose $N = 3$, because since we are fuzzing libraries *bottom-up*, the call stacks of crashes tend not to be very deep.

We also perform some other preprocessing steps of the exception call stacks. If a frame is repeated many times, Python will collapse the repeated frames with a line that says `[Previous line repeated N times]`. In cases where this line appears in the last few lines of the stack trace printout, and N is different, this causes the stack traces to mismatch. So, we "unroll" this line before comparing.

Crash Filtering

The type hint fuzzer has managed to identify many crashes in the 5 repositories selected for RQ2 and RQ3. To pare the list of crashes down to something that is more representative of legitimate bugs, we apply two heuristics. First, we exclude all crashes that originated from intentionally thrown exceptions. We check this by looking for the keyword "`raise`", which is used to trigger an exception manually, in the last line of the exception traceback. Second, we filter out exceptions with types in a predefined list that we consider to be likely to be due to a filesystem configuration or external environment that the code did not expect. Currently, we filter out the following set of exceptions: `{SystemExit, ModuleNotFoundError, FileNotFoundError, UnicodeEncodeError}`. `SystemExit` is raised when user code intentionally terminates the program, so reporting this as *un*intended behavior typically does not make sense. However, it is true that `SystemExit` could be raised under the wrong circumstances, so there is a case to be made for removing this from the filtered set in future evaluation. `ModuleNotFoundError` is raised when a module cannot be found and imported. We attempt to install dependencies of the repositories that we evaluate against based on a `pyproject.toml` file if it exists, but there may be hidden challenges with replicating the development environment for the package that we did not investigate. `FileNotFoundError` is raised when a file on disk is not found. This is typically not a crash that warrants investigation because this is a normal signal to a developer that something is misconfigured in the environment; for this reason, we do not report it. `UnicodeEncodeError` is an exception that is often triggered by *pickle*, but our custom encoding does not due to the way they handle Unicode strings differently. We are continuing to investigate how to modify our encoding to handle Unicode strings similarly to *pickle* in order to trigger these exceptions. These exceptions typically occur when attempting to print a malformed Unicode string.

Crash Survival Analysis

An important characteristic of fuzzer performance is how quickly crashes are found. The time it takes to find a crash is known as how long that crash "survives" [59]. The survival analysis we perform has two parts. First, following in the footsteps of previous work [59, 12, 3, 131, 64], we compute Kaplan-Meier survival curves, with confidence intervals [64]. Secondly, we compute hypothesis tests of difference in Restricted Mean Survival Time (RMST). RMST is essentially the integral of the survival function, and can be regarded as a cumulative amount of "survival" represented by the curve. We choose to compare RMST because previous work in the medical field [51, 110, 8] has recommended the use of this metric as a better summary metric for survival comparison. Unlike the log-rank test which is used in Herrera et al. [59], comparisons of restricted mean survival time do not assume proportional hazards [51]. To statistically compare approaches, we use the `survRM2` R package [22] to perform hypothesis tests of the difference between RMST. We present two-tailed hypothesis tests of difference in RMST, pairwise by configuration, with p-values corrected with the Holm-Bonferroni method [60].

7.2.3 RQ3. Does the tool find issues that developers care about?

This research question focuses on the practical value that the tool provides. We define an "issue" to be one of the two types of feedback that the tool provides: a return type annotation mismatch, or a crash. Recall that we define a crash to be an exception or other abnormal program termination. To evaluate this practical value, we conduct a manual analysis of issues that were discovered during the evaluation of RQ2 and submit crashes that are likely candidates to be bugs to repository maintainers, based on the criteria described in the following. If the repository maintainers conclude that the proposed bug report or pull request is worthwhile (that is, a repository maintainer fixes the issue, encourages us to fix the issue, or merges in a pull request that we have opened), then we declare that this issue is a true positive.

Our method for classifying issues is given as follows. We use the same crash deduplication and filtering methodology as given in Section 7.2.2.

Qualitative Issue Classification

After crashes have been deduplicated in filtered, as described in Section 7.2.2, we merge this set of crashes with the set of return type annotation mismatches to produce the full set of issues. We then categorize each issue according to a qualitative scheme, given in Table 7.1. Type annotation mismatches fall into the TE category, and all other categories describe attributes of exceptions. The ER category describes reproduction from the "top-level," which means that the crash can be reproduced by passing some input through the codebase's intended API. In the case of *mypy*, for example, this would mean passing a pathological Python file in through the command-line interface.

With this scheme, we analyze all issues found under any configuration and under any random seed in the 5 repositories cselected for RQ2 and RQ3. When manually analyzing a crash, we first attempt to understand the cause of the crash, and apply the categorizations given in Table 7.1. If the exception is not EX (unmet external dependencies), then we attempt to reproduce it by using the library through its intended API. If we can reproduce the exception, we apply the ER category and report it. In

Abbreviation	**Definition**
EX	exception is due to unmet dependencies on **ex**ternal environment (filesystem, configuration, etc)
FO	exception message clearly explains how to remedy the error (**f**ix **o**bvious)
EE	**e**xception is **e**xpected/not indicative of issue (either by calling code, indicated in docstring, noted as an "unrealistic" exception by developers, or there is no reasonable way to fix the issue except by raising another exception)
ER	**e**xception can be **r**eproduced from top-level
LB	exception is noted as a **l**egitimate **b**ug by the community
AE	the exception is caused by supplying the wrong type of value for an "Any" type annotation, indicating a type **a**nnotation **e**nhancement opportunity
TE	the function returned the wrong type of value according to its annotation

TABLE 7.1: Exception category abbreviation definitions

cases where we cannot reproduce from the top-level, we check the FUT's documentation and code that calls the FUT to see if the exception is expected and/or handled. If so, we apply the EE categorization. If the EE categorization is not applied, we report the bug and apply it if the repository maintainers indicate the issue is not legitimate. We do not report bugs to which we have applied EX or EE prior to reporting. FO is assigned based on a best-judgement interpretation of the exception message. LB is assigned if and only if the issue was accepted as legitimate by the open-source community. AE is applied if an exception is determined to be due to a value of an unexpected type passed into a function that has a parameter type annotation of "Any," and these exceptions are not reported. Data on the reported issues can be found in 7.12, and the full data sheet for all issues can be found in Appendix B.

7.2.4 Experimental Configurations

Here, we cover unfamiliar terminology and notation of experimental configurations, so that the presentation of the results in Section 7.3 make sense.

Notational Conventions

We generally shorten experimental configurations to strings of the form:

```
{repository_name}_{corpus_size}_{encoding_or_acl2s_reachout_freq}
```

`repository_name` is the name of the repository. `corpus_size` is the corpus size, and `acl2s_reachout_freq` is the frequency at which the fuzzing process reaches out to ACL2s for a new type example.

7.2.5 Repository Selection Methodology

We selected 15 open-source Python projects against which to evaluate our tool. Due to time constraints, we further pared this set down to 5 repositories for RQ2 and RQ3. Table 7.2 contains a listing of them, and the number of *function candidates* that we extracted from each. "#FCs" contains the number of total function candidates extracted. Note that this number may not be the number of functions we have data for in the results–some functions were not importable dynamically, even though they were resolved by mypy, and some functions fell into infinite loops, or otherwise had significant performance problems that prevented them from being fuzzed. In the results further in this chapter, we include the number of functions for which we have results in the relevant figures. Most repositories were picked from the list of "regular" type annotation users according to Di Grazia and Pradel [28]. The paper characterizes a project as a regular type annotation user if the number of type annotations grows roughly at the same rate as the overall code size. Two other repositories, *django* and *flask*, are from the "sprinter" category, and not in the type annotation study, respectively. These repositories were chosen due to their perceived popularity in the ecosystem as standard web server frameworks. For *django*, a version that is older than the latest, v4.2, had to be used in fuzzing because the tool is written in Python 3.8. The Django project has ceased to support Python 3.8 in versions after 4.2. Therefore, there is a risk that bugs found in this repository are not current to the latest version. But, 4.2 has been marked as a long-term support (LTS) version by Django, so any bugs found would indeed still be relevant to many users of this version.[1]

TABLE 7.2: Selected Repositories

Name	**# Contributors**	**# FCs**	**Category**	**Special Notes**
mypy	664	196	Regular	None
mindsdb	756	92	Regular	None
django	2,502	64	Sprinter	Had to use `stable/4.2.x` to maintain compatibility with Python 3.8
black	438	50	Regular	None
manticore	99	45	Regular	In maintenance mode
pipx	135	39	Regular	None
httpx	215	37	Regular	None
rich	232	33	Regular	None
vibora	25	32	Regular	Inactive
fuzzowski	2	28	Regular	None
rlcard	35	13	Regular	None
flask	715	12	N/A	Not in Type Ann. Study
qutebrowser	355	12	Regular	None
hbmqtt	35	8	Regular	None
returns	50	4	Regular	None

[1] https://www.djangoproject.com/download/

7.2.6 Environment

All experiments were run on MIT Supercloud [106] using Python 3.8.18.

7.2.7 Statistical Test Corrections

In order to be responsible in statistical inference, and to control the probability of Type I Error, we have decided to apply the Holm-Bonferroni method, which is a sequentially rejective procedure that aims to limit family-wise error rate [60]. The use of such statistical corrections when performing multiple tests is suggested in Arcuri and Briand [3]. Except when otherwise stated, we consider the statistical tests performed in each RQ to be a "family," and correct all of the p-values simultaneously according to this procedure.

7.3 Results

7.3.1 RQ1

Here, we present the results obtained to answer the first research question: *How does the input encoding affect fuzzing performance?* We find strong evidence that our custom encoding does not impair fuzzing, and in some ways it enhances the fuzzing performance of the tool, when compared against *pickle*.

Post-Mutation Decode Rates

Put simply, the custom encoding decodes successfully post-mutation **every time**. *pickle* decodes successfully post-mutation an average of **8.3%** of the time.

Finding 1 Our custom encoding works as designed: any encoded bit stream can be successfully decoded after arbitrary transformation to the data section.

Unique and total points

Figures 7.1, 7.2, 7.3, and 7.4 show comparisons of the unique and total points for corpus sizes 1, 5, and 10 for the 15 repositories. Each row of graphs contains three plots which correspond to the 1, 5, and 10 corpus size comparisons. Each point on each graph represents a single function, and the x-axis represents the ratio of average total points obtained under *custom* to the average total points obtained under *pickle*. The y-axis shows the same for unique points. Values of one (10^0) indicate equal numbers of total/unique points. The dotted guidelines are drawn at the 10^0 lines to divide the plots into four quadrants. Points in Quadrant I (top right) represent functions where both total and unique point values are higher for *custom* than for *pickle*. Quadrant II (top left) represents functions where total points are fewer for *custom* than for *pickle* but unique points are higher. Quadrant III (bottom left) represents functions where *pickle* does better for both quantities. Quadrant IV (bottom right) represents functions where the total points for *custom* are greater than *pickle* but unique points are less. Note the logarithmic scales on the axes.

There is a fairly consistent pattern among these graphs: most of the points lie in Quadrant I, which implies that under *custom*, there is a both a greater amount of raw fuzzing activity, and more of that fuzzing activity is unique. Points also consistently lie on the border between Quadrants II and III, meaning that *custom* produces fewer total trials throughout the fuzzing campaign but the same number of unique trials are produced.

To understand whether using the custom encoding has a significant impact on how much of the sample space is explored, we have devised two statistical tests. First, we test the hypothesis that the ratio of unique points to total points will be significantly increased under the custom encoding relative to pickle. Table 7.3 presents results of first batch of hypothesis tests. For each corpus size, we construct sample populations corresponding to each of the two encodings out of the ratios of unique points to total points for each independent trial of each function that was fuzzed across the 15 repositories. Each hypothesis test is a one-tailed Mann-Whitney U-test of the hypothesis that the sample population corresponding to our custom encoding is stochastically greater than that of the pickle encoding. Statistically significant results (post-Holm-Bonferroni correction) have their p-values highlighted in green.

Secondly, we test the hypothesis that the number of total points is significantly impacted. Table 7.4 contains the results of these tests. Each test is a two-tailed Mann-Whitney U-test of stochastic difference. Any test for which we reject the null hypothesis after Holm-Bonferroni correction (indicating that the custom and pickle total point distributions are different) has its p-value highlighted in green.

The reason why the sample size (N), is less in the unique/total point ratio hypothesis tests is because we filter out samples where the total points would be zero, leading to a division by zero. It is unclear why total points would be zero–these must be situations where the fuzzer was unable to successfully call the function-under-test. Due to time and resource constraints, we leave understanding this better to future work.

Note that all null hypotheses have been rejected, even after Holm-Bonferroni adjustment [60]. The unique/total point ratio tests indicate that there is a clearly significant increase in unique fuzzing activity under our custom encoding, and interestingly, the effect is more pronounced as the corpus grows larger ($\hat{A} = 0.6005$ for $|C| = 1$, $\hat{A} = 0.6353$ for $|C| = 10$). Recall also that for this RQ, we are only using the "shape"

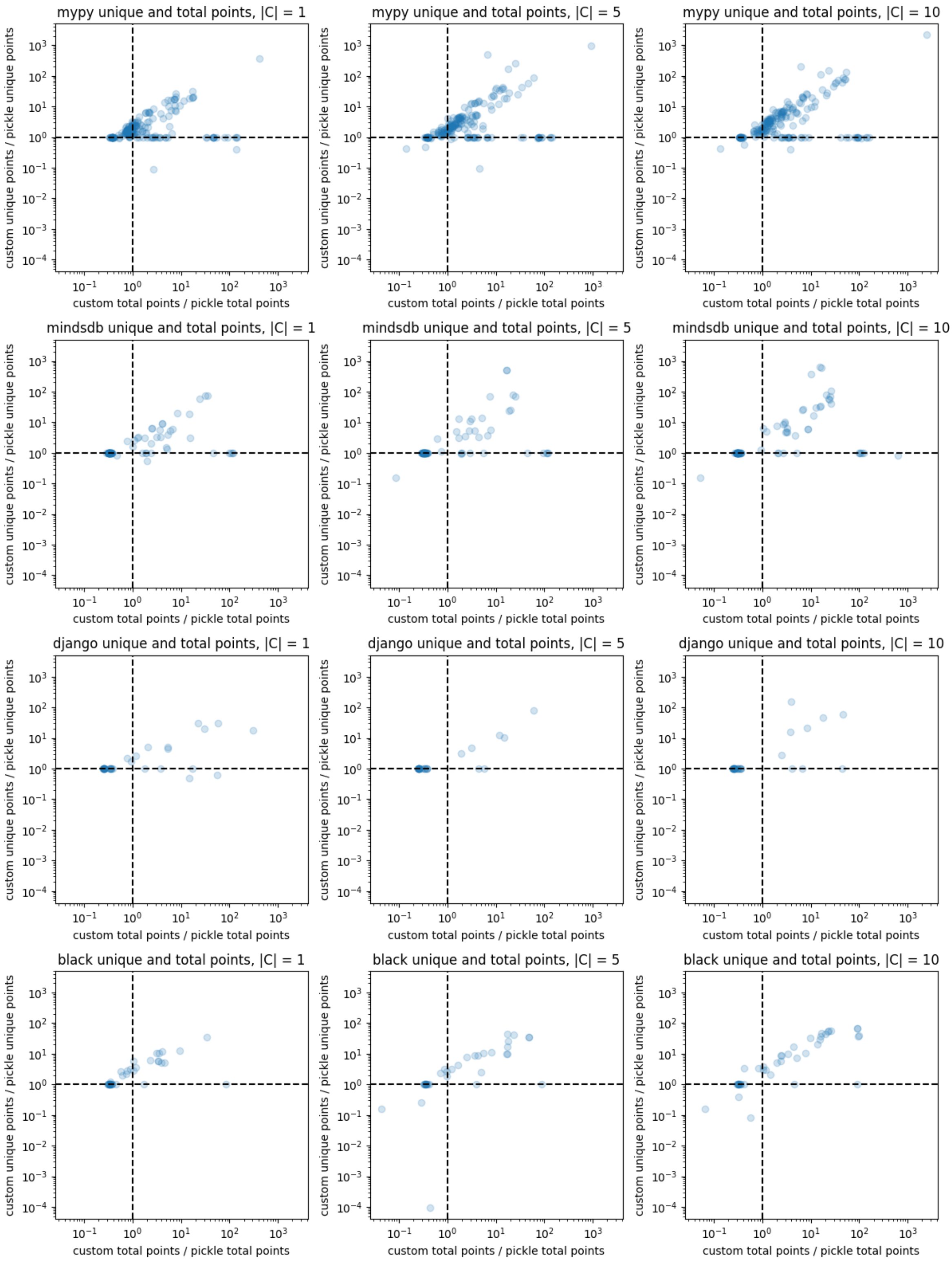

FIGURE 7.1: RQ1 Custom / Pickle unique and total point ratios, part 1

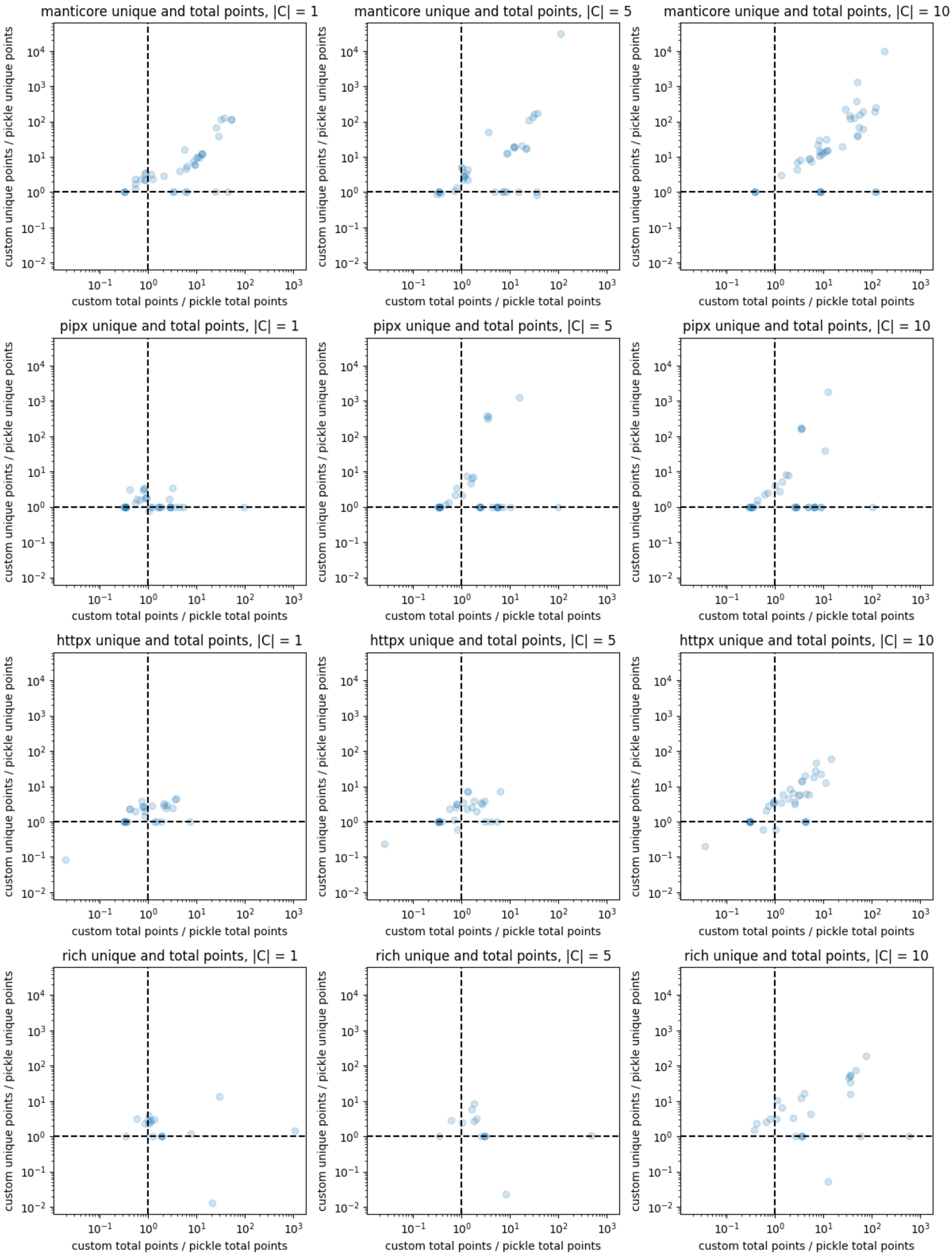

FIGURE 7.2: RQ1 Custom / Pickle unique and total point ratios, part 2

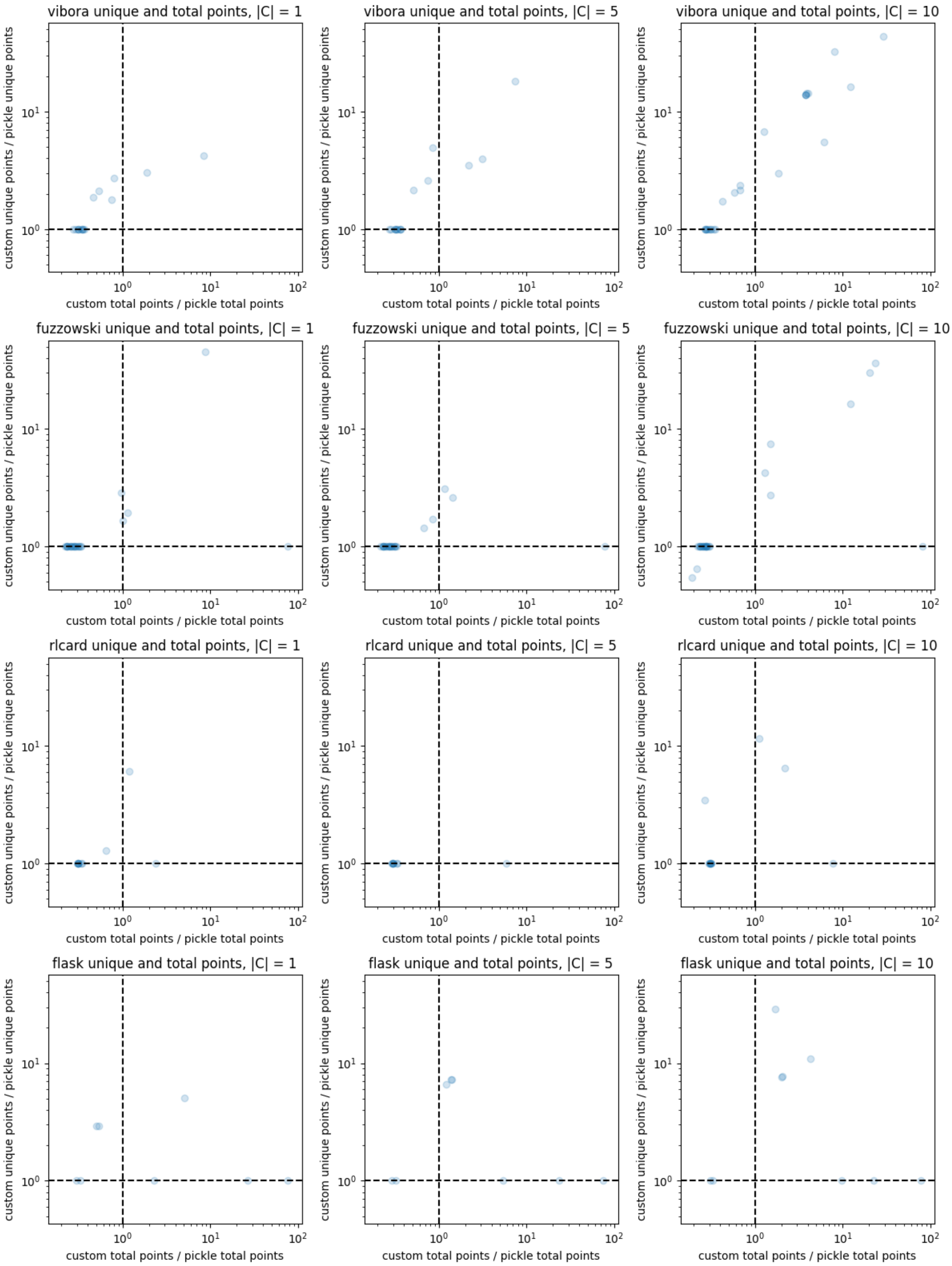

FIGURE 7.3: RQ1 Custom / Pickle unique and total point ratios, part 3

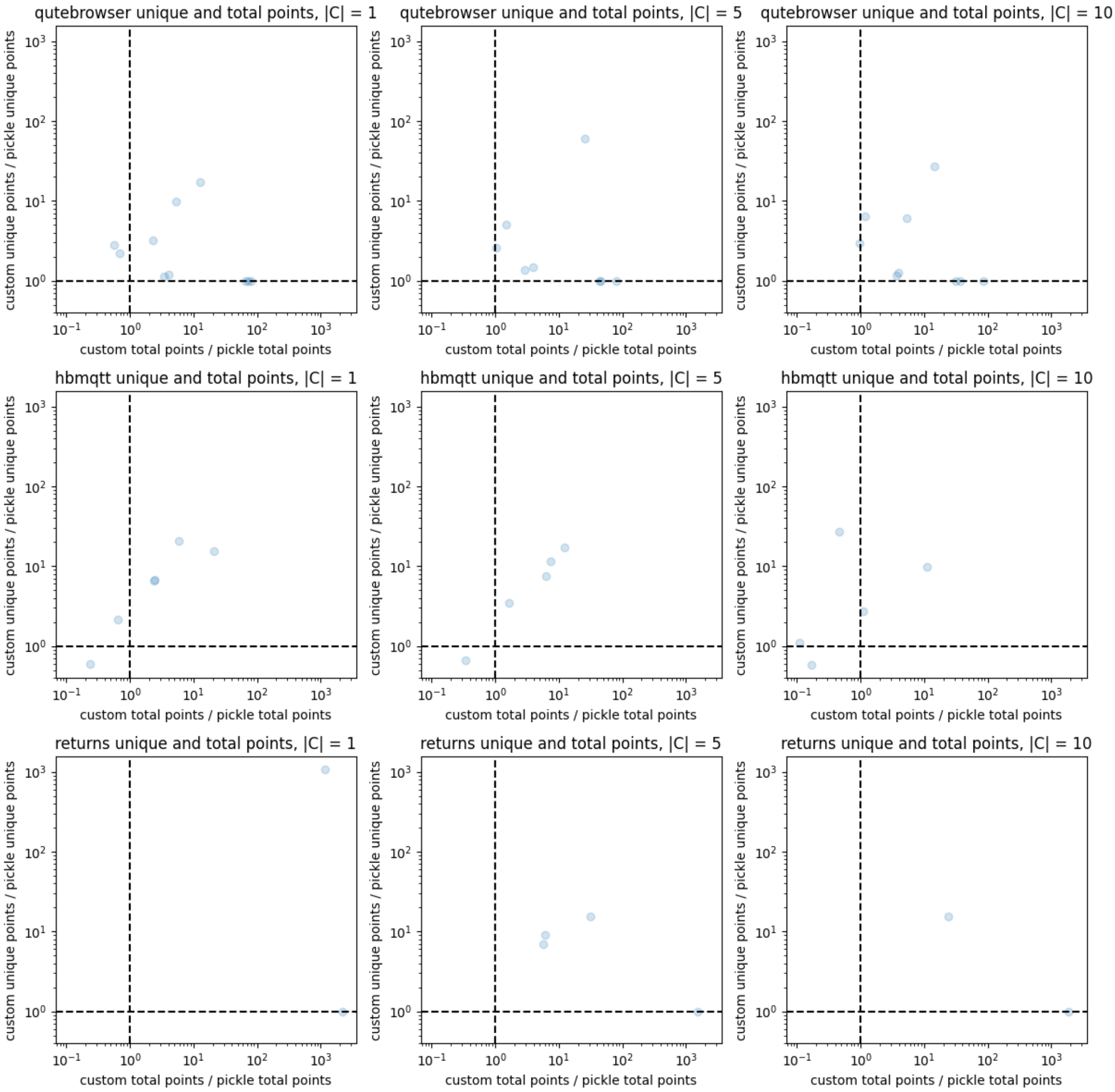

FIGURE 7.4: RQ1 Custom / Pickle unique and total point ratios, part 4

TABLE 7.3: Unique/Total Point Ratio Hypothesis Tests

Test	**C > P?**, $\|C\| = 1$	**C > P?**, $\|C\| = 5$	**C > P?**, $\|C\| = 10$
Sample Medians	P: 0.05998 C: 0.06667	P: 0.03315 C: 0.15043	P: 0.10758 C: 0.48045
Sample Means	P: 0.24805 C: 0.40934	P: 0.23379 C: 0.41661	P: 0.22261 C: 0.44295
Sample Size (N)	2,669	P: 2,540 C: 2,538	P: 2,830 C: 2,828
U-statistic	4277543.0	3974863.0	5084729.0
p-value	$2.472 * 10^{-37}$	$3.036 * 10^{-47}$	$7.204 * 10^{-70}$
$\hat{A}$ **Effect Size**	0.6005 $(0.5998, 0.6008)$	0.6166 $(0.6160, 0.6171)$	0.6353 $(0.6349, 0.6359)$

TABLE 7.4: Total Point Hypothesis Tests

Test	**C $\neq$ P?**, $\|C\| = 1$	**C $\neq$ P?**, $\|C\| = 5$	**C $\neq$ P?**, $\|C\| = 10$
Sample Medians	P: 7625.5 C: 18725.5	P: 5863.5 C: 18199.0	P: 7947.5 C: 25867.5
Sample Means	P: 62291.34 C: 229174.63	P: 53694.75 C: 316258.30	P: 58645.17 C: 362551.97
Sample Size (N)	2,700	2,570	2,860
U-statistic	3935159.5	3772902.0	4890565.0
p-value	$4.072 * 10^{-7}$	$9.233 * 10^{-19}$	$1.217 * 10^{-37}$
$\hat{A}$ **Effect Size**	0.5398 $(0.5392, 0.5402)$	0.5712 $(0.5705, 0.5716)$	0.5979 $(0.5972, 0.5982)$

information of the corpus elements, and scrambling the data. According to the interpretative guidance given in Vargha and Delaney [131], the effect size is "small" for $|C| = 1, 5$, but approaches a "medium" effect size for $|C| = 10$. The total point ratio null hypotheses were also rejected. The difference is also in favor of the custom encoding. In summary, not only is there more *unique* fuzzing activity under the custom encoding, but there is also more *overall* fuzzing activity under the custom encoding. These results strongly support the effectiveness of the custom encoding.

Finding 2 There is clear statistically significant evidence that there is both more *unique* fuzzing activity under the custom encoding, and more *overall* fuzzing activity under the custom encoding. These results strongly support the effectiveness of the encoding design in enabling the fuzzer to better explore the space of possibilities.

Coverage Growth

Figures 7.5, 7.6 and 7.7 show the cumulative coverage curves for the 15 repositories that were tested (note that it is not an error that coverage for the *returns* repository is a flat line–none of the 4 extracted functions in the repository were successfully fuzzed across all five independent trials and across all configurations). The solid lines represent mean statement coverage across all function bodies that were fuzzed throughout the 440 seconds. The dashed lines represent upper 95% confidence interval bounds, and the dotted lines represent lower 95% confidence interval bounds. A vertical tick on the solid line, if it is present, represents the knee that was detected on the curve, according to our definition in 13. There is a fair amount of variability between the repositories, but the general pattern is that the *custom* encoding curves tend to achieve higher coverage than the *pickle* curves. The knees also tend to occur earlier for our custom encoding, and the curves up to the knee for the custom encoding also tend to have higher slopes. Next, we compare time-to-knee, coverage-at-knee, and the ratio of the two to provide a more numerical comparison.

Finding 3 As expected, and as pointed out also by Klees et al. [70], there is variability in the amount of coverage obtained by each approach, and there is variability across repositories as to which approach does best. The general pattern that can be observed in the graphs is that the custom encoding approaches *usually* outperform the pickle approaches. Which size of corpus maximizes coverage across each repo is variable as well.

TTK, CAK, and CAK/TTK

To judge whether our encoding is better at facilitating effective fuzzing, we want to identify whether 1) our encoding reaches a "knee" quickly, indicating that it can allow the fuzzing algorithm to quickly explore the space, and 2) the coverage when that knee is reached is high, indicating the fullness with which the fuzzing algorithm is able to explore before hitting a wall. Figure 7.8 shows mean TTK for all independent fuzzing trials (416 function candidates in 15 repos * 5 random seeds = 2,080). The orange line shows mean TTK, in seconds for our custom encoding, while pickle is shown in the blue line. The gray whiskers represent 95% confidence intervals. In this graph, lower is better, since we want to measure how quickly the state space is explored. The custom encoding performs about as well as *pickle* when corpus size is 1 or 5, but *pickle* interestingly exhibits an increasing time-to-knee as the size of the corpus increases. This can be explained by the fact that random mutation of pickle files produces files that exhaust memory on decoding, slowing down the fuzzing process, and this occurs more for larger corpora because random mutation is more likely to generate pickle objects that decode with these memory usage problems. We have observed and collected a number of such occurrences throughout the process of running these experiments.

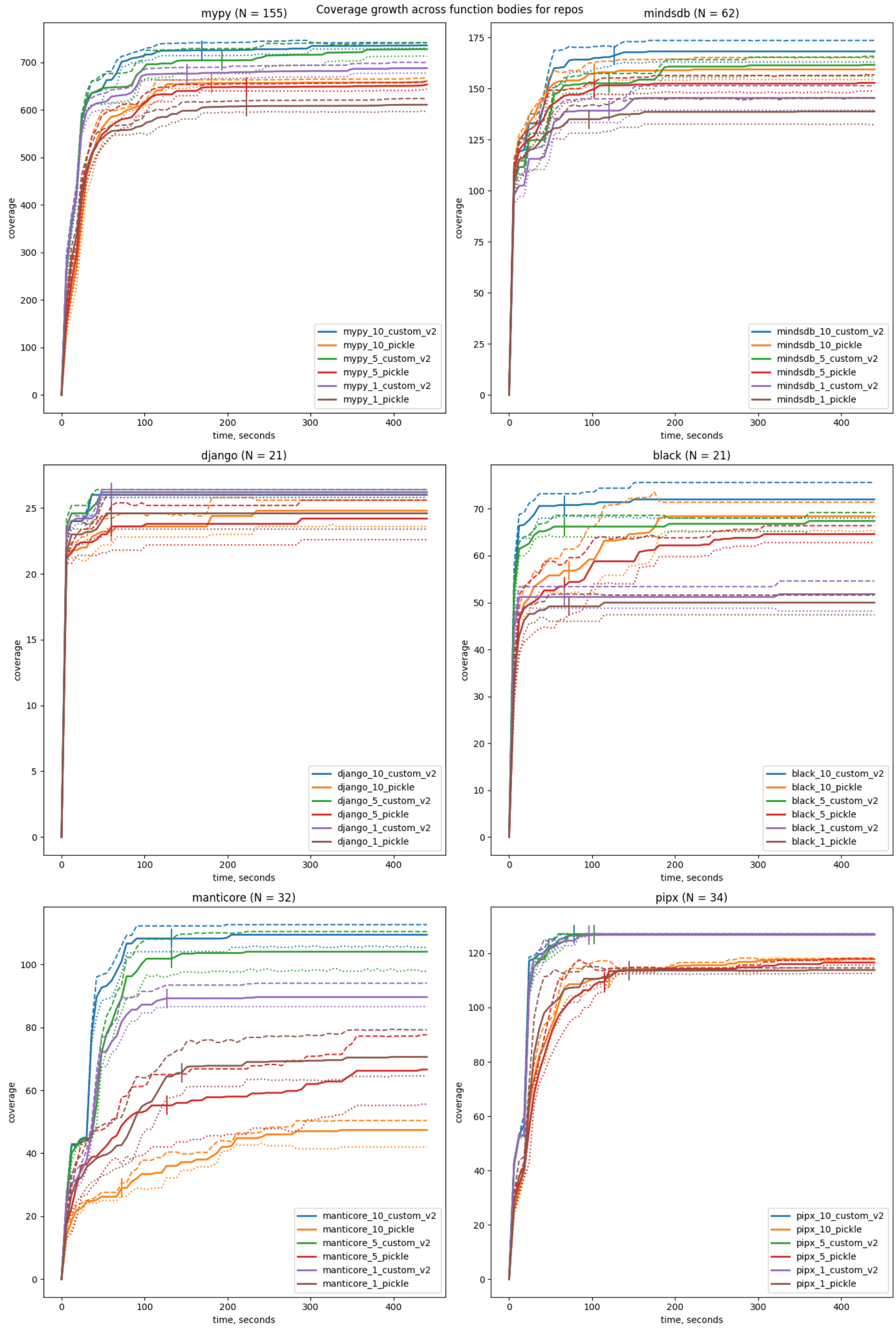

FIGURE 7.5: RQ1 Cumulative Coverage, Part 1

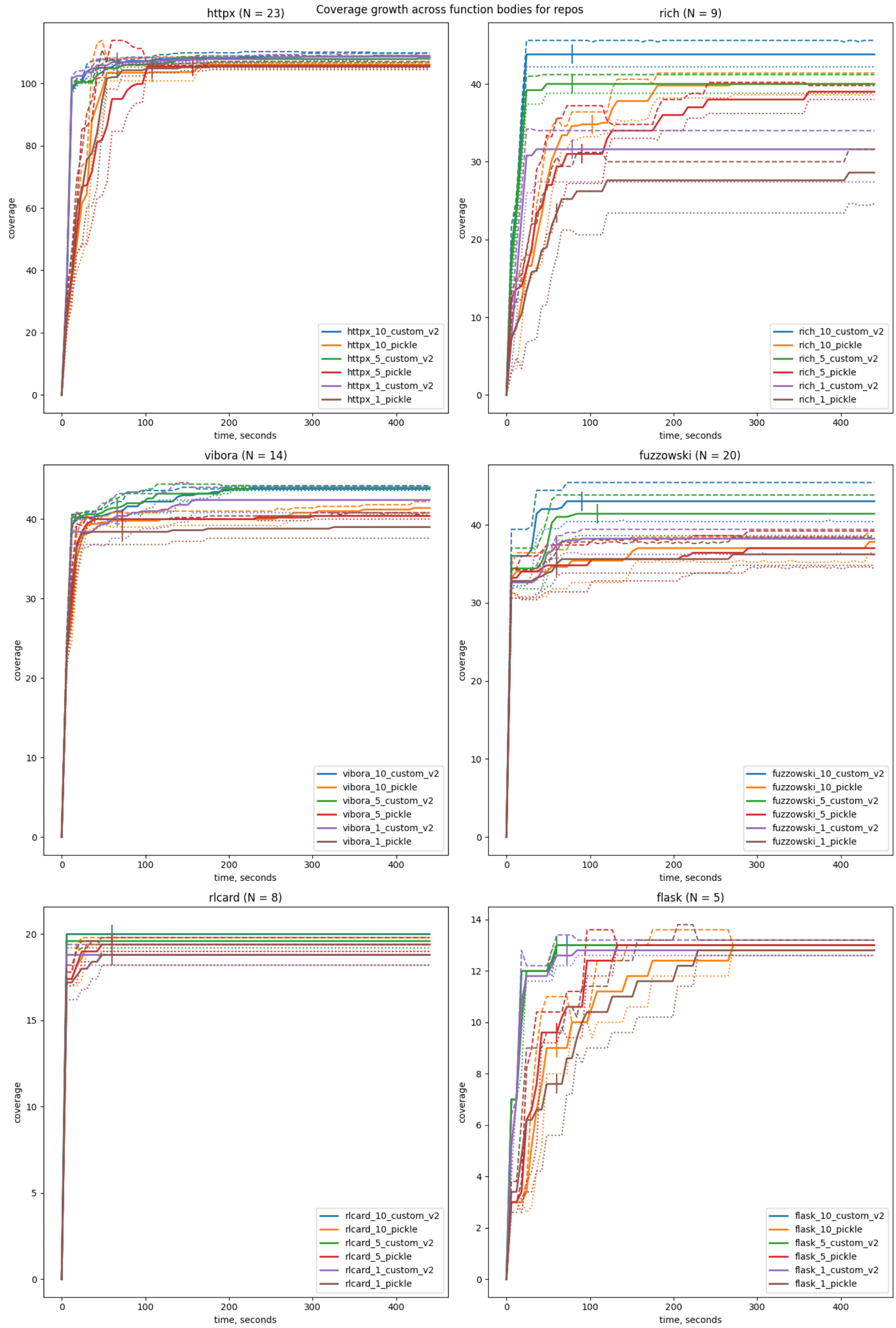

FIGURE 7.6: RQ1 Cumulative Coverage, Part 2

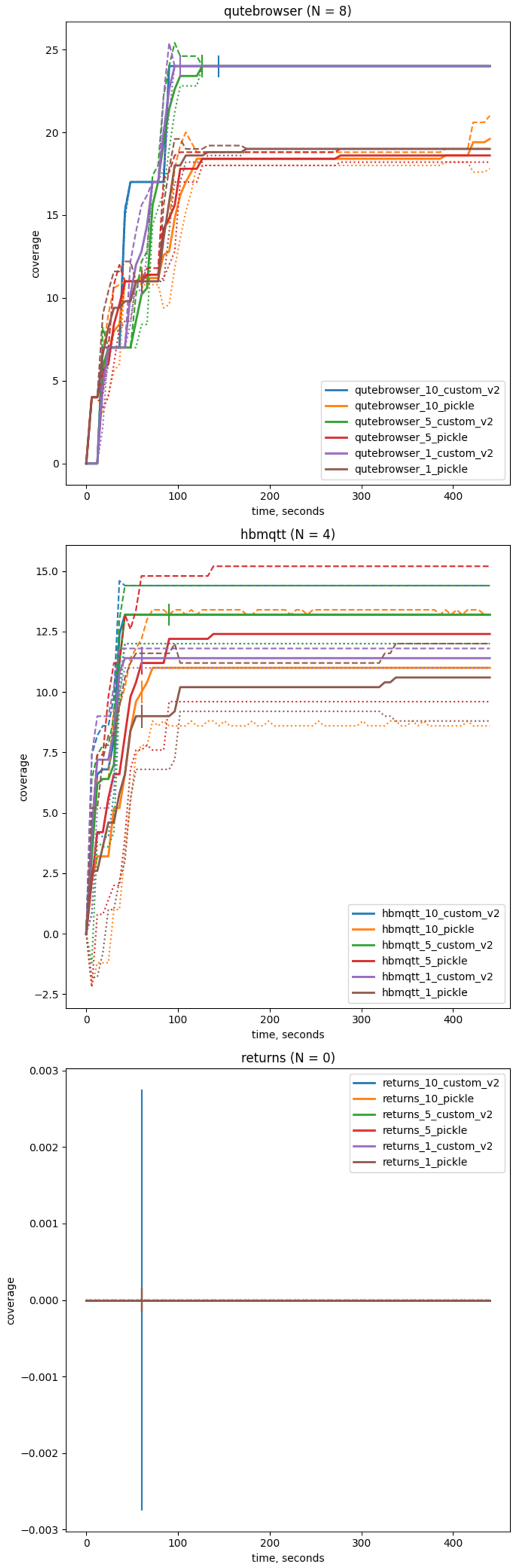

FIGURE 7.7: RQ1 Cumulative Coverage, Part 3

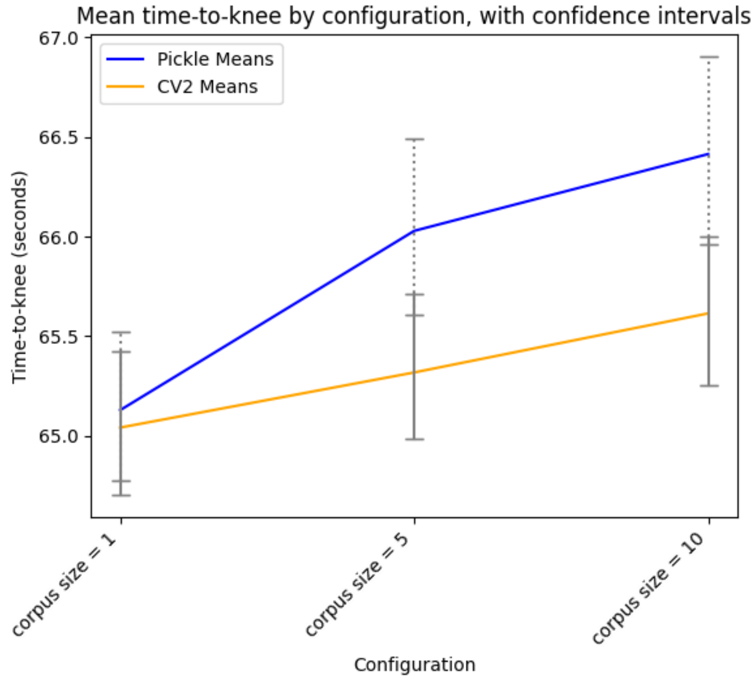

FIGURE 7.8: Mean times-to-knee across all successfully fuzzed functions in the 15 repos

Figure 7.9 shows the mean coverage at the knee. This value is calculated simply be taking the statement coverage of the function-under-test for each trial at the time of knee, calculated according to the definitions given in Section 7.2.1. Here, higher is better, since the unit is statements covered, and we have non-overlapping confidence intervals across all three corpus size configurations. When the knee is hit, one can expect our custom encoding to have covered an additional 0.4 statements in the body of the function being tested, approximately. Note that coverage in this case does not include coverage of function bodies other than the one being tested. This is a limitation of our analysis currently that we hope to improve upon in future work.

Putting this all together, 7.10 shows the CAK/TTK ratios for the six configurations. This figure summarizes the behavior we are optimizing for, hitting the knee quickly with greater coverage, into a single metric. Again, 95% confidence intervals

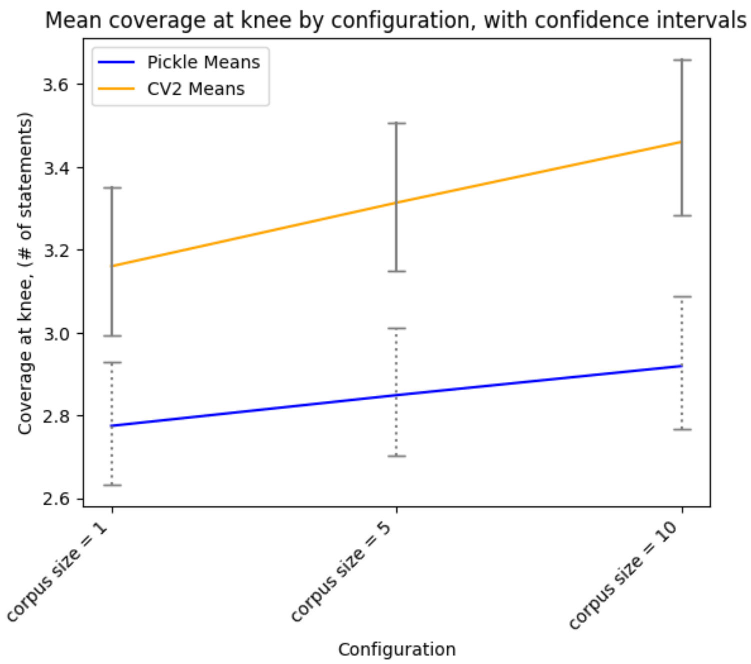

FIGURE 7.9: Mean Coverage-at-knee across all successfully fuzzed functions in the 15 repos

are given in the gray whiskers above and below the line. Notice that all custom encoding configurations have confidence intervals that are non-overlapping with any *pickle* configurations. From these results, we conclude that our custom encoding is achieving its intended goal to facilitate effective fuzzing. Given this and the other security considerations that motivate us to avoid using pickle, we evaluate the following RQs using the custom encoding exclusively, and encourage all potential users of this tool to exclusively use the custom encoding as well.

Finally, we present the results of statistical tests of CAK/TTK ratio between pickle and the custom encoding. Table 7.5 shows the results. Tests for which the null hypothesis was rejected, after Holm-Bonferroni correction [60] have their p-values highlighted in green. Note that all null hypotheses have been rejected. The $\hat{A}$ effect sizes fall into the "small" effect size range, as specified by Vargha and Delaney [131]. Although the effects are small, it is encouraging that these results do not contradict the

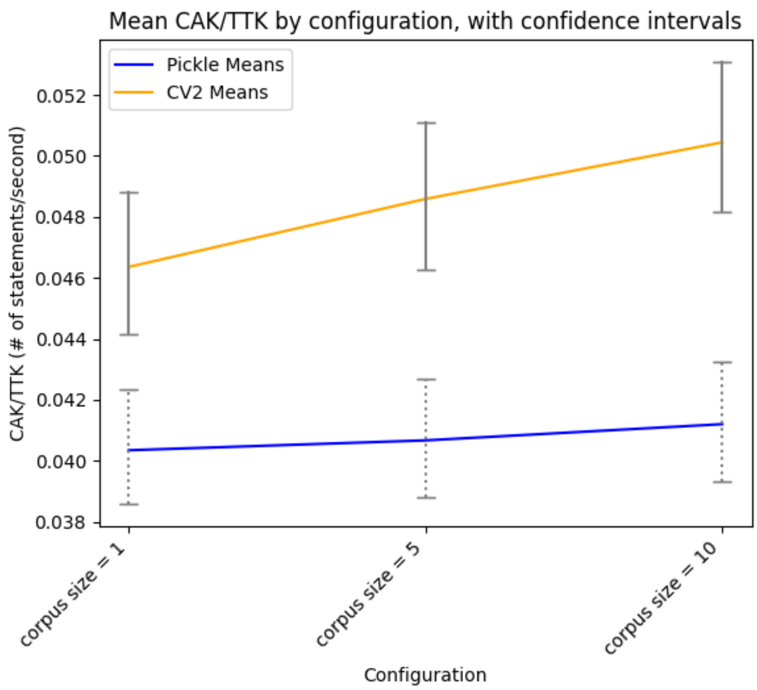

FIGURE 7.10: Mean CAK/TTK ratios across all successfully fuzzed functions in the 15 repos

earlier findings. Overall, there is strong evidence that the custom encoding, when compared to pickle, has a pronounced impact on the effectiveness of fuzzing, both in terms of the amount of input space exploration, and in the amount of coverage obtained in within short periods of time.

Finding 4 CAK/TTK analysis demonstrates that the custom encoding achieves more coverage faster on a statistically significant basis. We conclude that this, along with the other findings for this RQ, and in light of *pickle*'s security concerns, justifies the exclusive use of the custom encoding in our tool going forward.

TABLE 7.5: RQ1 CAK/TTK Ratio Hypothesis Tests

Test	**C > P?,** $\|C\| = 1$	**C > P?,** $\|C\| = 5$	**C > P?,** $\|C\| = 10$
Sample Medians	P: 0.03135 C: 0.03273	P: 0.03138 C: 0.03328	P: 0.03138 C: 0.03327
Sample Means	P: 0.04034 C: 0.04636	P: 0.04067 C: 0.04859	P: 0.04120 C: 0.05044
Sample Size (N**)**	2,080	2,080	2,080
U-statistic	2389827.0	2360841.5	2386624.0
p-value	$2.385 * 10^{-9}$	$1.633 * 10^{-7}$	$3.891 * 10^{-9}$
$\hat{A}$ **Effect Size**	0.5524 $(0.5519, 0.5530)$	0.5457 $(0.5450, 0.5461)$	0.5516 $(0.5511, 0.5521)$

7.3.2 RQ2

Here, we answer the second research question, "How does the configuration of the tool affect fuzzing performance?" We perform an evaluation across a variety of tool configurations, where we tune two parameters: `acl2s_reachout_freq`, which controls the weighting of selection between coverage-guided mutation input and fresh examples from ACL2s, and corpus size ($|C|$). Recall that we notate experimental configurations with values for these parameters separated by underscores. For example, `100_0.2` represents the configuration in which corpus size, $|C|$, equals 100, and we reach out to ACL2s for fresh examples 20% of the time, using coverage guided mutation the other 80%. The ACL2s-only configuration is notated as `acl2s`.

To understand the effect of tool configuration on the two core fuzzing metrics, coverage and crashes triggered, we present cumulative code coverage throughout the fuzzing campaign, as in the last section, as well as Coverage At Knee/Time-to-Knee ratios to show how quickly different configurations can explore the search space. We compute statistical tests of difference in CAK/TTK pairwise by approach. Following Klees et al. [70], we also present cumulative crashes found. Finally, to quantify how different approaches perform in terms of bug discovery, we perform *survival analysis* [3]. Kaplan-Meier survival curves for each configuration are given, and we compute pairwise statistical tests of difference in Restricted Mean Survival Time (RMST) [64].

We perform a mainline evaluation for RQ2 with the following configurations: `1_0`, `100_0`, `10000_0`, `1_0.1`, `100_0.1`, `10000_0.1`, `1_0.2`, `100_0.2`, `10000_0.2`, and `acl2s`. Due to time and resource constraints, we evaluate only on *mypy*, *mindsdb*, *django*, *black*, and *manticore*. We selected these five because they are the repositories with the most extracted function candidates. For reasons that will be elucidated shortly, we also perform an auxiliary evaluation on just *mypy* with the following additional configurations: `100_0.5`, `100_0.8`, and `100_1.0`. `mypy` is chosen because this is the repository with the greatest number of extracted function candidates. Again due to time and resource constraints, we could not extend this auxiliary evaluation to the other four repositories and make it part of the mainline. This will be improved in future work.

Furthermore, we encourage interpretation of the results contained in this section with care. The experimental design we have executed in this work compares configurations based on timeouts; timeouts are useful to evaluate on because, from the perspective of an end user, time is what matters most. However, this iteration of the TYPE HINT FUZZING implementation has not been extensively optimized, and differences in speed between independent trials and tool configurations may paint a misleading picture of the respective abilities of each configuration. Especially because we use a relatively short timeout per function candidate (440 seconds), inefficiencies in the use of time by various parts of the algorithm, such as corpus generation, filtering, and other startup costs can greatly affect performance metrics. We discuss these and other thoughts for improvement of this evaluation further in Chapter 8.

Cumulative Coverage

Figures 7.11 and 7.12 show cumulative coverage growth over 440 seconds of fuzzing for the five repositories. For ease of viewing, the evaluated configurations are split across two figures for each repository. There are no obvious consistent trends, but we make four observations:

1. the `acl2s` experiments (purple lines on the right column of plots) seem to consistently achieve the greatest or near-greatest coverage and they achieve it quickly

2. having a higher `acl2s_reachout_freq` seems to negatively affect the rate of new coverage gained up to the knee (compare `*_100_0` in the left column of figures to `*_100_0.2` on the right, for instance), but coverage is nearly equal by the end of the fuzzing campaign

3. very large corpora (i.e. $|C| = 10,000$) have a drastic negative effect on code coverage for some repositories

4. the first "knee" the is found is often not the end of coverage gains, and in some repositories, such as *mypy*, coverage appears to be increasing at the end of the fuzzing campaign, suggesting again that the timeout of 440 seconds is too short

As `acl2s_reachout_freq` approaches 1, fuzzing should become increasingly indistinguishable from pure ACL2s-driven fuzzing (`acl2s` configuration), except potentially in execution speed. Since `acl2s` does very well in code coverage, perhaps the reachout frequency choices of 0, 0.1, and 0.2 are too low. We initially expected that even 0.2 would be rather high, and would hinder the fuzzer's ability to mutate interesting inputs. The third observation corroborates Herrera et al. [58] in that smaller corpora lead to more efficient fuzzing, although perhaps for a different reason: in our experimentation, we observed computational slowdown due to resource strain when manipulating large corpora, which consequently hindered fuzzing performance. We also do not perform any corpus minimization, which means that a 10,000 element corpus may contain many redundant inputs. In this evaluation, based on coverage obtained by each configuration, it seems that 100 corpus elements is a good balance between too few and too many.

In the next subsection, we transition to a more quantitative analysis by reviewing CAK/TTK ratios and performing hypothesis tests of statistical difference.

Finding 5 Pure ACL2s-driven fuzzing (`acl2s`) performs best in code coverage for the configurations studied. However, the configurations studied represent a narrow band of the possibilities, and the story may change in a more comprehensive evaluation. Also, very large corpora place a burden on the fuzzing process, hindering code coverage (in part due to increased computational requirements). 100 corpus elements seems to be a reasonable balance between too few and too many, as configurations with $|C| = 100$ also achieve coverage comparable to `acl2s`.

CAK/TTK Ratios

Figure 7.13 shows the mean CAK/TTK ratios, with bootstrapped 95% confidence intervals, for each configuration. The means are taken over all independent trials of function candidates across the five repositories we studied for this RQ. The trend of the `acl2s` configuration outpacing all of the other fuzzing configurations becomes even clearer. Interestingly, it also seems that, in the set of configurations studied, greater $|C|$ and greater `acl2s_reachout_freq` both have negative effects on CAK/TTK ratio.

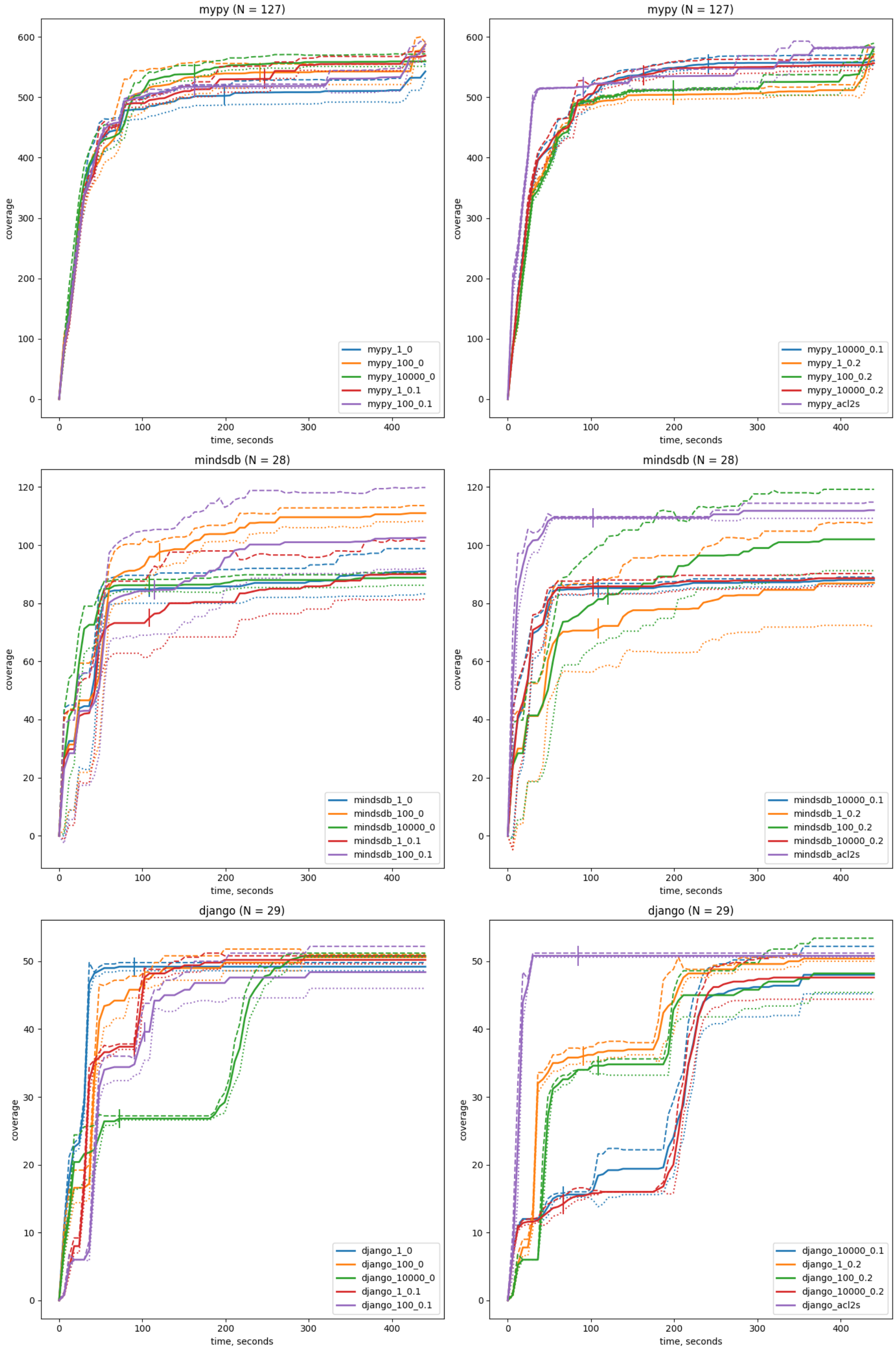

FIGURE 7.11: RQ2 Cumulative Coverage: mypy, mindsdb, django

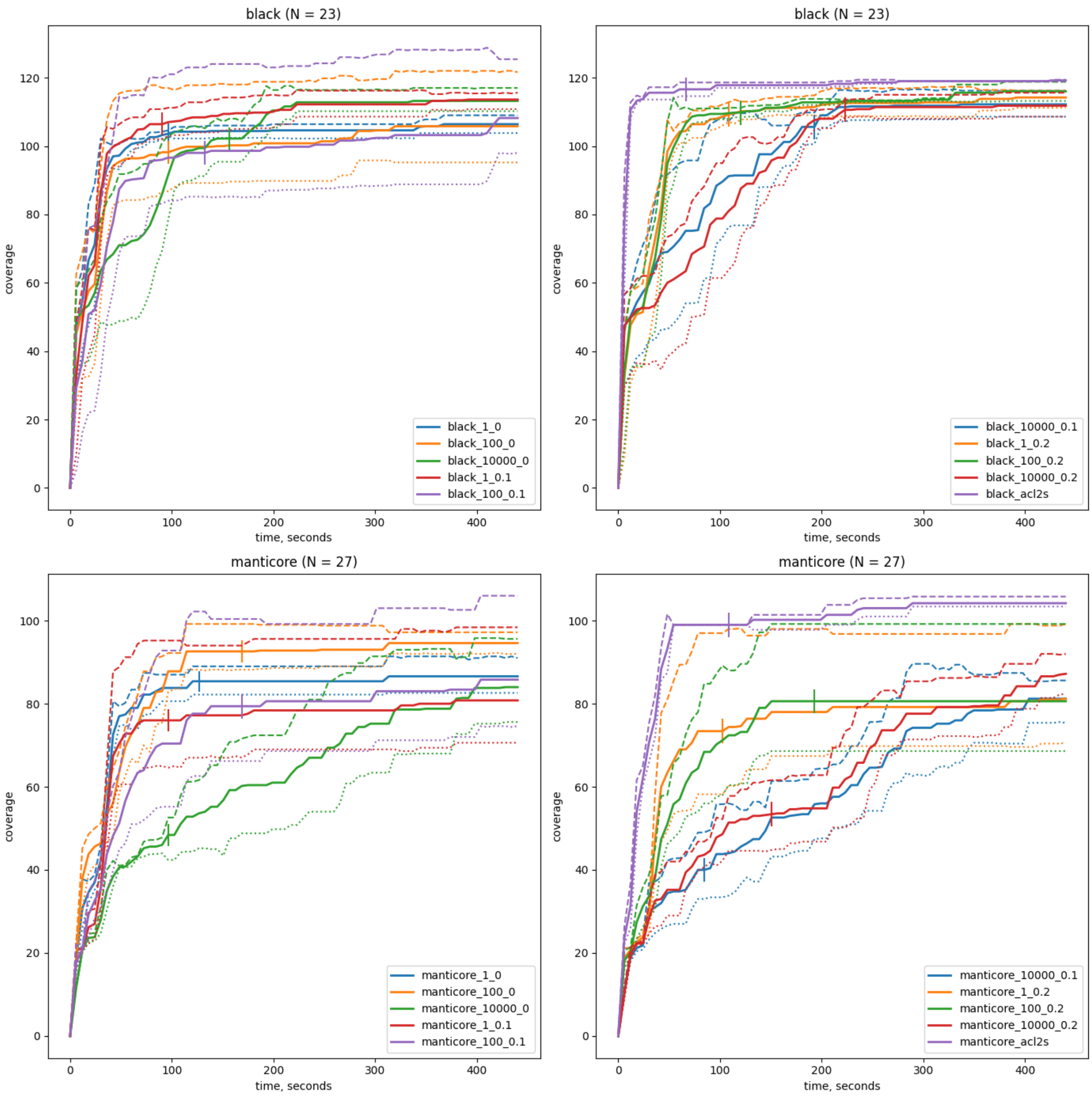

FIGURE 7.12: RQ2 Cumulative Coverage: black, manticore

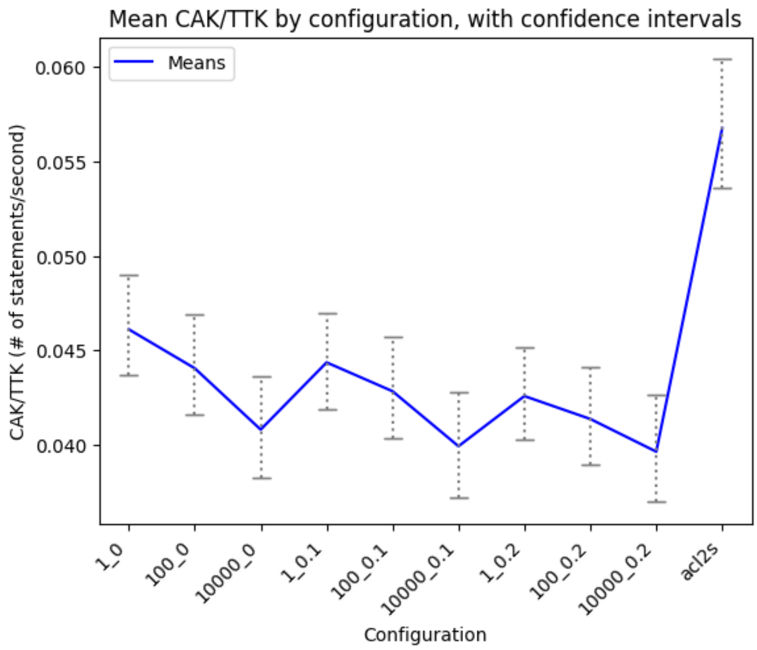

FIGURE 7.13: RQ2 CAK/TTK ratios

Auxiliary Evaluation

It initially seemed reasonable to expect that even small amounts of ACL2s input would be able to greatly assist the fuzzer in achieving more coverage faster, but, under the conditions of this evaluation, the previous results in this section have indicated that this is false. Since the range of `acl2s_reachout_freq` values we included in the initial evaluation only goes up to 0.2, it remains plausible that higher values should start to approach, or even exceed, `acl2s` in performance. Intuitively, as `acl2s_reachout_freq` approaches 1, the Atheris-ACL2s hybrid should begin to behave more like ACL2s alone. To test this hypothesis, we perform a small auxiliary evaluation on *mypy* with additional configurations `mypy_100_0.5`, `mypy_100_0.8`, and `mypy_100_1.0`. We choose *mypy* because it is the repository with by far the largest number of function candidates in our selection.

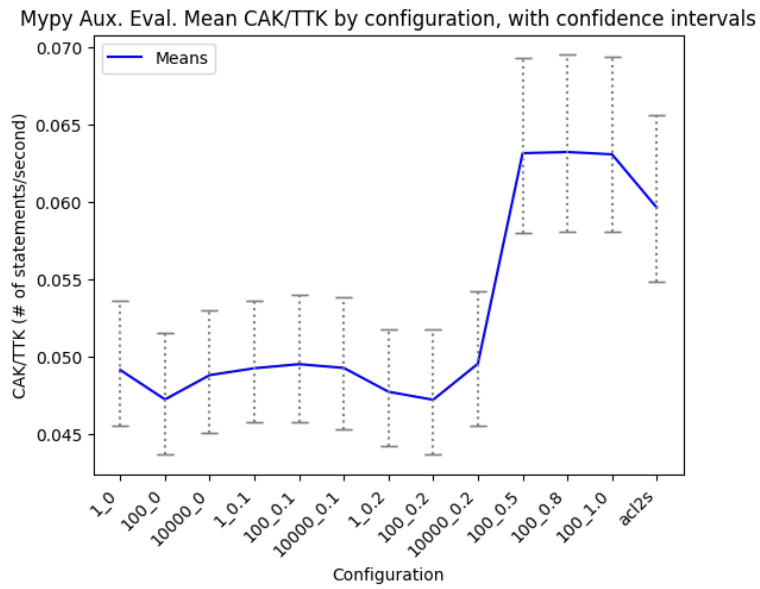

FIGURE 7.14: RQ2 Auxiliary Eval. CAK/TTK ratios

Figure 7.14 contains mean CAK/TTK ratios with bootstrapped 95% confidence intervals for the auxiliary evaluation, including the three new configurations. Here, we see that our hypothesis is confirmed: as `acl2s_reachout_freq` approaches 1, the performance becomes comparable to ACL2s alone. In this case, the CAK/TTK for `mypy_100_0.5`, `mypy_100_0.8`, and `mypy_100_1.0` exceed that of `acl2s`. We see that this trend is confirmed in Figure 7.15, which shows cumulative coverage across all configurations for *mypy*. Unfortunately, due to time and resource constraints, we could not complete this evaluation across the other repositories that are studied in RQ2, but there is no reason that we see to believe that the trend would not generalize. For the remainder of the evaluation of RQ2, we will present figures from the auxiliary evaluation alongside figures from the original RQ2 evaluation.

Following the guidance of Klees et al. [70] and Arcuri and Briand [3] to justify decision making with statistical rigor, we also perform hypothesis tests of statistical difference in CAK/TTK with the Mann-Whitney U-test, pairwise by configuration, with p-values adjusted according to Holm [60]. The null and alternative hypotheses of each test are given as follows:

- H_0: The distributions of the two samples are equal
- H_1: The distributions of the two samples are not equal

The p-value "family," for the purposes of Holm-Bonferroni correction [60], includes these results, and the results of the RMST comparison later in this section (we treat the mainline RQ2 evaluation and the auxiliary evaluation as two separate "families" that are corrected separately). The results for the RQ2 mainline evaluation are given in Table 7.8. The results for the auxiliary evaluation are given in Table 7.9. To aid in interpreting these, and following the guidance of Arcuri and Briand [3], the median CAK/TTK ratios for each configuration in the mainline and auxiliary evaluations are given in Tables 7.6 and 7.7, respectively. Distributions of CAK/TTK for the auxiliary evaluation are also given in 7.16. In these tables, the Vargha-Delaney $\hat{A}_{12}$ effect size is computed with sample 1 being the row and sample 2 being the column. Therefore, effect sizes above 0.5 indicate that the CAK/TTK of the configuration of the row of the cell is stochastically greater than that of the column, and vice versa for effect sizes below 0.5 [131]. Statistically significant results are highlighted in green. The first line of each cell gives the U-statistic of each test (U-statistics are omitted in Table 7.9 due to space constraints), the second line gives the point estimate of $\hat{A}_{12}$, the third line gives a bootstrapped confidence 95% interval for A, and the fourth line gives the p-value.

The results in Table 7.8 clearly indicate that `acl2s` achieves better CAK/TTK on a statistically significant basis than every configuration tested except `1_0`. The configurations with $|C| = 10,000$, conversely, show statistically significantly worse CAK/TTK than almost every other configuration tested. Another strong signal is displayed in the `100_0` column, where it achieves higher CAK/TTK on a statistically significant basis

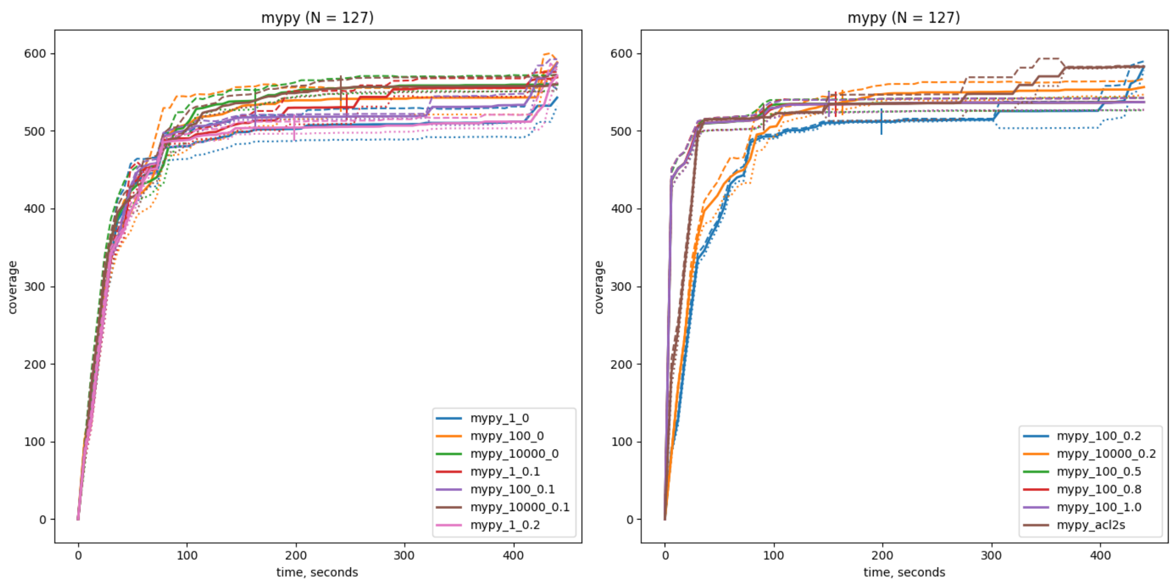

FIGURE 7.15: RQ2 Auxiliary Eval. Cumulative Coverage for mypy

than every other configuration except `1_0` and `1_0.1`, where the null hypotheses were not rejected, and `acl2s`, where `acl2s` performs better.

Experiment	Median CAK/TTK
1_0	0.032960
100_0	0.033074
10000_0	0.031201
1_0.1	0.033044
100_0.1	0.033064
10000_0.1	0.031394
1_0.2	0.033059
100_0.2	0.032802
10000_0.2	0.016551
acl2s	0.047296

TABLE 7.6: RQ2 Median CAK/TTK Ratios

The results from the auxiliary evaluation in Table 7.9 are less clear. Interestingly, it seems that the impact of the higher `acl2s_reachout_freq` configurations is not as pronounced as Figure 7.14 would indicate: none of the null hypotheses in the `100_0.5`, `100_0.8`, and `100_1.0` rows are rejected except for those in the `10000_0` column, and

Experiment	Median CAK/TTK
1_0	0.032947
100_0	0.033074
10000_0	0.031201
1_0.1	0.038910
100_0.1	0.036759
10000_0.1	0.032759
1_0.2	0.036757
100_0.2	0.033061
10000_0.2	0.032742
100_0.5	0.048637
100_0.8	0.048636
100_1.0	0.048636
acl2s	0.049318

TABLE 7.7: RQ2 Auxiliary Eval. Median CAK/TTK Ratios

when one of these configurations is compared to another. The `acl2s` results are surprising as well: `acl2s` still achieves statistically significant performance gains over `100_0.5`, `100_0.8`, and `100_1.0`, but not `100_0`, `100_0.1`, or `100_0.2`; we do not reject the null hypotheses in the latter cases. The strongest signal from the auxiliary evaluation that concurs with the mainline evaluation is that configurations with $|C| = 10,000$ perform significantly worse.

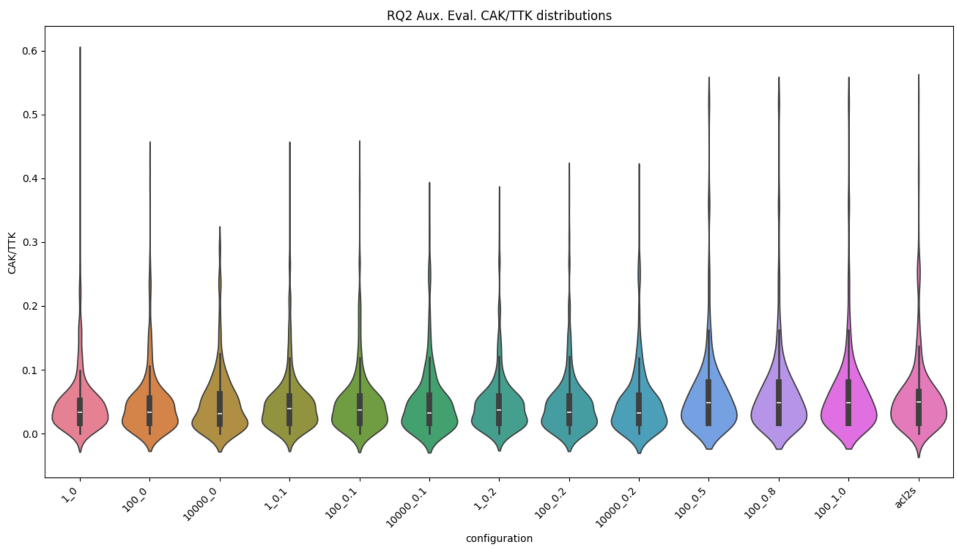

FIGURE 7.16: RQ2 Auxiliary Eval. CAK/TTK Distributions

It is important to note that Figures 7.13 and 7.14 are showing mean CAK/TTK, but the distributions are strongly non-normal, and they do not exhibit variance homogeneity, as shown in Figure 7.16.

Finding 6 When examining Coverage At Knee/Time-to-Knee, a measure of how much code can be covered per unit time, before a "knee" is encountered where progress slows, the strongest signal is that $|C| = 10,000$ is too large, and evidently hinders fuzzing progress. $|C| = 100$ again appears to be the best all-around choice, with mixed results on what value of `acl2s_reachout_freq` is best. More work is needed to fully understand the impact of this parameter. `acl2s`, on the other hand, consistently performs very well at achieving code coverage quickly.

We now turn our study to the ability of each configuration to find crashes in code-under-test.

	1_0	100_0	10000_0	1_0.1	100_0.1	10000_0.1	1_0.2	100_0.2	10000_0.2
100_0	$U = 697438.500$ $\hat{A} = 0.509$ $(0.509, 0.510)$ $p = 0.4263$								
10000_0	$U = 551740.000$ $\hat{A} = 0.403$ $(0.402, 0.404)$ $p = 4.0348 * 10^{-16}$	$U = 579532.000$ $\hat{A} = 0.423$ $(0.423, 0.424)$ $p = 1.1807 * 10^{-10}$							
1_0.1	$U = 680573.500$ $\hat{A} = 0.497$ $(0.496, 0.498)$ $p = 0.8123$	$U = 645311.500$ $\hat{A} = 0.471$ $(0.471, 0.472)$ $p = 0.0165$	$U = 792493.000$ $\hat{A} = 0.579$ $(0.578, 0.580)$ $p = 3.3627 * 10^{-11}$						
100_0.1	$U = 651310.500$ $\hat{A} = 0.476$ $(0.475, 0.476)$ $p = 0.0423$	$U = 616457.500$ $\hat{A} = 0.450$ $(0.449, 0.451)$ $p = 3.0759 * 10^{-05}$	$U = 763128.000$ $\hat{A} = 0.557$ $(0.557, 0.558)$ $p = 1.3475 * 10^{-06}$	$U = 679250.500$ $\hat{A} = 0.496$ $(0.496, 0.497)$ $p = 0.7500$					
10000_0.1	$U = 528409.500$ $\hat{A} = 0.386$ $(0.385, 0.387)$ $p = 1.0550 * 10^{-21}$	$U = 565014.500$ $\hat{A} = 0.413$ $(0.412, 0.414)$ $p = 2.1802 * 10^{-13}$	$U = 691554.000$ $\hat{A} = 0.505$ $(0.504, 0.506)$ $p = 0.6613$	$U = 557549.500$ $\hat{A} = 0.407$ $(0.406, 0.408)$ $p = 6.6735 * 10^{-15}$	$U = 605831.500$ $\hat{A} = 0.443$ $(0.442, 0.443)$ $p = 1.3523 * 10^{-06}$				
1_0.2	$U = 661078.000$ $\hat{A} = 0.483$ $(0.482, 0.484)$ $p = 0.1522$	$U = 627078.000$ $\hat{A} = 0.458$ $(0.458, 0.459)$ $p = 0.0004$	$U = 772857.000$ $\hat{A} = 0.565$ $(0.564, 0.565)$ $p = 5.7128 * 10^{-08}$	$U = 689220.000$ $\hat{A} = 0.503$ $(0.503, 0.504)$ $p = 0.7701$	$U = 671765.000$ $\hat{A} = 0.491$ $(0.490, 0.492)$ $p = 0.4369$	$U = 792972.000$ $\hat{A} = 0.579$ $(0.578, 0.580)$ $p = 2.6270 * 10^{-11}$			
100_0.2	$U = 633649.000$ $\hat{A} = 0.463$ $(0.462, 0.463)$ $p = 0.0019$	$U = 601905.000$ $\hat{A} = 0.440$ $(0.439, 0.440)$ $p = 4.1538 * 10^{-07}$	$U = 744951.000$ $\hat{A} = 0.544$ $(0.543, 0.545)$ $p = 0.0002$	$U = 670254.000$ $\hat{A} = 0.490$ $(0.489, 0.491)$ $p = 0.3842$	$U = 646526.000$ $\hat{A} = 0.472$ $(0.471, 0.473)$ $p = 0.0200$	$U = 765176.000$ $\hat{A} = 0.559$ $(0.558, 0.560)$ $p = 6.8306 * 10^{-07}$	$U = 688579.000$ $\hat{A} = 0.503$ $(0.502, 0.504)$ $p = 0.8001$		
10000_0.2	$U = 525649.500$ $\hat{A} = 0.384$ $(0.383, 0.385)$ $p = 2.0034 * 10^{-22}$	$U = 561906.500$ $\hat{A} = 0.410$ $(0.409, 0.411)$ $p = 5.1011 * 10^{-14}$	$U = 690997.000$ $\hat{A} = 0.505$ $(0.504, 0.505)$ $p = 0.6863$	$U = 554675.500$ $\hat{A} = 0.405$ $(0.405, 0.406)$ $p = 1.6131 * 10^{-15}$	$U = 583610.500$ $\hat{A} = 0.426$ $(0.426, 0.427)$ $p = 5.6964 * 10^{-10}$	$U = 657524.500$ $\hat{A} = 0.480$ $(0.479, 0.481)$ $p = 0.0963$	$U = 573003.000$ $\hat{A} = 0.419$ $(0.418, 0.419)$ $p = 7.5546 * 10^{-12}$	$U = 600943.000$ $\hat{A} = 0.439$ $(0.438, 0.440)$ $p = 2.7745 * 10^{-07}$	
acl2s	$U = 725950.000$ $\hat{A} = 0.530$ $(0.530, 0.531)$ $p = 0.0110$	$U = 749616.000$ $\hat{A} = 0.548$ $(0.547, 0.548)$ $p = 6.5710 * 10^{-05}$	$U = 881823.000$ $\hat{A} = 0.644$ $(0.643, 0.645)$ $p = 1.0783 * 10^{-33}$	$U = 774591.000$ $\hat{A} = 0.566$ $(0.565, 0.567)$ $p = 3.3735 * 10^{-08}$	$U = 801346.000$ $\hat{A} = 0.585$ $(0.585, 0.586)$ $p = 8.0492 * 10^{-13}$	$U = 902053.000$ $\hat{A} = 0.659$ $(0.658, 0.660)$ $p = 1.3158 * 10^{-40}$	$U = 790918.000$ $\hat{A} = 0.578$ $(0.577, 0.578)$ $p = 6.9753 * 10^{-11}$	$U = 813575.000$ $\hat{A} = 0.594$ $(0.594, 0.595)$ $p = 2.5652 * 10^{-15}$	$U = 905180.000$ $\hat{A} = 0.661$ $(0.660, 0.662)$ $p = 9.7128 * 10^{-42}$

TABLE 7.8: Hypothesis Test Results of Difference in CAK/TTK

	1_0	100_0	10000_0	1_0.1	100_0.1	10000_0.1	1_0.2	100_0.2	10000_0.2	100_0.5	100_0.8	100_1.0
100_0	$\hat{A} = 0.552$ (0.551, 0.553) $p = 0.0012$											
10000_0	$\hat{A} = 0.424$ (0.423, 0.425) $p = 2.4136 * 10^{-06}$	$\hat{A} = 0.450$ (0.449, 0.451) $p = 0.0020$										
1_0.1	$\hat{A} = 0.576$ (0.575, 0.577) $p = 2.3131 * 10^{-06}$	$\hat{A} = 0.452$ (0.451, 0.453) $p = 0.0029$	$\hat{A} = 0.571$ (0.571, 0.573) $p = 9.3390 * 10^{-06}$									
100_0.1	$\hat{A} = 0.571$ (0.570, 0.572) $p = 1.0453 * 10^{-05}$	$\hat{A} = 0.450$ (0.449, 0.451) $p = 0.0021$	$\hat{A} = 0.565$ (0.564, 0.566) $p = 5.7948 * 10^{-05}$	$\hat{A} = 0.566$ (0.565, 0.567) $p = 4.2160 * 10^{-05}$								
10000_0.1	$\hat{A} = 0.414$ (0.412, 0.415) $p = 8.5069 * 10^{-08}$	$\hat{A} = 0.439$ (0.438, 0.440) $p = 0.0002$	$\hat{A} = 0.553$ (0.551, 0.554) $p = 0.0011$	$\hat{A} = 0.417$ (0.416, 0.418) $p = 2.4951 * 10^{-07}$	$\hat{A} = 0.421$ (0.420, 0.422) $p = 1.1007 * 10^{-06}$							
1_0.2	$\hat{A} = 0.568$ (0.567, 0.569) $p = 2.7839 * 10^{-05}$	$\hat{A} = 0.445$ (0.444, 0.446) $p = 0.0006$	$\hat{A} = 0.562$ (0.561, 0.563) $p = 0.0001$	$\hat{A} = 0.563$ (0.562, 0.564) $p = 9.9606 * 10^{-05}$	$\hat{A} = 0.427$ (0.426, 0.427) $p = 5.2129 * 10^{-06}$	$\hat{A} = 0.575$ (0.574, 0.576) $p = 3.0354 * 10^{-06}$						
100_0.2	$\hat{A} = 0.554$ (0.553, 0.555) $p = 0.0009$	$\hat{A} = 0.437$ (0.436, 0.438) $p = 9.1112 * 10^{-05}$	$\hat{A} = 0.550$ (0.549, 0.551) $p = 0.0021$	$\hat{A} = 0.549$ (0.548, 0.550) $p = 0.0022$	$\hat{A} = 0.419$ (0.419, 0.421) $p = 5.7448 * 10^{-07}$	$\hat{A} = 0.562$ (0.561, 0.563) $p = 0.0001$	$\hat{A} = 0.557$ (0.556, 0.558) $p = 0.0004$					
10000_0.2	$\hat{A} = 0.413$ (0.412, 0.415) $p = 8.0903 * 10^{-08}$	$\hat{A} = 0.439$ (0.438, 0.440) $p = 0.0002$	$\hat{A} = 0.554$ (0.553, 0.555) $p = 0.0007$	$\hat{A} = 0.417$ (0.416, 0.418) $p = 2.4529 * 10^{-07}$	$\hat{A} = 0.421$ (0.420, 0.422) $p = 1.1123 * 10^{-06}$	$\hat{A} = 0.437$ (0.436, 0.438) $p = 0.0001$	$\hat{A} = 0.425$ (0.423, 0.425) $p = 3.0116 * 10^{-06}$	$\hat{A} = 0.438$ (0.437, 0.439) $p = 0.0001$				
100_0.5	$\hat{A} = 0.487$ (0.485, 0.488) $p = 0.4157$	$\hat{A} = 0.512$ (0.511, 0.513) $p = 0.4585$	$\hat{A} = 0.643$ (0.642, 0.644) $p = 7.6654 * 10^{-19}$	$\hat{A} = 0.490$ (0.489, 0.491) $p = 0.5237$	$\hat{A} = 0.496$ (0.494, 0.497) $p = 0.7833$	$\hat{A} = 0.508$ (0.507, 0.509) $p = 0.6104$	$\hat{A} = 0.499$ (0.498, 0.500) $p = 0.9393$	$\hat{A} = 0.511$ (0.510, 0.512) $p = 0.4844$	$\hat{A} = 0.507$ (0.506, 0.508) $p = 0.6867$			
100_0.8	$\hat{A} = 0.487$ (0.486, 0.488) $p = 0.4220$	$\hat{A} = 0.512$ (0.511, 0.513) $p = 0.4496$	$\hat{A} = 0.643$ (0.642, 0.644) $p = 6.4258 * 10^{-19}$	$\hat{A} = 0.490$ (0.489, 0.491) $p = 0.5329$	$\hat{A} = 0.496$ (0.495, 0.497) $p = 0.7954$	$\hat{A} = 0.508$ (0.508, 0.510) $p = 0.5997$	$\hat{A} = 0.499$ (0.498, 0.500) $p = 0.9508$	$\hat{A} = 0.512$ (0.511, 0.513) $p = 0.4748$	$\hat{A} = 0.507$ (0.506, 0.508) $p = 0.6752$	$\hat{A} = 0.412$ (0.411, 0.413) $p = 4.0889 * 10^{-08}$		
100_1.0	$\hat{A} = 0.487$ (0.486, 0.488) $p = 0.4156$	$\hat{A} = 0.512$ (0.511, 0.513) $p = 0.4585$	$\hat{A} = 0.643$ (0.642, 0.644) $p = 7.6232 * 10^{-19}$	$\hat{A} = 0.490$ (0.489, 0.491) $p = 0.5234$	$\hat{A} = 0.496$ (0.494, 0.497) $p = 0.7831$	$\hat{A} = 0.508$ (0.507, 0.509) $p = 0.6108$	$\hat{A} = 0.499$ (0.498, 0.500) $p = 0.9389$	$\hat{A} = 0.511$ (0.510, 0.512) $p = 0.4848$	$\hat{A} = 0.506$ (0.506, 0.508) $p = 0.6871$	$\hat{A} = 0.411$ (0.410, 0.412) $p = 3.4938 * 10^{-08}$	$\hat{A} = 0.588$ (0.587, 0.589) $p = 4.0996 * 10^{-08}$	
acl2s	$\hat{A} = 0.479$ (0.478, 0.480) $p = 0.1921$	$\hat{A} = 0.502$ (0.500, 0.502) $p = 0.9246$	$\hat{A} = 0.618$ (0.617, 0.619) $p = 2.9340 * 10^{-13}$	$\hat{A} = 0.481$ (0.480, 0.482) $p = 0.2272$	$\hat{A} = 0.484$ (0.483, 0.485) $p = 0.3305$	$\hat{A} = 0.631$ (0.630, 0.632) $p = 4.9544 * 10^{-16}$	$\hat{A} = 0.488$ (0.487, 0.489) $p = 0.4596$	$\hat{A} = 0.500$ (0.499, 0.501) $p = 0.9917$	$\hat{A} = 0.631$ (0.630, 0.632) $p = 4.8169 * 10^{-16}$	$\hat{A} = 0.559$ (0.558, 0.560) $p = 0.0002$	$\hat{A} = 0.559$ (0.558, 0.560) $p = 0.0003$	$\hat{A} = 0.559$ (0.558, 0.560) $p = 0.0002$

TABLE 7.9: Hypothesis Test Results of Difference in CAK/TTK

Cumulative Crash Counts

Figures 7.17 and 7.18 show the cumulative number of unique (based on our deduplication criteria) crashes triggered by each configuration for each repo. These figures suggest that, although pure ACL2s does quite well in detecting crashes, there are other approaches that do equally well or better. It is worth recalling again, however, that the stack trace deduplication methology has issues [70], but the relative performances of the different configurations correlate with their cumulative coverages.

Figure 7.19 shows the cumulative crash counts for mypy in the auxiliary evaluation. Although approaches which have higher `acl2s_reachout_freq`–and are therefore more similar to `acl2s`–perform better in terms of coverage and CAK/TTK, this trend does not necessarily carry over to the number of unique crashes found. This corroborates Klees et al. [70] who suggest that code coverage, while important, is not the only way a fuzzer should be evaluated.

Crash Survival Analysis

Now, we compare survival curves for each configuration. In order for this comparison to be statistically justified, we make the assumption that the survival times of all detected crashes across all independent trials and repositories are identically distributed. This assumption is justified by the further assumption that the sample of repositories, and therefore the function candidates in them, are representative of the general population. Additionally, in survival analysis, the set of entities for which we are measuring survival is typically known. In our case, the full set of crashes is unknown, so we use as a proxy the union of all crashes found in any independent trial in any repository in the configuration. Figure 7.20 shows these survival curves. The shaded regions are 95% confidence intervals. Note that the y-axis is given in terms of

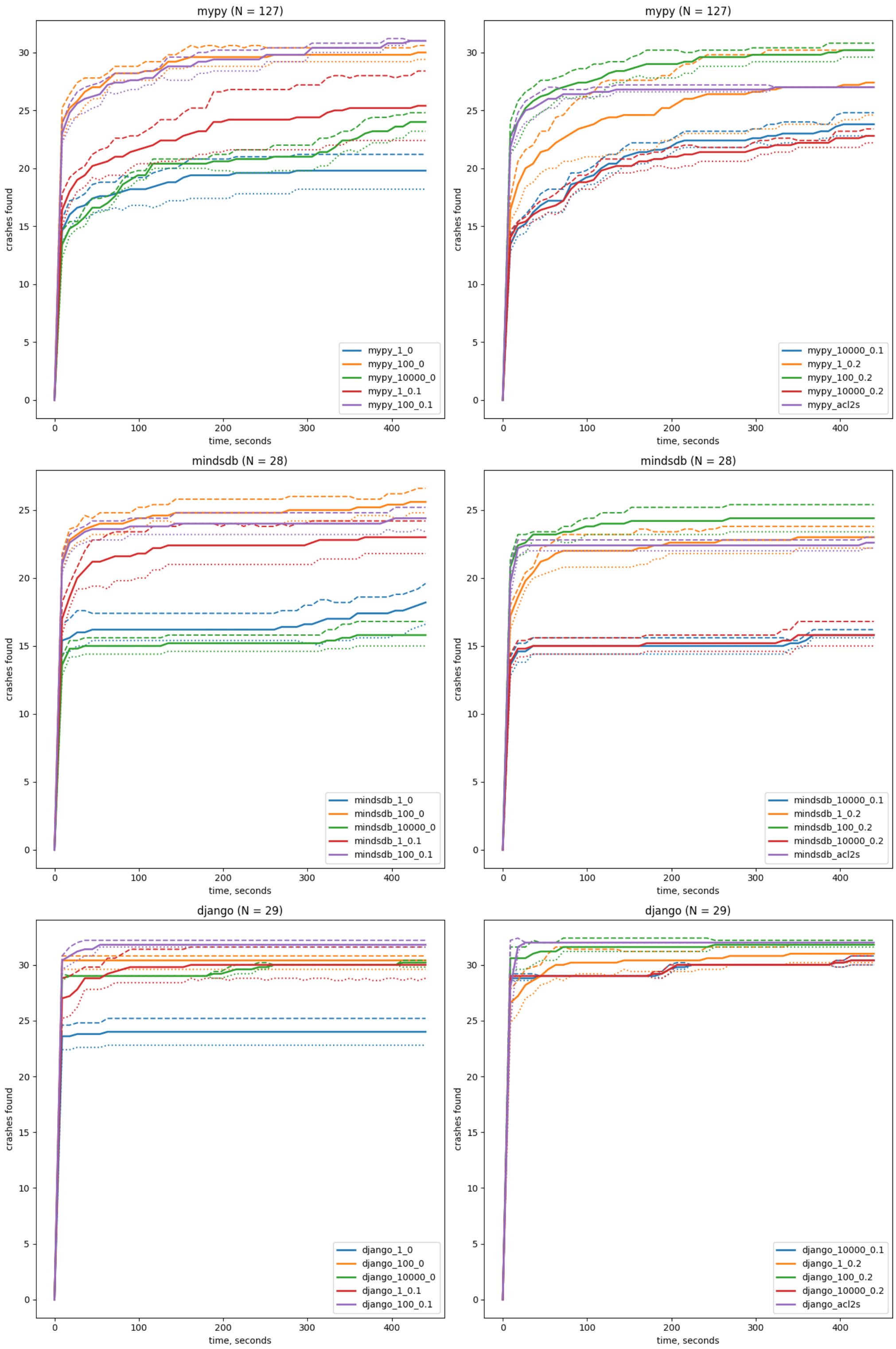

FIGURE 7.17: RQ2 Cumulative Crashes: mypy, mindsdb, django

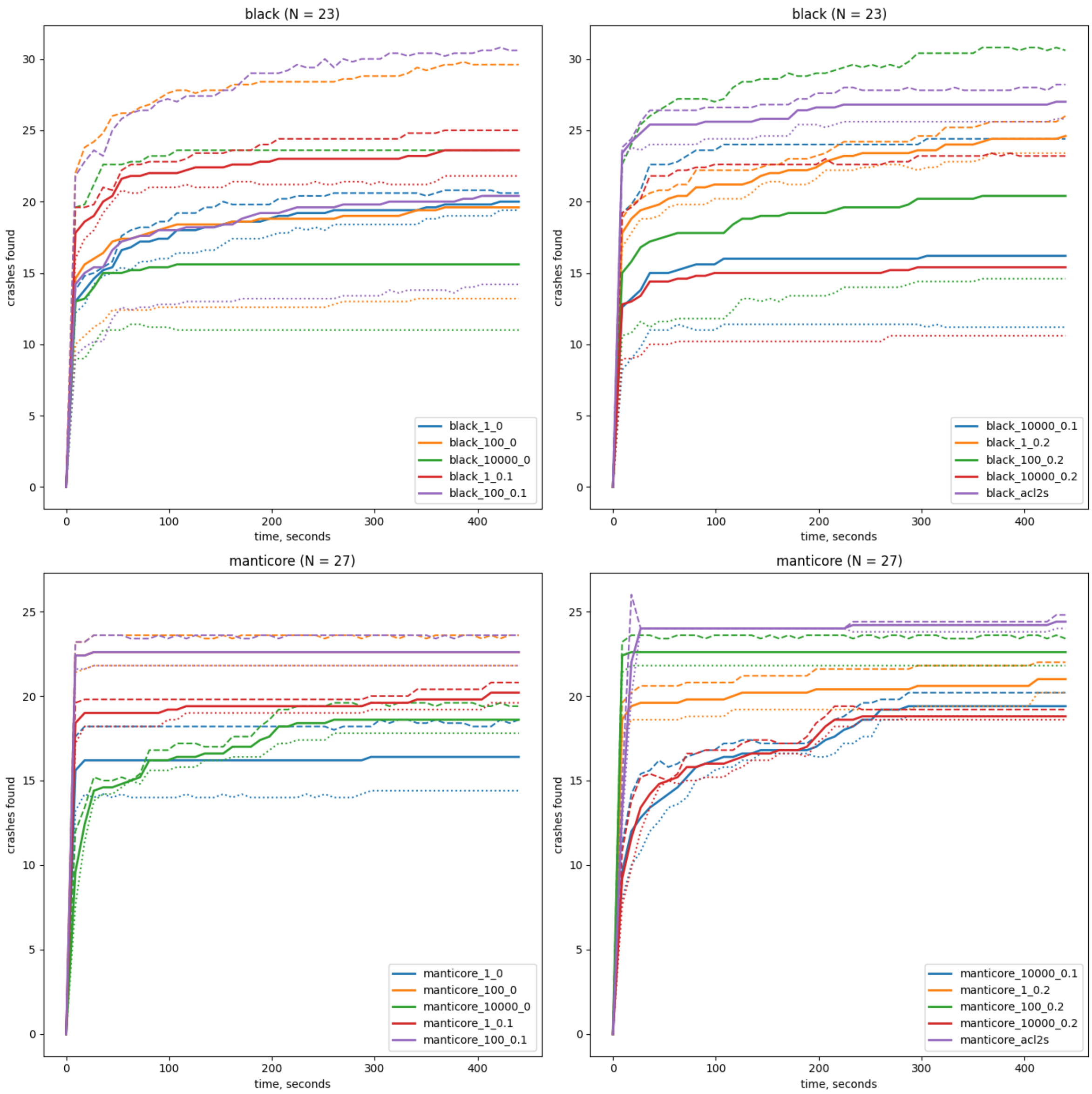

FIGURE 7.18: RQ2 Cumulative Crashes: black, manticore

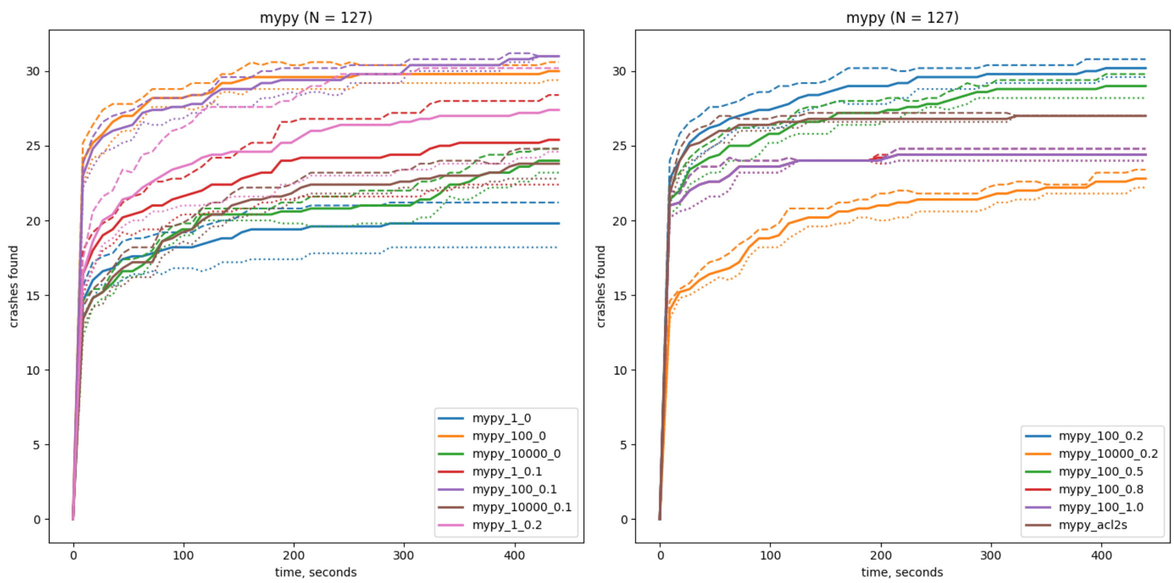

FIGURE 7.19: RQ2 Auxiliary Eval. Cumulative Crashes

a theoretical α, which can be thought of as the probability that an unknown bug survives the 440 seconds of fuzzing. For the purposes of the following RMST analysis, we assume $\alpha = 0$. It is clear visually from this figure that the configuration with $|C| = 100$ and `acl2s_reachout_freq` $= 0$ has the lowest survival curve.

The auxiliary evaluation on *mypy* also exhibits this pattern. In Figure 7.21, we observe the opposite pattern when compared to code coverage: crashes seem to be triggered more quickly with lower values of `acl2s_reachout_freq`. These results are counterintuitive, because it is expected that code coverage correlates, albeit sometimes weakly [70], with crashes triggered. This demands further exploration that we could not perform in this work due to time constraints.

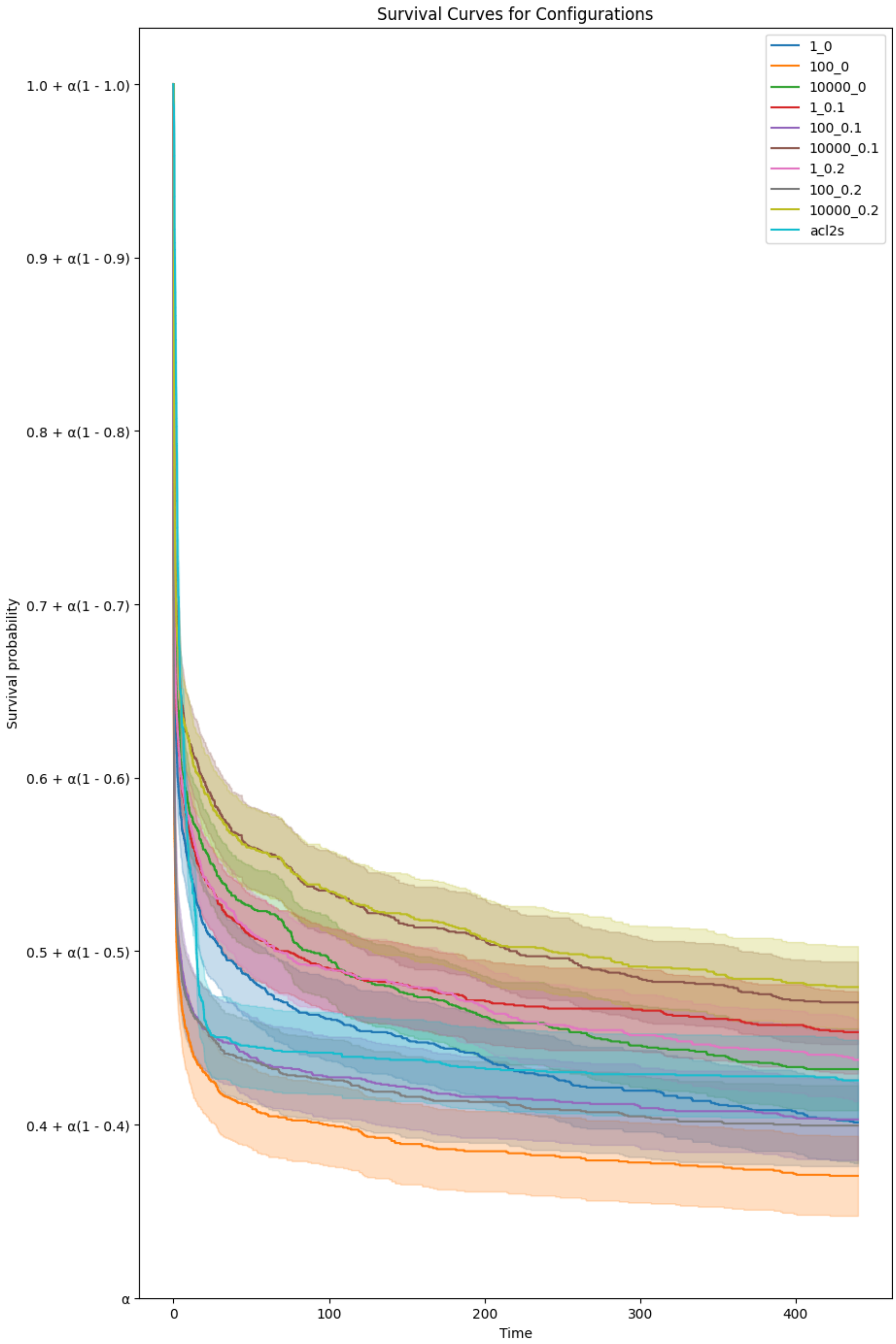

FIGURE 7.20: RQ2 Survival Curves

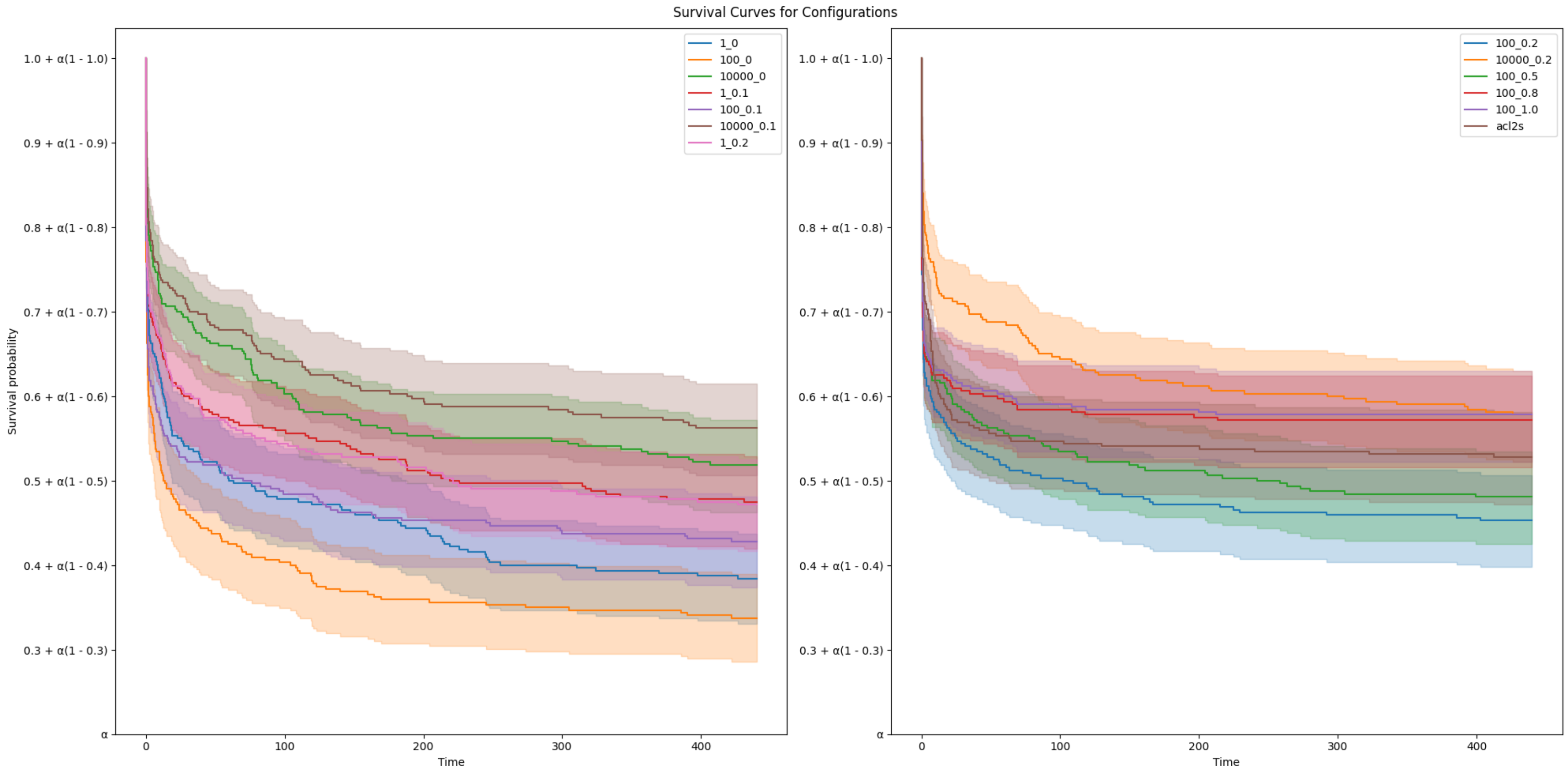

FIGURE 7.21: RQ2 Auxiliary Eval. Survival Curves

RMST Comparison

Table 7.10 shows the results of pairwise statistical hypothesis tests of difference in RMST between configurations of the tool. Significant results after Holm-Bonferroni correction [60] are highlighted in green. The first line in each cell shows the estimate of the difference, the second line shows the lower and upper values of 95% confidence intervals, and the third line shows the p-value. The differences in RMST are calculated as $row - col$. A negative value, therefore, indicates that the configuration of the row label has lower RMST, and vice versa for positive values. Recall that lower values of RMST are better, as this indicates that crashes "survive" for a shorter amount of time.

The results for the auxiliary evaluation, presented in a similar way, are given in Table 7.11. The consistent signal across both the mainline evaluation and the auxiliary evaluation is that `100_0` has statistically significantly lower RMST for the known bugs when compared to almost every other configuration. Notable exceptions are `100_0.1` and `100_0.2`.

The findings here indicate that $|C| = 100$, with low (≤ 0.2) `acl2s_reachout_freq` produce the best results in crash discovery, but again, more work is needed to understand the main factors behind these results. Possible avenues of future exploration include performing an audit of the exception deduplication methodology and looking for patterns in the classes of exceptions that each configuration triggers.

Finding 7 The results for crash discovery indicate that $|C| = 100$, with `acl2s_reachout_freq` ≤ 0.2 produce the best results. More work is needed, however, to fully understand these effects with a more comprehensive evaluation.

	1_0	100_0	10000_0	1_0.1	100_0.1	10000_0.1	1_0.2	100_0.2	10000_0.2
100_0	-23.095 $(-37.252, -8.939)$ $p = 0.0014$								
10000_0	13.419 $(-0.735, 27.572)$ $p = 0.0631$	36.514 $(22.355, 50.673)$ $p = 4.3196 * 10^{-07}$							
1_0.1	16.749 $(2.469, 31.028)$ $p = 0.0215$	39.844 $(25.559, 54.129)$ $p = 4.5810 * 10^{-08}$	3.330 $(-10.952, 17.612)$ $p = 0.6477$						
100_0.1	-9.476 $(-23.738, 4.787)$ $p = 0.1929$	13.620 $(-0.648, 27.888)$ $p = 0.0614$	-22.894 $(-37.160, -8.629)$ $p = 0.0017$	-26.224 $(-40.614, -11.835)$ $p = 0.0004$					
10000_0.1	30.770 $(16.604, 44.937)$ $p = 2.0701 * 10^{-05}$	53.866 $(39.694, 68.038)$ $p = 9.3660 * 10^{-14}$	17.352 $(3.182, 31.521)$ $p = 0.0164$	14.022 $(-0.273, 28.316)$ $p = 0.0545$	40.246 $(25.968, 54.524)$ $p = 3.3000 * 10^{-08}$				
1_0.2	13.722 $(-0.493, 27.937)$ $p = 0.0585$	36.817 $(22.597, 51.038)$ $p = 3.8887 * 10^{-07}$	0.303 $(-13.915, 14.522)$ $p = 0.9666$	-3.027 $(-17.370, 11.316)$ $p = 0.6792$	23.198 $(8.872, 37.524)$ $p = 0.0015$	-17.048 $(-31.279, -2.818)$ $p = 0.0189$			
100_0.2	-11.361 $(-25.604, 2.883)$ $p = 0.1180$	11.735 $(-2.514, 25.984)$ $p = 0.1065$	-24.779 $(-39.026, -10.532)$ $p = 0.0007$	-28.109 $(-42.480, -13.738)$ $p = 0.0001$	-1.885 $(-16.239, 12.470)$ $p = 0.7969$	-42.131 $(-56.390, -27.872)$ $p = 6.9967 * 10^{-09}$	-25.082 $(-39.390, -10.775)$ $p = 0.0006$		
10000_0.2	32.509 $(18.311, 46.708)$ $p = 7.2026 * 10^{-06}$	55.605 $(41.401, 69.809)$ $p = 1.6847 * 10^{-14}$	19.091 $(4.889, 33.292)$ $p = 0.0084$	15.761 $(1.434, 30.087)$ $p = 0.0311$	41.985 $(27.675, 56.295)$ $p = 8.8925 * 10^{-09}$	1.739 $(-12.475, 15.953)$ $p = 0.8105$	18.787 $(4.525, 33.050)$ $p = 0.0098$	43.870 $(29.579, 58.161)$ $p = 1.7816 * 10^{-09}$	
acl2s	-0.474 $(-14.757, 13.809)$ $p = 0.9481$	22.622 $(8.333, 36.910)$ $p = 0.0019$	-13.892 $(-28.178, 0.393)$ $p = 0.0566$	-17.222 $(-31.632, -2.812)$ $p = 0.0192$	9.002 $(-5.391, 23.395)$ $p = 0.2203$	-31.244 $(-45.542, -16.946)$ $p = 1.8448 * 10^{-05}$	-14.196 $(-28.542, 0.151)$ $p = 0.0525$	10.887 $(-3.488, 25.261)$ $p = 0.1377$	-32.983 $(-47.313, -18.653)$ $p = 6.4453 * 10^{-06}$

TABLE 7.10: RQ2 Hypothesis Test Results of Difference in RMST

	1_0	100_0	10000_0	1_0.1	100_0.1	10000_0.1	1_0.2	100_0.2	10000_0.2	100_0.5	100_0.8	100_1.0
100_0	-29.514 $(-61.033, 2.005)$ $p = 0.0665$											
10000_0	58.176 $(26.563, 89.789)$ $p = 0.0003$	87.690 $(56.186, 119.195)$ $p = 4.8865 * 10^{-08}$										
1_0.1	35.179 $(3.223, 67.134)$ $p = 0.0310$	64.692 $(32.845, 96.540)$ $p = 6.8547 * 10^{-05}$	-22.998 $(-54.939, 8.943)$ $p = 0.1582$									
100_0.1	9.398 $(-22.757, 41.553)$ $p = 0.5667$	38.912 $(6.864, 70.961)$ $p = 0.0173$	-48.778 $(-80.919, -16.637)$ $p = 0.0029$	-25.780 $(-58.258, 6.697)$ $p = 0.1198$								
10000_0.1	74.314 $(42.851, 105.776)$ $p = 3.6678 * 10^{-06}$	103.828 $(72.474, 135.181)$ $p = 8.5596 * 10^{-11}$	16.137 $(-15.311, 47.586)$ $p = 0.3145$	39.135 $(7.343, 70.927)$ $p = 0.0158$	64.915 $(32.922, 96.908)$ $p = 6.9833 * 10^{-05}$							
1_0.2	33.058 $(1.076, 65.040)$ $p = 0.0428$	62.572 $(30.697, 94.447)$ $p = 0.0001$	-25.118 $(-57.086, 6.850)$ $p = 0.1236$	-2.120 $(-34.427, 30.186)$ $p = 0.8976$	23.660 $(-8.844, 56.164)$ $p = 0.1537$	-41.256 $(-73.075, -9.436)$ $p = 0.0110$						
100_0.2	17.015 $(-15.239, 49.270)$ $p = 0.3012$	46.529 $(14.381, 78.677)$ $p = 0.0046$	-41.161 $(-73.401, -8.921)$ $p = 0.0123$	-18.163 $(-50.739, 14.413)$ $p = 0.2745$	7.617 $(-25.155, 40.389)$ $p = 0.6487$	-57.298 $(-89.391, -25.205)$ $p = 0.0005$	-16.043 $(-48.645, 16.560)$ $p = 0.3348$					
10000_0.2	79.438 $(47.937, 110.940)$ $p = 7.7131 * 10^{-07}$	108.952 $(77.559, 140.345)$ $p = 1.0296 * 10^{-11}$	21.262 $(-10.225, 52.749)$ $p = 0.1857$	44.260 $(12.429, 76.090)$ $p = 0.0064$	70.040 $(38.008, 102.071)$ $p = 1.8219 * 10^{-05}$	5.124 $(-26.212, 36.461)$ $p = 0.7486$	46.380 $(14.522, 78.238)$ $p = 0.0043$	62.423 $(30.292, 94.554)$ $p = 0.0001$				
100_0.5	31.528 $(-0.639, 63.696)$ $p = 0.0547$	61.042 $(28.981, 93.103)$ $p = 0.0002$	-26.648 $(-58.802, 5.505)$ $p = 0.1043$	-3.650 $(-36.140, 28.840)$ $p = 0.8257$	22.130 $(-10.557, 54.816)$ $p = 0.1845$	-42.786 $(-74.791, -10.780)$ $p = 0.0088$	-1.530 $(-34.047, 30.987)$ $p = 0.9265$	14.513 $(-18.272, 47.297)$ $p = 0.3856$	-47.910 $(-79.954, -15.866)$ $p = 0.0034$			
100_0.8	60.045 $(27.636, 92.455)$ $p = 0.0003$	89.559 $(57.256, 121.863)$ $p = 5.5150 * 10^{-08}$	1.869 $(-30.526, 34.264)$ $p = 0.9100$	24.867 $(-7.863, 57.596)$ $p = 0.1365$	50.647 $(17.722, 83.571)$ $p = 0.0026$	-14.268 $(-46.517, 17.980)$ $p = 0.3858$	26.987 $(-5.769, 59.743)$ $p = 0.1064$	43.030 $(10.008, 76.052)$ $p = 0.0106$	-19.393 $(-51.679, 12.894)$ $p = 0.2391$	28.517 $(-4.420, 61.454)$ $p = 0.0897$		
100_1.0	62.839 $(30.471, 95.207)$ $p = 0.0001$	92.353 $(60.091, 124.615)$ $p = 2.0172 * 10^{-08}$	4.663 $(-27.692, 37.017)$ $p = 0.7776$	27.660 $(-5.028, 60.349)$ $p = 0.0972$	53.441 $(20.556, 86.325)$ $p = 0.0014$	-11.475 $(-43.682, 20.733)$ $p = 0.4850$	29.781 $(-2.934, 62.496)$ $p = 0.0744$	45.824 $(12.842, 78.805)$ $p = 0.0065$	-16.599 $(-48.845, 15.646)$ $p = 0.3130$	31.311 $(-1.586, 64.207)$ $p = 0.0621$	2.794 $(-30.339, 35.927)$ $p = 0.8687$	
acl2s	44.034 $(11.559, 76.509)$ $p = 0.0079$	73.548 $(41.179, 105.917)$ $p = 8.4545 * 10^{-06}$	-14.142 $(-46.603, 18.319)$ $p = 0.3932$	8.855 $(-23.939, 41.650)$ $p = 0.5966$	34.636 $(1.647, 67.625)$ $p = 0.0396$	-30.280 $(-62.594, 2.035)$ $p = 0.0663$	10.976 $(-21.845, 43.797)$ $p = 0.5122$	27.019 $(-6.067, 60.105)$ $p = 0.1095$	-35.404 $(-67.756, -3.052)$ $p = 0.0320$	12.506 $(-20.495, 45.507)$ $p = 0.4576$	-16.011 $(-49.248, 17.226)$ $p = 0.3451$	-18.805 $(-52.002, 14.392)$ $p = 0.2669$

TABLE 7.11: RQ2 Auxiliary Eval. Hypothesis Test Results of Difference in RMST

7.3.3 RQ3

Now that we have analyzed the performance of the custom encoding in RQ1, and the effects of tool configuration in RQ2, we ask our third research question, "Does the tool find issues developers care about?" Recall from Section 7.2.3 that to produce the pool of exceptions, we aggregated all crashes found in at least one independent trial for all of the configurations evaluated in RQ2 (excluding the auxiliary evaluation), and deduplicated them according to stack trace-based deduplication (with N, the number of most recent stack frames to compare, equal to 3). This yielded 341 crashes, which we manually analyzed and categorized. We filed 16 bug reports, many of which cover several of the individual crashes (recall from Klees et al. [70] that stack trace-based deduplication is imperfect–in our case, we observed overcounting of logical "bugs"). We decided to invest manual effort into collapsing crashes with similar causes into single bug reports primarily to be courteous to the repository maintainers, so that the they are not inundated with many bug reports that are very similar.

Reported Crashes

Table 7.12 shows the outcomes of the reported bugs as of the time of this writing. Recall that Table 7.1 gives the definitions of the abbreviations used in the column headers. If none of the categories are checked, this indicates that the crash's categorization is unknown because the bug report has not received a response from the community at the time of this writing. We have included the URLs of each report as hyperlinks in the table so that the reader may check the reports for updates. The full data for the 341 crashes, with stack traces, as well as lists of the experimental configurations that found each crash, are given in Appendix B.

Community Feedback

At the time of this writing, we have received responses for 22 of the crashes, which equates to 6 bug reports. Unfortunately, although we made the most reports in the manticore repository, we have not received responses for any of them; the repository page indicates that manticore is in "maintenance mode," so its community is presumably not very active. We have had reports accepted as legitimate by the community in two of the five repositories, mypy and mindsdb. Listing 7.1 shows a stacktrace of one of the accepted mypy bugs that for which we were able to find a proof-of-concept reproduction from the top-level. This bug manifests when mypy is attempting to analyze a source file that contains a division expression between two integers. It has constant folding logic built in to attempt to resolve these expressions, presumably to aid in type analysis. When this constant folding logic tries to divide an extremely large integer (integers are arbitrary precision in Python) by a small integer, it cannot fit the result into a float (which is usually a machine double-precision floating point number, not arbitrary precision), and mypy crashes with an unhandled exception, resulting in a bad user experience. The fix that the community has suggested is to gracefully handle these exceptions and print a more meaningful error message.

```
Traceback (most recent call last):
  File ".../fuzzer/atheris_run.py", line 137, in harness
    retval = fn(*args, *star_args, **kwargs_copy, **double_star_kwargs)
  File ".../repos/mypy/mypy/constant_fold.py", line 123, in constant_fold_binary_int_op
    return left / right
OverflowError: integer division result too large for a float
```

LISTING 7.1: mypy OverflowError in constant folding logic

Listing 7.2 shows an excerpt of the pull request we submitted to mypy to fix the return type mismatch that we detected in `mypy.stubtest.parse_options`. This pull request was accepted and merged, and is now part of mypy's codebase.

```
@@ -1878,8 +1878,8 @@ class _Arguments:
   allowlist: list[str]
   generate_allowlist: bool
   ignore_unused_allowlist: bool
-  mypy_config_file: str
-  custom_typeshed_dir: str
+  mypy_config_file: str | None
+  custom_typeshed_dir: str | None
   check_typeshed: bool
   version: str
```

LISTING 7.2: Diff of pull request submitted to fix detected type error in mypy

Listing 7.3 shows the traceback of the crash detected in mindsdb that was accepted. The fix, which was to ensure that the value of the `num_packets` variable is an integer rather than a float, was generated by a member of the community.

```
Traceback (most recent call last):
  ...
  File ".../data/repos/mindsdb/mindsdb/api/mysql/mysql_proxy/data_types/mysql_packet.py", line 163, in test
    pprint.pprint(Packet.bodyStringToPackets("abdds")[0].get_packet_string())
  File ".../data/repos/mindsdb/mindsdb/api/mysql/mysql_proxy/data_types/mysql_packet.py",
      line 148, in bodyStringToPackets
    for i in range(num_packets):
TypeError: 'float' object cannot be interpreted as an integer
```

LISTING 7.3: Diff of pull request submitted to fix detected type error in mypy

These accepted reports are a promising indicator that this tool is capable of finding meaningful issues in Python code. On the other hand, other responses have been critical of whether the crashes that were detected indicate legitimate bugs. Jelle Zijlstra, a key contributor to both mypy and black, and to the Python interpreter itself, wrote the following in response to a report of `OverflowErrors` and `MemoryErrors` when using Python's string multiplication functionality:

> Just to expand on this (in case it's helpful for your research project): Your post reads as if you assume it is a problem if a Python function can raise an exception. That's not how Python is generally written; almost every operation can raise an exception, and robust code handles exceptions that can reasonably be expected to appear (say, an HTTPError when making an HTTP request), but not everything that could possibly happen. Every memory allocation could raise MemoryError and every function call could raise RecursionError; nobody is going to catch those.

> In this case, the functions you flagged use string multiplication for creating an indentation string. Obviously, that's only expected to indent by a relatively small number of spaces in normal operation. We could add an explicit check `if n > some_big_number: raise ValueError` to catch a possible bug where someone passes a huge number, but that wouldn't really make for a more useful error than the OverflowError you'll get now.
>
> - @JelleZijlstra on GitHub

This is useful context to understand how Python programmers think of exceptions. When reporting a bug, two of the fields that are often required to be filled out are "expected behavior" and "actual behavior." In this situation, as Jelle pointed out, the expected behavior is not altogether different from the actual behavior: the functions-under-test received input that caused an invalid memory allocation or integer overflow, and the language reported this in a graceful way by throwing an exception that can be handled from within the language. In some safety critical settings, there may be a requirement that the software cannot crash unexpectedly. In these situations, exceptions which are not explicitly accounted for and caught would become more of an issue, but in the context of type-checking software, where typical inputs to the program are within some reasonable input space, this is not a problem.

A similar response was received from another community member on a bug report in black related to a `MemoryError`:

> I'm not familiar with the `ipynb` format, but I assume the code is the string in source, the `(((...(((t)))...)))`. If so, then it makes sense that it would fail, since that is causing a bunch of expressions, though you probably have a low memory allocation to whatever is running black. The playground formats that many just fine, and copy pasting more eventually gives a too many parenthesis for safe mode error.
>
> If someone did encounter this in actual code, I'd recommend allocating more memory, splitting the expression up, or skipping that line with `# fmt: skip / off` and `on`.

> I can't speak for all of black's dev team, I just lurk in the issues, but that many nested parenthesis looks to me like enough of an edge case that reducing the memory used isn't a pressing concern, given that it requires both an obscene amount and low memory availability.
>
> - @MeGaGiGaGon on GitHub

The root cause of the crash here is actually a stack overflow in the Python parser, caused by many nested parentheses. In normal idiomatic Python code, this many nested parentheses would be extremely rare to find, and probably difficult to pass through code review, although a malicious actor could in theory cause unexpected crashes of, say, a continuous integration pipeline by inserting such code. In any case, given the context, it seems reasonable that the community would respond in this way.

It is worth pointing out that we are not the only ones to have encountered such responses to bugs found in fuzzing from the community. Some of the bugs reported as part of PyRTFuzz [72] were also closed as illegitimate issues. A particularly interesting discussion can be found in *RecursionError in pyclbr.readmodule_ex* [105]. The reported crash is a recursion error when a library function that attempts to import a package receives an extremely long import path, but the feedback from the community was clear: although it is possible that such a crash could be exploited by attackers to induce an application crash, it is not the responsibility of the Python interpreter to guard all application code against attacks. Furthermore, one of the commenters introduced the notion of the RecursionError in this case being "implementation-specific." That is, there is a difference between properties of the language specification and properties of the language implementation, and each can have different categories of exceptions that it raises. The last comment on the post succinctly summarizes the perspective of the community in their response to this crash report:

> You can also solve this "issue" by increasing the recursion limit (if you really want to).
>
> I agree that this seems to work as expected. Garbage in - nice error out.

- @sobolevn on GitHub

We do not construe these responses as negative, but rather as a logical and practical interpretation of what issues really matter to the average Python programmer. This is, of course, far from a settled debate–allowing unhandled exceptions to propagate can lead to denial-of-service, and potentially more severe security vulnerabilities and information leakage. There is certainly a case to be made for more defensive programming and better documentation of what errors can be thrown in Python so that programmers do not unintentionally expose themselves to these issues.

Furthermore, in light of the community feedback, we hope to improve the usefulness of the feedback our tool provides in future work by 1) using control flow analysis to understand automatically how crashes can be reproduced from top-level APIs, 2) exploring the use of generative AI, or other more intelligent heuristics, to automatically filter out exceptions that are unlikely to differ from expected behavior of the code, or are handled at a higher level and 3) using generative AI to propose fixes, such as inserting proper exception handling code when it is missing.

Generative AI Case Study

As just mentioned, we are motivated in future work to explore, among other avenues, the use of generative AI to triage crashes and propose fixes for them. There is precedent for using generative AI for automated bug triage and remediation in the literature as well: Sergeyuk et al. [114] study the ways developers are interacting with coding "copilot" tools. They found that 43.7% of respondents use generative AI assistants for determining potential causes of bugs, and 42.4% report using generative AI assistants for generating code for potential bug fixes. However, some respondents expressed concern about the fact that effective bug triaging requires a lot of context. As context windows for large language models become larger [31], and new techniques for more efficiently utilizing context emerge [100], these models continue to become more viable for large-context tasks such as bug triaging, where entire codebases could potentially be provided as context.

We performed a small case study to gain some intuition around how promising augmenting TYPE HINT FUZZING with generative AI might be. Using the first negative report in the *mypy* repository given previously, where Jelle Zijlstra provided an explanation, we prompted GPT-4o [90], OpenAI's latest flagship model at the time of this writing, giving it the following prompts (paraphrased) in the same conversation:

- P1: given the bug report that was originally written, and the source code of the function in which the exception occurred, explain the problem, why it warrants a fix, and propose a solution
- P2: Make your strongest argument to the *mypy* maintainers for why this issue is worth considering and fixing
- P3: Respond as the *mypy* maintainers, making your strongest argument for why the issue is *not* worth considering and fixing
- P4: Evaluate which argument is stronger and make a recommendation

The response to P1 is given in Figure 7.22. The issue analysis is largely correct. The causes of the exceptions are correctly identified, and the impacts are mostly correct, except for the subtle detail that the *MemoryError* is handled by the language runtime such that an out-of-memory condition is not triggered. However, the fact that such a large allocation is even attempted to be made poses significant risk if the implementations of the language interpreter or runtime libraries are not disciplined in their memory management. The reasoning of why this issue warrants a fix also aligns with security best practices by implying vulnerability to denial-of-service and suggesting better input validation (for "edge cases"). GPT-4o also ties this back to the end user, which maintainers are likely to care about, by highlighting that uncaught exceptions harm the user experience.

In the response to P1, GPT-4o also proposes fixes to the issue. The first proposal, given in Figure 7.23, is to remedy the issue with input validation–note the validation step and `raise ValueError` statement. The second proposal, given in Figure 7.24, is to use "graceful degradation," where the input is validated and constrained to a reasonable limit. If a constraint action takes place, a warning is emitted to the logs. These are both reasonable solutions, each taking up few lines of code, and they effectively solve the problem.

The ability of GPT-4o to generate reasonable and correct fixes for this problem shows promise in generative AI's ability to understand and fix issues (this is a simple example, however). After observing this response, we prompted GPT-4o with P2, to have it argue in favor of fixing this issue. Figures 7.25 and 7.26 contain the argument. We find the four pillars of the argument, *future proofing*, *defensive programming*, *encouraging best practices*, and *minimal cost for long-term benefits* quite compelling. The overarching theme of strong input validation and defensive programming align well with cybersecurity best practices, such as those enumerated in OWASP's Top Ten [94].

We then prompted GPT-4o, in the same chat window, with P3, to have it argue the other side, in favor of not fixing the issue. Figures 7.27 and 7.28 contain this response. This argument is less compelling to us, especially given how trivial the fix for this issue is (2-4 lines of code). However, taking the point of view of a maintainer who is dealing with large amounts of bugs and feature requests every day, the case to ignore small trivialities like this example, in aggregate, may be beneficial to the product as a whole, as more time is devoted to key objectives that move the project along its roadmap.

Finally, we asked GPT-4o to decide which side of the argument it thought to be more compelling. Figure 7.29 contains GPT-4o's determination. In this instance, GPT-4o took the side of the maintainers, in accordance with what the true response was from *mypy*'s maintainer, Jelle Zijlstra.

This case study indicates that GPT-4o can correctly identify and fix issues in code with relevant and justifiable solutions when provided sufficient context about the symptoms of the problem. It also displays strong argumentation skills, and it can even come to its own conclusions as to which argument is more compelling. In this case, if we had hooked GPT-4o into the crash analysis pipeline, this crash would've been filtered out, in agreement with the *mypy* maintainers. Of course, more rigorous study is needed; results for other crashes may not be so promising, and LLM performance can vary from run to run. We also did not use the context window to its fullest extent–we only provided the function in which the exception occurred for codebase-level context. However, this case study does suggest that there is great potential in this approach, and further motivates future work in this area.

Other Observations

Although we detected 341 crashes, only a small subset of them had potential to be semantic bugs, and were reported. 151 of the 341 crashes, nearly half, fall into the broad category of "Type Annotation Enhancement Opportunity" (abbreviated as AE, see Table 7.1). Recall that when the tool detects an "Any" type annotation (this can either be an explicit annotation, or what the type extraction defaults to when there is no type annotation), it attempts to exercise it by passing in inputs across a variety of types. "Type Annotation Enhancement Opportunity" crashes therefore are crashes caused by the function implicitly expecting a type that is more specific than "Any." See Appendix B for the full data sheet of these, and other, crashes. As we will discuss more in Chapter 8, we would like to explore using the information from these crashes to offer type annotation suggestions to users of the tool in future work.

```
Traceback (most recent call last):
  File ".../fuzzer/atheris_run.py", line 137, in harness
    retval = fn(*args, *star_args, **kwargs_copy, **double_star_kwargs)
  File ".../repos/mypy/mypy/messages.py", line 2997, in format_string_list
    assert lst
AssertionError
```

LISTING 7.4: Violation of assertion to enforce that a list is non-empty in mypy

Another 32 of the crashes were `AssertionErrors`. In our analysis, we observed a practice across Python repositories where `assert` statements (which produce these errors upon violation) are used to enforce additional type constraints that cannot be expressed through the standard system of type hints. Listing 7.4 shows an example of this practice in *mypy*, and Listing 7.5 shows an example in *black*.

```
Traceback (most recent call last):
  File ".../fuzzer/atheris_run.py", line 137, in harness
    retval = fn(*args, *star_args, **kwargs_copy, **double_star_kwargs)
  File ".../repos/black/src/black/handle_ipynb_magics.py", line 171, in
      get_token
    assert magic
AssertionError
```

LISTING 7.5: Violation of assertion to enforce that a string is non-empty in black

This practice, which is commonplace, suggests that the existing system of type hints is insufficient for all types that Python programmers want to express. We offer ideas for expansion of the type annotation system in Chapter 8.

Finding 8 The tool indeed is capable of finding bugs that developers care about, as shown by the 3 reports that were accepted, of which 2 have had fixes implemented. There is still work to be done on filtering out noise in crash reports, and utilizing the information from crashes which are not caused by semantic bugs to improve the type annotations of the code-under-test.

Repository	Exception Type	EX	FO	EE	ER	LB	AE	TE	Report Link
mypy	OverflowError			✓					https://github.com/python/mypy/issues/17454
mypy	MemoryError			✓					https://github.com/python/mypy/issues/17454
mypy	OverflowError			✓					https://github.com/python/mypy/issues/17454
mypy	MemoryError			✓					https://github.com/python/mypy/issues/17454
mypy	MemoryError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	OverflowError				✓	✓			https://github.com/python/mypy/issues/17008
mypy	N/A					✓		✓	https://github.com/python/mypy/pull/16897
mindsdb	TypeError					✓			https://github.com/mindsdb/mindsdb/issues/9426
mindsdb	OverflowError			✓					https://github.com/mindsdb/mindsdb/issues/9467
black	MemoryError			✓	✓				https://github.com/psf/black/issues/4400
black	MemoryError			✓					https://github.com/psf/black/issues/4400
black	MemoryError			✓					https://github.com/psf/black/issues/4400
black	MemoryError			✓					https://github.com/psf/black/issues/4400
black	MemoryError			✓					https://github.com/psf/black/issues/4400
black	MemoryError			✓					https://github.com/psf/black/issues/4400
black	MemoryError			✓					https://github.com/psf/black/issues/4400
manticore	TypeError								https://github.com/trailofbits/manticore/issues/2660
manticore	OverflowError								https://github.com/trailofbits/manticore/issues/2660
manticore	ValueError								https://github.com/trailofbits/manticore/issues/2660
manticore	MemoryError								https://github.com/trailofbits/manticore/issues/2660
manticore	ValueError								https://github.com/trailofbits/manticore/issues/2651
manticore	OverflowError								https://github.com/trailofbits/manticore/issues/2651
manticore	MemoryError								https://github.com/trailofbits/manticore/issues/2651
manticore	RecursionError								https://github.com/trailofbits/manticore/issues/2652
manticore	ValueError								https://github.com/trailofbits/manticore/issues/2653
manticore	OverflowError								https://github.com/trailofbits/manticore/issues/2653
manticore	MemoryError								https://github.com/trailofbits/manticore/issues/2653
manticore	ValueError								https://github.com/trailofbits/manticore/issues/2654
manticore	RecursionError								https://github.com/trailofbits/manticore/issues/2652
manticore	ValueError								https://github.com/trailofbits/manticore/issues/2655
manticore	ZeroDivisionError								https://github.com/trailofbits/manticore/issues/2656
manticore	ValueError								https://github.com/trailofbits/manticore/issues/2654
manticore	MemoryError								https://github.com/trailofbits/manticore/issues/2657
manticore	OverflowError								https://github.com/trailofbits/manticore/issues/2657
manticore	MemoryError								https://github.com/trailofbits/manticore/issues/2658
manticore	TypeError								https://github.com/trailofbits/manticore/issues/2659
manticore	OverflowError								https://github.com/trailofbits/manticore/issues/2658
manticore	MemoryError								https://github.com/trailofbits/manticore/issues/2658
manticore	OverflowError								https://github.com/trailofbits/manticore/issues/2658

TABLE 7.12: Exception Data with Report Links

The exceptions in the `mypy.util.soft_wrap` function, specifically the `OverflowError` and `MemoryError`, are indeed noteworthy and could be problematic in certain situations, particularly when dealing with large inputs. Here's a breakdown of the issues and possible solutions:

Issue Analysis

1. **OverflowError:**
    - **Cause:** This error occurs because the multiplication operator is used to generate a string of spaces, where the number of spaces (`num_indent`) exceeds the maximum allowable size for an integer on the system (i.e., larger than `sys.maxsize`).
    - **Impact:** While this is a rare situation in most practical applications of the `mypy` type checker, it could occur in edge cases where very large values are inadvertently passed as `num_indent`. If unhandled, this could lead to a crash, which might affect users running automated scripts or fuzz testing with extreme inputs.
2. **MemoryError:**
    - **Cause:** This error is due to the memory limitations when attempting to allocate a huge string in memory, caused by multiplying a space by an excessively large integer.
    - **Impact:** This error can lead to out-of-memory conditions, potentially causing the program to crash or become unresponsive. This is especially important for environments with limited memory resources.

Why This Warrants a Fix

- **User Experience:** Even though these errors might not occur frequently in typical usage scenarios, they can still affect users who accidentally or maliciously provide very large integers. Crashes from unhandled exceptions can degrade user experience and reliability.
- **Security:** Unhandled exceptions might be exploitable in certain environments, leading to denial-of-service attacks, particularly when `mypy` is used in automated pipelines or servers.
- **Code Quality:** Addressing these edge cases aligns with good coding practices, ensuring that the function behaves predictably even in extreme conditions.

FIGURE 7.22: Excerpt of GPT-4o conversation: Issue Identification

Proposed Fix

1. **Input Validation:**
   - **Solution:** Introduce validation logic to check the size of `num_indent` before proceeding with the multiplication operation. If `num_indent` exceeds a reasonable threshold (e.g., `sys.maxsize // 2`), raise a custom error or handle it gracefully by capping the value.

python Copy code

```python
import sys

def soft_wrap(msg: str, max_len: int, first_offset: int, num_indent: int = 0) -
    # Validate num_indent to prevent OverflowError and MemoryError
    if num_indent > sys.maxsize // 2:
        raise ValueError(f"num_indent too large: {num_indent}")

    words = split_words(msg)
    next_line = words.pop(0)
    lines: list[str] = []
    while words:
        next_word = words.pop(0)
        max_line_len = max_len - num_indent if lines else max_len - first_offse
        if len(next_line) + len(next_word) + 1 <= max_line_len:
            next_line += " " + next_word
        else:
            lines.append(next_line)
            next_line = next_word
    lines.append(next_line)
    padding = "\n" + " " * num_indent
    return padding.join(lines)
```

FIGURE 7.23: Excerpt of GPT-4o conversation: First Proposed Fix

2. **Graceful Degradation:**

   - **Solution:** Instead of raising an error, you could cap `num_indent` to a maximum allowed value and log a warning or debug message when this occurs. This way, the function continues to operate, but with a manageable size of spaces.

python Copy code

```python
import sys
import logging

def soft_wrap(msg: str, max_len: int, first_offset: int, num_indent: int = 0) -
    max_safe_indent = sys.maxsize // 2
    if num_indent > max_safe_indent:
        logging.warning(f"num_indent capped from {num_indent} to {max_safe_inde
        num_indent = max_safe_indent

    words = split_words(msg)
    next_line = words.pop(0)
    lines: list[str] = []
    while words:
        next_word = words.pop(0)
        max_line_len = max_len - num_indent if lines else max_len - first_offse
        if len(next_line) + len(next_word) + 1 <= max_line_len:
            next_line += " " + next_word
        else:
            lines.append(next_line)
            next_line = next_word
    lines.append(next_line)
    padding = "\n" + " " * num_indent
    return padding.join(lines)
```

FIGURE 7.24: Excerpt of GPT-4o conversation: Second Proposed Fix

Even though the `num_indent` parameter in the `soft_wrap` function currently defaults to zero and isn't explicitly set anywhere else in the codebase, addressing the potential `OverflowError` and `MemoryError` is still crucial. Here's why:

1. Future-Proofing the Codebase

- **Unforeseen Usage:** Code evolves over time, and what is true today might not be true tomorrow. While `num_indent` isn't currently set to a non-zero value, future changes in the codebase, refactoring, or new feature additions might involve setting this parameter to a non-zero value. If these edge cases aren't handled now, they could become a source of bugs or crashes later.
- **Third-Party Integrations:** `mypy` is an open-source project used by a wide array of developers. While the internal codebase might not currently set `num_indent` explicitly, developers who extend or integrate `mypy` into other tools might do so. They could pass large values to `num_indent`, leading to unexpected crashes if the issue is not addressed.

2. Defensive Programming

- **Robustness and Reliability:** Defensive programming is about anticipating potential issues and making the code resilient to them. By addressing these potential exceptions now, you ensure that the function behaves reliably under all circumstances, even those that may not be immediately apparent. This approach is a hallmark of high-quality, maintainable code.
- **Error Containment:** Handling the possibility of `OverflowError` and `MemoryError` ensures that if unexpected input ever reaches this function, it won't cause a cascading failure that could affect other parts of the system. Instead, the function can either handle the input gracefully or fail fast with a clear and informative error.

FIGURE 7.25: Excerpt of GPT-4o conversation: Pro-Fix Argument

3. Encouraging Best Practices

- **Signal to Contributors:** Fixing this issue sends a message to contributors that code quality and robustness are priorities in the `mypy` project. It sets a standard that even edge cases should be considered and handled thoughtfully. This can elevate the overall quality of contributions and reduce technical debt over time.
- **Documentation of Intent:** By explicitly handling this potential issue, the code becomes more self-documenting. Future developers will see that care was taken to address possible errors, which can guide them in making similarly robust decisions elsewhere in the code.

4. Minimal Cost for Long-Term Benefits

- **Low Overhead:** The proposed fix requires minimal changes to the codebase—essentially adding a few lines to check and handle the `num_indent` parameter. The cost of implementing this change is low, especially compared to the potential headaches and debugging time saved in the future if this issue were to arise unexpectedly.
- **Preventative Measure:** While the issue might not seem pressing right now, fixing it is a preventative measure that can save significant time and effort down the road. The best time to fix a potential bug is before it manifests in the wild.

Conclusion

Even though the `num_indent` parameter is currently set to zero by default and not explicitly used elsewhere, addressing this issue is a prudent step. It future-proofs the codebase, adheres to defensive programming principles, encourages best practices, and does so with minimal cost. This proactive approach ensures that the `mypy` project remains robust and reliable, ready to handle any future developments or integrations without unexpected failures.

FIGURE 7.26: Excerpt of GPT-4o conversation: Pro-Fix Argument, continued

As the maintainers of the `mypy` project, here's the strongest argument against fixing this issue:

1. YAGNI (You Aren't Gonna Need It)

- **Current Usage:** The `num_indent` parameter is currently always set to its default value of `0`, and nowhere in the codebase do we pass a non-zero value. Introducing safeguards for edge cases that are not currently relevant would go against the YAGNI principle. Fixing potential problems that don't exist adds unnecessary complexity and distraction to a function that works perfectly in its current use case.
- **Avoid Premature Optimization:** This issue isn't causing any real-world problems at present. Premature optimization—fixing something that has no immediate value or benefit—can lead to unnecessary code changes, making maintenance more difficult for a hypothetical situation that may never materialize.

2. Unlikely Edge Cases

- **Extremely Rare Scenario:** Even if a developer were to use `num_indent` in the future, the likelihood of someone passing a value large enough to trigger an `OverflowError` or `MemoryError` is extraordinarily low. In the rare event that this does happen, handling such an edge case might not justify the complexity or extra checks.
- **Scope of Practical Use:** In practical terms, it is highly improbable that anyone would set an indentation value (`num_indent`) large enough to cause an overflow or memory error. It would require values far exceeding what would be sensible or practical in the context of wrapping lines of error messages. There are no foreseeable real-world scenarios where such extreme values would be meaningful or desirable in `mypy`.

3. Simplicity of Codebase

- **Maintain Code Simplicity:** Adding input validation and error handling for an extremely unlikely case introduces additional complexity to the codebase. Simple, readable, and maintainable code is a priority for the `mypy` project, and we should avoid cluttering the code with unnecessary checks that don't improve the practical usability of the function.
- **Minimal Risk in the Real World:** Given that the function's current behavior works fine under reasonable usage scenarios, fixing a potential `OverflowError` or `MemoryError` is over-engineering. The risk of a catastrophic failure is minimal, especially given that `mypy` is primarily used in environments where such large integer inputs would not typically occur.

FIGURE 7.27: Excerpt of GPT-4o conversation: Anti-Fix Argument

4. Other Priorities

- **Higher Priority Issues:** There are more pressing issues, feature requests, and optimizations that require attention. Our resources are better spent improving performance, adding new features, fixing real bugs, and enhancing the user experience. Diverting attention to fixing theoretical problems detracts from our ability to focus on these higher-priority tasks.
- **Focus on Real Bugs:** We should focus on actual bugs reported by users or issues that significantly impact functionality. Right now, this issue is hypothetical, and we have no evidence to suggest that it's impacting users or causing crashes in real-world scenarios.

5. The Cost of Unnecessary Validation

- **Increased Code Overhead:** Adding checks for `num_indent` could introduce unnecessary overhead in terms of performance, however slight, and create a pattern where we over-validate inputs in every part of the code. We want to avoid making changes that impose additional performance costs for edge cases that will likely never occur.
- **Technical Debt from Over-Engineering:** Over-engineering by adding redundant checks could increase the technical debt of the project. It forces future contributors to consider these checks, maintain them, and ensure they are working correctly, despite being unnecessary for almost all practical cases.

Conclusion

Given that this issue currently poses no practical threat to the stability or performance of `mypy`, and the extremely low likelihood of it becoming a real problem, fixing it would be an unnecessary use of resources. The risk is minimal, and the potential downsides of adding complexity outweigh the speculative benefits. We should focus on more pressing improvements that provide real value to the project and its users.

FIGURE 7.28: Excerpt of GPT-4o conversation: Anti-Fix Argument, continued

Which is Stronger?

In this specific case, **the argument against fixing the issue** may be slightly stronger because of the practical and realistic approach it takes. If there is no evidence of this issue affecting users, and fixing it could be considered premature optimization, it makes sense to focus on higher-priority tasks that provide immediate value.

However, if `mypy` maintains a policy of strong defensive programming or if the maintainers want to future-proof the code for potential future integrations, the argument for fixing the issue becomes compelling due to its minimal cost and alignment with best practices.

Ultimately, **the decision hinges on the maintainers' philosophy**: if they prioritize simplicity and immediate needs (YAGNI), they would side with not fixing the issue. If they prioritize robustness and future-proofing, they would favor fixing it.

FIGURE 7.29: Excerpt of GPT-4o conversation: GPT's Opinion

8 Discussion and Future Work

In this chapter, we discuss the results and conclusions that we draw from them and address threats to validity. We also contribute proposals for enhancements to the Python type annotation system based on our observations of how Python programmers make up for deficiencies in the type system. Finally, we conclude with an overview of the areas of future work that can be explored.

8.1 Discussion

Overall, our evaluation shows that the tool has promise as a useful complement to static analysis and other software testing methodologies for Python, and also that more work must be done.

In Section 7.3.1, we uncovered clear evidence that the custom encoding we designed and implemented to support fuzzing is functional and effective at facilitating fuzzing. We therefore recommend the exclusive use of this encoding in future use of the tool. There are some ways the encoding could be improved, however, such as by compressing the metadata section so that encoded strings take up less memory.

In Section 7.3.2, we examined and compared various configurations of the tool. Our initial evaluation inspired us to perform an auxiliary evaluation on a single repository to provide a better view into the effect of the `acl2s_reachout_freq` parameter on code coverage and crash discovery. Our findings do not conclusively point to any one configuration, but they suggest weakly that a corpus with $|C| = 100$ is the best overall of the corpus sizes we tested, and higher `acl2s_reachout_freq` reaches the knee quickly with better coverage (according to CAK/TTK metrics), but lower `acl2s_reachout_freq` is better for triggering crashes. This is rather counterintuitive, as there is a correlation between code coverage and crashes triggered, although it may be weak [70]. Furthermore, based on the data, paying attention only to the "first knee" with CAK/TTK may be ignoring important parts of the story. Figure 7.15, for example, exhibits multiple instances where the knee occurs early, but then the overall coverage gained changes substantially later in the fuzzing campaign, and the ranking of configurations based on Coverage At Knee may be different from the ranking based on coverage at the end of the campaign. `mypy_100_0.2` and `mypy_acl2s` exhibit this in the right half of Figure 7.15.

All of this points to the need for longer timeouts, as encouraged by Klees et al. [70], so that the state space can be more fully explored and a true knee can be reached in the allotted time. Because of these flaws, we encourage interpretation of the results for RQ2 with care–we have attempted to answer the research question in this context, with the time and resources that were available, but unanswered questions remain. Additionally, this iteration of the TYPE HINT FUZZING implementation has not been extensively optimized, and differences in speed between independent trials and tool configurations may lead to the actual abilities of each configuration not being represented. The short timeout per function candidate (440 seconds), small sample size for independent trials ($N = 5$), inefficiencies in the use of time by various parts of the algorithm, and other startup costs can all greatly affect performance metrics. Some ideas for improvement in a future evaluation include 1) profiling the implementation to remove obvious inefficiencies, 2) implementing corpus minimization so that a large corpus actually contains diverse inputs, 3) comparing across both timeout, and number of runs, and 4) exploring other evaluation metrics, such *average new coverage obtained per*

ACL2s input and *average new coverage obtained per mutated input* to understand the relative contribution of ACL2s inputs and mutated inputs to coverage obtained. Similar metrics can be defined for crashes triggered.

In Section 7.3.3, we presented the results for RQ3: issues in code that were uncovered by the tool during the evaluation of RQ2 and were categorized and reported. At the time of this writing, we have obtained responses for six of them, with three positive and three negative. Two of the issues have been patched. Many of the issues reported were in *manticore*, which is a repository without active maintenance. This was an oversight when selecting repositories on which to evaluate that we will correct in future work. According to our classification methodology, we found no reportable bugs in *django*, which was the only repository not in the "regular" category of type annotation practice for which we analyzed issues [28]. There were, however, a large amount of exceptions related to "Any" type annotations in *django* (AE category; see Section 7.2.3). This suggests that the tool may only be effective at finding legitimate issues in code if it is well covered with type annotations. This motivates future work in exploring integration with type inference tools, so that the tool can be effective with even less manual effort. We also found that due to the nature of bottom-up fuzzing, there is significant manual effort involved in identifying whether or not bugs actually would manifest at the top-level of a program in order to generate reproduction steps that would be actionable for developers. Providing automation, or perhaps AI augmentation, of this process is another exciting area of future work.

Another central theme from the results of RQ3 is the debate about what exceptions are worth handling in Python. The customer-focused software engineer and the security analyst have differing views on this matter. In theory, any unhandled exception can be construed as a denial-of-service vulnerability. In C/C++, overflow errors can potentially even be exploitable security vulnerabilies [29], although it is unclear whether this is possible in Python. Adding this level of defensiveness to code can be a difficult case to make with code maintainers who are also balancing other competing priorities, such as code readability and more prevalent semantic bugs that are affecting real users. Ultimately, what is considered a bug heavily depends on the problem domain the software system is operating in and the maintainers' preferences and coding styles. The generative AI case study we performed provides early results that suggest that AI augmentation could be effective for making these determinations on a case-by-case basis; the conversation with GPT-4o [90] demonstrates that it has understanding of the key arguments for and against fixing bugs and it can come up with reasonable recommendations.

8.2 Threats To Validity

There are several threats to validity of the results presented herein that we would like to acknowledge here.

8.2.1 Sample Sizes

In a departure from the recommendations of previous work [70, 59, 3], we use a relatively small trial count ($N = 5$) for all experimental configurations in the results presented. This is primarily due to resource and time constraints, and we hope to improve on this limitation of our results in future work.

8.2.2 Timeouts

As noted in Klees et al. [70] and Herrera et al. [59], fuzzing performance can vary wildly between trials, and between various settings for timeouts. Klees et al. [70] recommend a timeout of at least 24 hours. Due to the time constraints under which we were operating, however, the minimum timeout with which we ran fuzzing campaigns was 440 seconds. This is per function candidate though, so mypy, for instance, received $440 * 127 = 55,880 secs = 15.5 hrs$ of fuzzing time in RQ2. The threat to validity of the results of RQ1 and RQ2 is that the relative performances of the configurations may differ significantly when run for longer timeouts. For RQ3, the quantity and quality of issues discovered may have been different under longer timeouts. This is a deficiency in our evaluation we hope to correct in future work. However, we contend that our evaluation of CAK/TTK remains valid and valuable, because by focusing on knees, we control for time.

8.2.3 Algorithmic Inefficiencies

We have not extensively profiled the tool, and we also did not implement the fuzzing best practice of corpus minimization due to time constraints [58]. It is therefore unclear how much of the time budget is spent on startup costs, other non-fuzzing activity, or redundant fuzzing activity related to an unminimized corpus. These are clear areas for improvement, and they may compromise the validity of the results in RQ2, especially given the use of short timeouts. We plan to improve on this in future work.

8.2.4 Repository Selection

Recall that we selected the original set of 15 repositories primarily by using the set of "regular" type annotators in Di Grazia and Pradel [28], with two more repositories added in because of popularity in the Python ecosystem. Selection in this way may have introduced bias towards repositories on which our tool might perform better than on a randomly selected open source Python repository. Importantly, however, we acknowledge that, given that the tool depends on type annotations to function, we only expect these results to generalize to other repositories with similar density and quality of type annotations.

8.2.5 Exception Deduplication

We knowingly depart from the guidance of Klees et al. [70] in using stack-based deduplication in the analysis of the crashes produced by the tool. We do this because ground truth is not readily available in our case, and accoding to Klees et al. [70], it tends to be the better of the two methods they examined, in that it tends to overcount less. Our choices for the maximum number of stack frames to compare, $N = 3$, is arbitrary, and may have benefited from a sensitivity analysis to determine how the choice of N would affect crash counts, and consequently, the results presented. We would like to explore this further in future work.

8.2.6 Assumptions in Statistical Analysis

In Section 7.3.2, we note that we make the assumption that the sample of repositories is representative of the general population, and therefore that results pooled across repositories on a per-function candidate level can all be construed as independent and identically distributed. This I.I.D. assumption also ignores any potential interactions between functions in the codebase, where one function candidate might call another function candidate that has been separately fuzzed, which is an interaction that we have observed in the studied repositories. For the purposes of the analyses we perform, this is convenient, and it at least allows for relative comparison of the configurations over the repositories and function candidates that were studied, but in practice, these assumptions may not hold, and deserve more careful scrutiny in future work (though, it is worth noting that there is always a risk that results based on sampling will not generalize).

8.2.7 Confidence Intervals

All of the confidence intervals were constructed with bootstrapping, which is a known and acceptable method in statistics for estimating the distribution of a test statistic with randomness [46, 3]. An example of this method in use can be seen in Herrera et al. [59]. However, since these confidence intervals are randomly generated, different pseudorandom states may result in different confidence intervals, leading to different interpretations of significance of the results. This criticism of the bootstrap and others are given in Gleser [42]. Nonetheless, this method proves useful in situations where robustness against violations of underlying assumptions of statistical tests is needed, and the distribution of the statistic is unknown. Constructing confidence intervals with another method may yield different results, so all claims about statistical significance should be interpreted with care. We encourage the reader to focus primarily on the bugs found and reported as the primary measure of our tool's effectiveness.

8.2.8 Custom Enumerators

The construction of the custom enumerators given in Chapter 3 was based on simple heuristics. There may be opportunities for improvement based on previous work on common input validation mistakes in Python code.

8.2.9 Manual Bug Analysis and Reporting

The manual bug analysis was conducted entirely by the author of this thesis, which means that there may be bias or inaccuracies in that analysis. In future work, a review process where each bug is reviewed by at least two independent individuals should be used, following previous work [116, 104, 6].

8.3 Type Hint Enhancement Proposals

Throughout this work, we observed various instances where the type annotations specified in PEP-484 [108] seem to be deficient for Python developers' needs. Recall, for instance, Listing 2.4. Although a main advantage of the PEP-484 system is that it is simple, it lacks the expressivity that Python developers clearly indicate that they are demanding. The purpose of static analysis is defeated when runtime checks (i.e. `assert` statements) need to be inserted in order to make up for the deficiencies of the type system. We therefore propose the following enhancements to the Python type hint system to allow for commonly required subtypes of common types. These may or may not be feasible to implement in the current suite of tools–we offer them more as food for thought.

8.3.1 Non-empty List

This is the requirement that was not met in Listing 2.4. We propose the addition of a `NonEmptyList` type to the `typing` module which represents this type.

8.3.2 Natural Numbers/Non-negative Integers

In many instances, an integer value is being passed around, but certain integer values do not make sense for that specific context. We propose the addition of a `Natural` type to the `typing` module which represents the natural numbers (in the context of computer science, this set includes 0).

8.3.3 Generalization: Predicates

Given that the syntax of annotations in Python is actually unspecified in the grammar [142], it is possible to arbitrarily extend the language of type hints. We therefore propose the addition of predicates as a way to add arbitrary constraints to types. These would enable the previous proposals and much more, but this may be very challenging from an implementation standpoint. The set of predicates might have to be limited to a small language at first, and integration with an automated theorem prover such as ACL2s may be necessary. Interestingly, this is not a new idea in the Python ecosystem. As an example, *icontract* [97] is a Python library that supports a "design-by-contract" coding style, allowing users to specify arbitrary function contracts that will produce descriptive and useful failure messages when violated. As for how these contracts could be expressed in type annotations, we propose the syntax shown in Listing 8.1.

```
def myfunction(a: List[Integer[_ > 5]][len(_) > 0])
    -> Integer[_ > 5]:
    """
    sum all integers in the given nonempty list
    containing integers only greater than 5
    """
    return sum(a)
```

LISTING 8.1: StripMetadata definition in `FuzzerMutate.h`

The underscore character ('_') in each predicate is a placeholder for any value of the base type. In the example of `myfunction` given in the listing, it may not be trivial to a static analysis tool that the output of this function will always be an integer greater than five, provided that the input contract is satisfied. This is where theorem provers or SMT solvers may be able to help. A simple sublanguage that would cover the common use cases elucidated here would simply be the `len` function to represent the sizes of collections, and the equality and inequality comparison operators (==, !=, <, >, etc.). There is already official language support for arbitrary annotations which extend the PEP-484 type system as well [132], so perhaps this system could be built on top of what is already existent in the language.

8.4 Future Work

This thesis represents the first foray into a broad area of potential research. With this tool, we have pioneered the ability to apply fuzzing to Python code that is type annotated in an automated way. However, there are many areas we would like to improve the tool both in terms of core functionality, and in terms of future work, we would like to distinguish between short-term directions and long-term directions.

8.4.1 Short-term Directions

Basic enhancements to the feature set of the currently implemented tool include:

- Check input contracts for functions candidates that are being called during the fuzzing of another function candidate
- Expand set of supported types to include more types in Python's `typing` module, and improve support for arbitrary user-defined classes
- Reduce the space footprint of the custom encoding's encoded strings
- Implement corpus minimization, which is fuzzing best practice [58]
- Correct import and dependency resolution errors during repository setup to enable the extraction of more function candidates
- Improve fuzzing process isolation, perhaps by using Docker or a similar technology
- Add more representative integers to the custom enumerator for integers

We would also like to strengthen the evaluation of the tool in the following ways:

- Evaluate a more comprehensive set of tool configurations for RQ2, after optimizations have been made to the implementation
- Run a trial of the tool on the codebase of a software company in industry, to assess real-world performance and economic value

8.4.2 Long-term Directions

Some long-term directions we envision for this work include:

- AI-enabled explanations of issues found in code
- "Type repair:" suggesting more general or more specific type annotations based on fuzzing results
- Integration with other static analysis methods, such as control-flow analysis and type inference

9 Conclusion

In this thesis, we proposed, designed, and implemented TYPE HINT FUZZING, a novel approach to testing Python code in a dynamic way. We intend for this approach to complement existing Python code analysis offerings, such as static type checkers, and we have aimed to make the design modular and extensible to enable future development of this tool. We have validated the construction of our custom encoding (Section 7.3.1), explored various configurations of the tool (Section 7.3.2), and analyzed the tool's findings (Section 7.3.3). Reporting the findings of the tool has turned out to be successful, especially in inviting the community into a productive and insightful conversation with this work.

Frankly, despite this treatment exceeding one hundred pages in length, it has barely scratched the surface of what this approach is capable of. From expanding the amount of type checking the tool is doing during fuzzing, to adding more intelligence when filtering out crash reports, to expanding the tool's capabilities as a property-based testing engine, there are so many more avenues for evaluation, implementation, and feature additions that we could not fit in this thesis due to constraints of time and resources. We hope academia and industry both agree with this sentiment, and we are excited to see what comes of this work.

A Tool Configuration Reference

Here is the full reference for configuration options of the tool, at the time of this writing.

- `acl2s_reachout_frequency`: When `backend` is set to `atheris`, this controls the frequency at which a new input from ACL2s will be pulled during the fuzzing campaign, instead of using a mutated seed input from Atheris. 0.7, for example, means that 70% of fuzzing iterations will pull a fresh example from ACL2s.
- `backend`: This is the fuzzing backend to use. Can be either `acl2s` or `atheris`. The `acl2s` backend represents ACL2s-only fuzzing, with no coverage-guided mutation. `atheris` represents fuzzing with Atheris. More backends may be added in the future. `corpus_amendment_timeout_multiple`: The number of seconds each corpus element is allotted when it is passed into the FUT during corpus amendment, before the execution is interrupted. This is to prevent infinite loops or other slow execution from halting fuzzing progress indefinitely.
- `corpus_size`: The size of the corpus. This is applicable when `backend` is set to `atheris`.
- `encoding`: The encoding to use, either "custom_v2" or "pickle".
- `excluded_functions`: The list of patterns to exclude when collecting the list of functions to fuzz. Currently, this setting does not support wildcard characters or regex-like syntax.
- `fuzzing_unit_max_len`: The maximum length, in bytes, of the fuzzing input buffer in Atheris/libFuzzer.
- `included_functions`: The list of patterns to select when collecting the list of functions to fuzz. When this is set, the function name is first tested to check if it contains at least one pattern from `included_functions`, then `excluded_functions` is applied as described in its entry.
- `metadata_only`: If set to True, scrambles the data portion of all seed inputs with random bytes, to remove the influence of primitive type information.
- `memory_limit_mb`: The memory limit, in megabytes.
- `random_seed`: The random seed of the fuzzing campaign.

- `should_top_off_corpus`: When this is set to true, the corpus is "topped off" to its originally specified size (in `corpus_size`) after elements are filtered out during corpus amendment. This supercedes `should_top_off_corpus_at_empty`.
- `should_top_off_corpus_at_empty`: When this is set to true, when the corpus becomes empty during corpus amendment, exactly one element is added to the corpus to make it non-empty.
- `style`: Controls the "style" of the fuzzing run: whether it is limited by time, or by number of iterations. Takes two values, `TIMEOUT` and `RUN_LIMIT`.
- `timeout`: Timeout, in seconds, for each function in the fuzzing campaign.

B Full Crash Data

Definitions of the two-letter category abbreviations are given in Table 7.1. Exception sites are given in the form `ExceptionType@path/to/file:LineNo`. Exception sites whose paths are prefixed with `<python_internal>` occur within the Python standard library. Other exception sites are given relative to the root of the repository in which they occur.

TABLE B.1: Comprehensive Exception Data

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mypy	OverflowError@mypy/mypy/util.py:527			✓					https://github.com/python/mypy/issues/17454	mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_10000_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	MemoryError@mypy/mypy/util.py:527			✓					https://github.com/python/mypy/issues/17454	mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_10000_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	OverflowError@mypy/mypy/strconv.py:639			✓					https://github.com/python/mypy/issues/17454	mypy_1_0.1, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	MemoryError@mypy/mypy/strconv.py:639			✓					https://github.com/python/mypy/issues/17454	mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	AssertionError@mypy/mypy/modulefinder.py:850	✓		✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	UnicodeDecodeError@mypy/mypy/util.py:168			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	KeyError@mypy/mypy/stubinfo.py:13			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	ValueError@mypy/mypy/inspections.py:622			✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	OverflowError@mypy/mypy/dmypy_os.py:31			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	KeyError@mypy/mypy/report.py:128			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	ValueError@mypy/mypy/fscache.py:71			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	ValueError@mypy/mypy/fscache.py:71			✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	ValueError@mypy/mypy/fscache.py:71			✓						mypy_1_0
mypy	KeyError@mypy/mypy/config_parser.py:389			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	OverflowError@mypy/mypy/util.py:427			✓						mypy_1_0.1, mypy_1_0
mypy	OverflowError@mypy/mypy/util.py:844			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	KeyError@mypy/mypy/typeanal.py:173			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	ValueError@mypy/mypy/modulefinder.py:867	✓		✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mypy	ValueError@mypy/mypy/modulefinder.py:868	✓		✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	AssertionError@mypy/mypy/server/target.py:5			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	UnicodeDecodeError@mypy/mypy/util.py:547	✓		✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	TypeError@mypy/mypy/stubtest.py:76			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	ValueError@unknown			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	OverflowError@mypy/mypy/report.py:581			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_1_0, mypy_10000_0.1
mypy	AssertionError@mypy/mypy/messages.py:2997			✓						mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1
mypy	ValueError@mypy/mypy/fscache.py:172			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	TypeError@<python_internal>/lib/python3.8/re.py:210									mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1
mypy	ValueError@mypy/mypy/inspections.py:626			✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_1_0
mypy	ValueError@mypy/mypy/messages.py:3040	✓	✓	✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1
mypy	ValueError@mypy/mypy/messages.py:3034	✓	✓	✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	ZeroDivisionError@mypy/mypy/messages.py:3070		✓	✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1
mypy	AssertionError@mypy/mypy/constant_fold.py:151			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	AssertionError@mypy/mypy/modulefinder.py:850	✓		✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:123				✓	✓				mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	ValueError@<python_internal>/lib/python3.8/posixpath.py:259			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	IndexError@mypy/mypy/server/target.py:5			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mypy	IndexError@mypy/mypy/server/target.py:9			✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	error@mypy/mypy/util.py:548	✓	✓	✓						mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_acl2s, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1, mypy_1_0
mypy	ValueError@mypy/mypy/util.py:548			✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_1_0
mypy	ValueError@<python_internal>/lib/python3.8/json/encoder.py:257	✓	✓	✓						mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_100_0.2, mypy_10000_0.2, mypy_10000_0.1
mypy	PermissionError@mypy/mypy/fscache.py:172	✓	✓	✓						mypy_1_0.1, mypy_10000_0, mypy_acl2s, mypy_1_0.2, mypy_10000_0.2, mypy_10000_0.1
mypy	MemoryError@mypy/mypy/constant_fold.py:138				✓	✓			https://github.com/python/mypy/issues/17008	mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_10000_0.2, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:138				✓	✓			https://github.com/python/mypy/issues/17008	mypy_1_0.1, mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_1_0.2, mypy_100_0.2, mypy_10000_0.1, mypy_1_0
mypy	ValueError@<python_internal>/lib/python3.8/json/encoder.py:424	✓	✓	✓						mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_10000_0.2, mypy_10000_0.1
mypy	ValueError@mypy/mypy/build.py:537			✓						mypy_1_0.1, mypy_acl2s, mypy_100_0, mypy_1_0.2, mypy_100_0.1, mypy_100_0.2
mypy	OverflowError@mypy/mypy/constant_fold.py:153				✓	✓			https://github.com/python/mypy/issues/17008	mypy_1_0.1, mypy_100_0, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:163				✓	✓			https://github.com/python/mypy/issues/17008	mypy_100_0, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:160				✓	✓			https://github.com/python/mypy/issues/17008	mypy_100_0, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:155				✓	✓			https://github.com/python/mypy/issues/17008	mypy_100_0, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:157				✓	✓			https://github.com/python/mypy/issues/17008	mypy_100_0, mypy_1_0
mypy	OverflowError@mypy/mypy/constant_fold.py:166				✓	✓			https://github.com/python/mypy/issues/17008	mypy_100_0, mypy_1_0
mypy	ValueError@<python_internal>/lib/python3.8/posixpath.py:259			✓						mypy_1_0.1, mypy_acl2s, mypy_1_0.2, mypy_100_0.2, mypy_1_0
mypy	ValueError@unknown			✓						mypy_1_0.1, mypy_acl2s, mypy_1_0.2, mypy_100_0.2, mypy_1_0
mypy	ValueError@<python_internal>/lib/python3.8/posixpath.py:259			✓						mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_10000_0.2, mypy_10000_0.1
mypy	ValueError@unknown			✓						mypy_10000_0, mypy_100_0, mypy_100_0.1, mypy_10000_0.2, mypy_10000_0.1
mypy	TypeError@<python_internal>/lib/python3.8/re.py:210									mypy_10000_0, mypy_10000_0.1, mypy_10000_0.2, mypy_100_0.1
mypy	ValueError@<python_internal>/lib/python3.8/json/encoder.py:424	✓	✓	✓						mypy_10000_0, mypy_10000_0.1
mypy	ValueError@<python_internal>/lib/python3.8/json/encoder.py:257	✓	✓	✓						mypy_10000_0, mypy_10000_0.2
mypy	TypeError@<python_internal>/lib/python3.8/re.py:210									mypy_1_0.1, mypy_acl2s, mypy_100_0.2, mypy_1_0.2

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mindsdb	ValueError@mindsdb/mindsdb/interfaces/storage/fs.py:108			✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/api/http/utils.py:36						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/surrealdb_handler/utils/surreal_get_info.py:10						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	ValueError@mindsdb/mindsdb/integrations/utilities/sql_utils.py:115						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/utilities/sql_utils.py:115						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/byom_handler/proc_wrapper.py:77						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_1_0, mindsdb_1_0.1, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/api/mysql/mysql_proxy/data_types/mysql_packet.py:148					✓			https://github.com/mindsdb/mindsdb/issues/9426	mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/interfaces/database/projects.py:299	✓								mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/lightdash_handler/lightdash_tables.py:20						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/api/executor/datahub/datanodes/information_schema_datanode.py:346						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/genericpath.py:19						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	PermissionError@checksumdir/checksumdir/__init__.py:89	✓		✓						mindsdb_1_0.2, mindsdb_1_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/eventstoredb_handler/utils/helpers.py:35						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/lightdash_handler/lightdash_tables.py:15						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/eventbrite_handler/eventbrite_tables.py:14						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/utilities/hooks/profiling.py:18						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/web_handler/urlcrawl_helpers.py:163						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/monetdb_handler/utils/monet_get_id.py:5						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	LocationParseError@<string>:3			✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:210						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/migrations/migrate.py:29	✓		✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/utilities/sql_utils.py:66						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/utilities/ps.py:54						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:231						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/utilities/security.py:17			✓						mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_1_0, mindsdb_1_0.1
mindsdb	UnicodeError@mindsdb/mindsdb/utilities/security.py:17			✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	gaierror@mindsdb/mindsdb/utilities/security.py:17			✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/monetdb_handler/utils/monet_get_id.py:5						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/pathlib.py:667						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	OSError@<python_internal>/lib/python3.8/pathlib.py:1198	✓	✓	✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	KeyError@mindsdb/mindsdb/utilities/profiler/profiler.py:19						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mindsdb	AttributeError@mindsdb/mindsdb/api/mysql/mysql_proxy/classes/com_operators.py:20						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/eventstoredb_handler/utils/helpers.py:45						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/surrealdb_handler/utils/surreal_get_info.py:3						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/utilities/config.py:21						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/utilities/ml_task_queue/utils.py:50	✓								mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	OverflowError@mindsdb/mindsdb/utilities/ml_task_queue/utils.py:47								https://github.com/mindsdb/mindsdb/issues/9467	mindsdb_100_0, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	AttributeError@mindsdb/mindsdb/integrations/libs/handler_helpers.py:25						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@<python_internal>/lib/python3.8/urllib/parse.py:119						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/lightdash_handler/lightdash_tables.py:32						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	KeyError@mindsdb/mindsdb/integrations/handlers/rocket_chat_handler/rocket_chat_tables.py:11						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/libs/llm_utils.py:161						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/eventstoredb_handler/utils/helpers.py:31						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/interfaces/storage/json.py:53									mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/site-packages/requests/models.py:173						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	ValueError@<python_internal>/lib/python3.8/site-packages/requests/models.py:173						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mindsdb	ValueError@<python_internal>/lib/python3.8/site-packages/requests/utils.py:1038	✓	✓	✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/web_handler/urlcrawl_helpers.py:261						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/npm_handler/npm_tables.py:15						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/web_handler/urlcrawl_helpers.py:19						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/byom_handler/proc_wrapper.py:54									mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/utilities/datasets/dataset.py:66						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/openbb_handler/openbb_tables.py:25						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	OSError@mindsdb/mindsdb/interfaces/storage/fs.py:108	✓	✓	✓						mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/surrealdb_handler/utils/surreal_get_info.py:11						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/lightdash_handler/lightdash_tables.py:22						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	TypeError@mindsdb/mindsdb/interfaces/storage/fs.py:59						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/genericpath.py:42						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	OverflowError@<python_internal>/lib/python3.8/genericpath.py:19						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	OverflowError@<python_internal>/lib/python3.8/genericpath.py:42						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	TypeError@mindsdb/mindsdb/integrations/handlers/lightdash_handler/lightdash_tables.py:16						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	ValueError@mindsdb/mindsdb/integrations/handlers/eventstoredb_handler/utils/helpers.py:39	✓	✓	✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:191						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mindsdb	ValueError@mindsdb/mindsdb/integrations/utilities/utils.py:11						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/handlers/lightdash_handler/lightdash_tables.py:33						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	ValueError@<python_internal>/lib/python3.8/json/encoder.py:257	✓	✓	✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_10000_0
mindsdb	AttributeError@mindsdb/mindsdb/integrations/utilities/datasets/dataset.py:66						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	OSError@mindsdb/mindsdb/interfaces/storage/fs.py:110	✓		✓						mindsdb_acl2s, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@mindsdb/mindsdb/api/executor/utilities/sql.py:130						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_10000_0
mindsdb	KeyError@mindsdb_sql/mindsdb_sql/render/sqlalchemy_render.py:159						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_acl2s, mindsdb_10000_0
mindsdb	ValueError@werkzeug/src/werkzeug/_internal.py:140	✓	✓	✓						mindsdb_10000_0.1, mindsdb_10000_0
mindsdb	PermissionError@mindsdb/mindsdb/interfaces/storage/fs.py:108	✓		✓						mindsdb_1_0.1, mindsdb_acl2s, mindsdb_10000_0.1
mindsdb	PermissionError@mindsdb/mindsdb/interfaces/storage/fs.py:108	✓		✓						mindsdb_10000_0.2, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	PermissionError@mindsdb/mindsdb/interfaces/storage/fs.py:108	✓		✓						mindsdb_1_0.1, mindsdb_100_0.1, mindsdb_1_0.2, mindsdb_1_0
mindsdb	OverflowError@<python_internal>/lib/python3.8/genericpath.py:19	✓		✓						mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	OSError@<python_internal>/lib/python3.8/pathlib.py:1198	✓		✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/genericpath.py:19						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	TypeError@<python_internal>/lib/python3.8/genericpath.py:42						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	OverflowError@<python_internal>/lib/python3.8/genericpath.py:42	✓		✓						mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	TypeError@<python_internal>/lib/python3.8/pathlib.py:667						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:191						✓			mindsdb_acl2s, mindsdb_100_0, mindsdb_100_0.1, mindsdb_100_0.2
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:210						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:231						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	AttributeError@<python_internal>/lib/python3.8/urllib/parse.py:119						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/site-packages/requests/models.py:173						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	ValueError@<python_internal>/lib/python3.8/site-packages/requests/models.py:173						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	ValueError@<python_internal>/lib/python3.8/site-packages/requests/utils.py:1050	✓	✓	✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_1_0, mindsdb_1_0.1, mindsdb_10000_0
mindsdb	ValueError@<python_internal>/lib/python3.8/json/encoder.py:257	✓	✓	✓						mindsdb_10000_0
mindsdb	UnicodeEncodeError@mindsdb/mindsdb/utilities/cache.py:82			✓						mindsdb_1_0.2, mindsdb_100_0.2
mindsdb	OverflowError@<python_internal>/lib/python3.8/genericpath.py:19	✓		✓						mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	OSError@<python_internal>/lib/python3.8/pathlib.py:1198	✓		✓						mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/genericpath.py:42						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	TypeError@<python_internal>/lib/python3.8/genericpath.py:19						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	OverflowError@<python_internal>/lib/python3.8/genericpath.py:42						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
mindsdb	TypeError@<python_internal>/lib/python3.8/pathlib.py:667						✓			mindsdb_10000_0.2, mindsdb_10000_0.1, mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2, mindsdb_10000_0
mindsdb	TypeError@<python_internal>/lib/python3.8/re.py:191						✓			mindsdb_100_0, mindsdb_acl2s, mindsdb_100_0.1, mindsdb_100_0.2, mindsdb_1_0.1, mindsdb_1_0.2
django	NameError@django/django/contrib/admindocs/utils.py:124	✓								django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	TypeError@<python_internal>/lib/python3.8/re.py:277						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:173						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:36						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/db/migrations/migration.py:239						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/geom.py:20						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:27						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	TypeError@django/django/utils/deconstruct.py:59									django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:123						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:36						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	TypeError@django/django/utils/version.py:61						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:91						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:157						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:154						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:36						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:80						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:51						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:71						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
django	AttributeError@django/django/contrib/admin/decorators.py:67						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/admin/decorators.py:65						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/admin/decorators.py:71						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/geom.py:32						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:36						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:164						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/utils/jslex.py:28						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/admin/decorators.py:25						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/admin/decorators.py:23						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:98						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	TypeError@django/django/utils/version.py:61						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/utils/deconstruct.py:52						✓			django_10000_0.1, django_1_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AssertionError@django/django/utils/version.py:61						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0
django	AttributeError@django/django/contrib/gis/gdal/prototypes/generation.py:150						✓			django_10000_0.1, django_1_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	AssertionError@django/django/utils/regex_helper.py:350									django_10000_0.1, django_1_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
django	TypeError@<python_internal>/lib/python3.8/sre_parse.py:948						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0
django	OverflowError@<python_internal>/lib/python3.8/sre_compile.py:780						✓			django_1_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_100_0
django	AttributeError@django/django/contrib/admin/decorators.py:69						✓			django_10000_0.1, django_1_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	TypeError@django/django/utils/jslex.py:28						✓			django_10000_0.1, django_1_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0
django	AssertionError@django/django/utils/version.py:61						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0
django	AssertionError@django/django/utils/version.py:62						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0
django	AssertionError@django/django/utils/version.py:62						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_1_0, django_100_0.2, django_10000_0, django_100_0
django	OverflowError@<python_internal>/lib/python3.8/sre_compile.py:780						✓			django_1_0.1, django_100_0.1, django_acl2s, django_1_0, django_1_0.2, django_100_0.2, django_100_0
django	TypeError@<python_internal>/lib/python3.8/re.py:277						✓			django_1_0.1, django_10000_0.1, django_100_0.1, django_10000_0.2, django_acl2s, django_1_0.2, django_100_0.2, django_10000_0, django_100_0
django	TypeError@<python_internal>/lib/python3.8/sre_parse.py:948						✓			django_1_0.1, django_100_0.1, django_acl2s, django_1_0.2, django_100_0.2, django_100_0
black	AssertionError@black/src/black/strings.py:140			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	AssertionError@black/src/black/strings.py:137			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	AssertionError@black/src/black/strings.py:144			✓						black_acl2s, black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓	✓				https://github.com/psf/black/issues/4400	black_100_0, black_1_0.1, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2
black	IndentationError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@black/src/black/numerics.py:41			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	OverflowError@black/src/black/nodes.py:414			✓						black_100_0, black_1_0.1, black_100_0.1, black_1_0, black_100_0.2, black_1_0.2

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
black	OverflowError@black/src/black/nodes.py:413			✓						black_100_0, black_1_0.1, black_100_0.1, black_1_0, black_100_0.2, black_1_0.2
black	MemoryError@black/src/black/nodes.py:413			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@black/src/black/nodes.py:414			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@black/src/black/numerics.py:18			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@black/src/black/numerics.py:41			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	IndentationError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@black/src/black/handle_ipynb_magics.py:275			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/pathlib.py:452			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	OSError@<python_internal>/lib/python3.8/pathlib.py:1198	✓		✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@black/src/black/numerics.py:41			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓					https://github.com/psf/black/issues/4400	black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_1_0.2
black	UnicodeDecodeError@<python_internal>/lib/python3.8/codecs.py:322			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	JSONDecodeError@<python_internal>/lib/python3.8/json/decoder.py:353			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	AssertionError@black/src/black/strings.py:140			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	AssertionError@black/src/black/strings.py:137			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	TypeError@black/src/black/output.py:20			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	TypeError@black/src/black/output.py:20			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
black	TypeError@black/src/black/output.py:29			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	TypeError@black/src/black/output.py:29			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	IndexError@black/src/black/numerics.py:32			✓						black_100_0, black_acl2s, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	IndexError@black/src/black/strings.py:188			✓						black_100_0, black_acl2s, black_1_0, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	AssertionError@black/src/black/handle_ipynb_magics.py:171			✓						black_100_0, black_acl2s, black_1_0, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	AssertionError@black/src/black/strings.py:144			✓						black_100_0, black_acl2s, black_1_0, black_1_0.1, black_10000_0, black_100_0.1, black_10000_0.1, black_100_0.2, black_10000_0.2, black_1_0.2
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s, black_100_0, black_1_0.1, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓					https://github.com/psf/black/issues/4400	black_100_0, black_1_0, black_1_0.1, black_100_0.1, black_10000_0.1, black_1_0.2
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s, black_100_0, black_1_0.1, black_100_0.1, black_1_0, black_100_0.2
black	OSError@<python_internal>/lib/python3.8/tempfile.py:542			✓						black_acl2s
black	OSError@<python_internal>/lib/python3.8/tempfile.py:542			✓						black_acl2s
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_acl2s, black_100_0.1, black_100_0.2, black_1_0.2
black	IndentationError@<python_internal>/lib/python3.8/tokenize.py:512			✓						black_acl2s
black	UnicodeEncodeError@<python_internal>/lib/python3.8/tempfile.py:473			✓						black_100_0, black_10000_0
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓					https://github.com/psf/black/issues/4400	black_1_0, black_100_0.1, black_1_0.1
black	TypeError@black/src/black/output.py:53			✓						black_acl2s
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	IndentationError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓					https://github.com/psf/black/issues/4400	black_1_0, black_100_0.2, black_1_0.2

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
black	IndentationError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/pathlib.py:452			✓						black_100_0, black_1_0.1, black_10000_0, black_1_0, black_10000_0.2, black_1_0.2, black_10000_0.1
black	OSError@<python_internal>/lib/python3.8/pathlib.py:1198	✓		✓						black_100_0, black_1_0.1, black_10000_0, black_1_0, black_10000_0.2, black_1_0.2, black_10000_0.1
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	UnicodeDecodeError@<python_internal>/lib/python3.8/codecs.py:322			✓						black_1_0.1, black_10000_0, black_1_0, black_100_0.2, black_10000_0.2, black_10000_0.1
black	JSONDecodeError@<python_internal>/lib/python3.8/json/decoder.py:353			✓						black_100_0, black_1_0.1, black_10000_0, black_100_0.1, black_1_0, black_100_0.2, black_10000_0.2, black_1_0.2, black_10000_0.1
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓					https://github.com/psf/black/issues/4400	black_100_0.1
black	MemoryError@<python_internal>/lib/python3.8/ast.py:47			✓					https://github.com/psf/black/issues/4400	black_100_0.2
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	IndentationError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	SyntaxError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	IndentationError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	ValueError@<python_internal>/lib/python3.8/pathlib.py:452			✓						black_acl2s
black	OSError@<python_internal>/lib/python3.8/pathlib.py:1198	✓		✓						black_acl2s
black	OSError@<python_internal>/lib/python3.8/tempfile.py:558			✓						black_acl2s
black	OSError@<python_internal>/lib/python3.8/tempfile.py:558			✓						black_acl2s
black	ValueError@<python_internal>/lib/python3.8/ast.py:47			✓						black_acl2s
black	IndentationError@<python_internal>/lib/python3.8/tokenize.py:512			✓						black_acl2s
black	UnicodeDecodeError@<python_internal>/lib/python3.8/codecs.py:322			✓						black_acl2s
black	JSONDecodeError@<python_internal>/lib/python3.8/json/decoder.py:353			✓						black_acl2s
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:57						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:57						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:57						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:57						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	KeyError@manticore/manticore/utils/install_helper.py:14						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:172								https://github.com/trailofbits/manticore/issues/2660	manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	OverflowError@manticore/manticore/core/smtlib/operators.py:172								https://github.com/trailofbits/manticore/issues/2660	manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2
manticore	ValueError@manticore/manticore/core/smtlib/operators.py:172								https://github.com/trailofbits/manticore/issues/2660	manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	MemoryError@manticore/manticore/core/smtlib/operators.py:172								https://github.com/trailofbits/manticore/issues/2660	manticore_10000_0.1, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_10000_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:148						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/core/smtlib/expression.py:547								https://github.com/trailofbits/manticore/issues/2651	manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	OverflowError@manticore/manticore/core/smtlib/expression.py:547								https://github.com/trailofbits/manticore/issues/2651	manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	MemoryError@manticore/manticore/core/smtlib/expression.py:547								https://github.com/trailofbits/manticore/issues/2651	manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:120						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:139						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:201						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:68						✓			manticore_10000_0.1, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:68						✓			manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:68						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:68						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AttributeError@manticore/manticore/ethereum/cli.py:90						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AttributeError@manticore/manticore/wasm/cli.py:14						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AttributeError@<python_internal>/lib/python3.8/site-packages/atheris/function_hooks.py:372						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/visitors.py:1066						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:134						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:94						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AttributeError@manticore/manticore/utils/config.py:309						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	RecursionError@manticore/manticore/core/smtlib/operators.py:51								https://github.com/trailofbits/manticore/issues/2652	manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:157						✓			manticore_10000_0.1, manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_1_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:149						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/core/smtlib/operators.py:148								https://github.com/trailofbits/manticore/issues/2653	manticore_10000_0.1, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	OverflowError@manticore/manticore/core/smtlib/operators.py:148								https://github.com/trailofbits/manticore/issues/2653	manticore_1_0.1, manticore_100_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
manticore	MemoryError@manticore/manticore/core/smtlib/operators.py:148								https://github.com/trailofbits/manticore/issues/2653	manticore_1_0.1, manticore_100_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:118						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AttributeError@manticore/manticore/core/smtlib/operators.py:142						✓			manticore_10000_0.1, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_1_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:141						✓			manticore_10000_0.1, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/core/smtlib/operators.py:139								https://github.com/trailofbits/manticore/issues/2654	manticore_10000_0.1, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:202						✓			manticore_10000_0.1, manticore_acl2s, manticore_1_0.1, manticore_1_0.2, manticore_100_0, manticore_1_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:203						✓			manticore_10000_0.1, manticore_acl2s, manticore_1_0.2, manticore_1_0.1, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	RecursionError@manticore/manticore/core/smtlib/operators.py:62								https://github.com/trailofbits/manticore/issues/2652	manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/visitors.py:41						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/wasm/types.py:465	✓	✓	✓						manticore_10000_0.1, manticore_1_0.1, manticore_1_0.2, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/visitors.py:41						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/solver.py:103			✓			✓			manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:26						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/visitors.py:41						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/core/smtlib/operators.py:134								https://github.com/trailofbits/manticore/issues/2655	manticore_10000_0.1, manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:92						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:38						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:38						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_acl2s, manticore_100_0, manticore_1_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ZeroDivisionError@manticore/manticore/core/smtlib/operators.py:157								https://github.com/trailofbits/manticore/issues/2656	manticore_10000_0.1, manticore_acl2s, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:121						✓			manticore_10000_0.2, manticore_10000_0, manticore_acl2s, manticore_10000_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:119						✓			manticore_10000_0.1, manticore_acl2s, manticore_1_0.2, manticore_1_0.1, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/core/smtlib/operators.py:141								https://github.com/trailofbits/manticore/issues/2654	manticore_10000_0.1, manticore_1_0.2, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AssertionError@manticore/manticore/core/smtlib/operators.py:204						✓			manticore_10000_0.1, manticore_acl2s, manticore_1_0.1, manticore_1_0.2, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:95						✓			manticore_10000_0.2, manticore_10000_0, manticore_acl2s, manticore_10000_0.1
manticore	TypeError@manticore/manticore/core/smtlib/operators.py:93						✓			manticore_10000_0.1, manticore_acl2s, manticore_1_0.2, manticore_1_0.1, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@ply/ply/lex.py:253						✓			manticore_acl2s
manticore	AttributeError@manticore/manticore/core/smtlib/solver.py:93						✓			manticore_acl2s
manticore	ValueError@manticore/manticore/core/smtlib/solver.py:96						✓			manticore_acl2s
manticore	MemoryError@manticore/manticore/core/smtlib/operators.py:134								https://github.com/trailofbits/manticore/issues/2657	manticore_10000_0.1, manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_10000_0, manticore_10000_0.2, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	OverflowError@manticore/manticore/core/smtlib/operators.py:134								https://github.com/trailofbits/manticore/issues/2657	manticore_acl2s, manticore_1_0.1, manticore_100_0, manticore_1_0.2, manticore_100_0.2, manticore_100_0.1
manticore	MemoryError@manticore/manticore/core/smtlib/operators.py:139								https://github.com/trailofbits/manticore/issues/2658	manticore_10000_0.1, manticore_1_0.1, manticore_1_0.2, manticore_100_0, manticore_10000_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	TypeError@manticore/manticore/core/parser/parser.py:206								https://github.com/trailofbits/manticore/issues/2659	manticore_1_0, manticore_1_0.2, manticore_1_0.1
manticore	OverflowError@manticore/manticore/core/smtlib/operators.py:139								https://github.com/trailofbits/manticore/issues/2658	manticore_1_0.2, manticore_100_0.2, manticore_100_0, manticore_100_0.1

Continued on next page

– continued from previous page

Repository	Exception Site	EX	FO	EE	ER	LB	AE	TE	Report Link	Experiments
manticore	MemoryError@manticore/manticore/core/smtlib/operators.py:141								https://github.com/trailofbits/manticore/issues/2658	manticore_1_0.2, manticore_100_0.2, manticore_100_0, manticore_100_0.1
manticore	OverflowError@manticore/manticore/core/smtlib/operators.py:141								https://github.com/trailofbits/manticore/issues/2658	manticore_1_0.2, manticore_100_0.2, manticore_100_0, manticore_100_0.1
manticore	AttributeError@<python_internal>/lib/python3.8/site-packages/atheris/function_hooks.py:372						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	AttributeError@<python_internal>/lib/python3.8/site-packages/atheris/function_hooks.py:372						✓			manticore_10000_0.1, manticore_10000_0.2, manticore_1_0.1, manticore_100_0, manticore_1_0.2, manticore_10000_0, manticore_100_0.2, manticore_100_0.1
manticore	ValueError@manticore/manticore/core/smtlib/solver.py:94			✓			✓			manticore_acl2s
mypy	N/A							✓	https://github.com/python/mypy/pull/16897	mypy_10000_0, mypy_10000_0.1, mypy_1_0.1, mypy_1_0, mypy_100_0, mypy_1_0.2, mypy_acl2s, mypy_100_0.2, mypy_100_0.1, mypy_10000_0.2

Bibliography

[1] *AFL (american fuzzy lop)*. URL: https://afl-1.readthedocs.io/en/latest/ (visited on 03/08/2024).

[2] *Announcing OSS-Fuzz: Continuous fuzzing for open source software*. Dec. 2016. URL: https://opensource.googleblog.com/2016/12/announcing-oss-fuzz-continuous-fuzzing.html (visited on 03/07/2024).

[3] Andrea Arcuri and Lionel Briand. "A practical guide for using statistical tests to assess randomized algorithms in software engineering". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. event-place: Waikiki, Honolulu, HI, USA. New York, NY, USA: Association for Computing Machinery, 2011, pp. 1–10. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985795.

[4] *Automatic stub generation (stubgen)*. URL: https://mypy.readthedocs.io/en/stable/stubgen.html (visited on 03/07/2024).

[5] Jayanti Bhandari Neupane et al. "Characterization of Leptazolines A–D, Polar Oxazolines from the Cyanobacterium *Leptolyngbya* sp., Reveals a Glitch with the "Willoughby–Hoye" Scripts for Calculating NMR Chemical Shifts". In: *Organic Letters* 21.20 (Oct. 2019), pp. 8449–8453. ISSN: 1523-7060, 1523-7052. DOI: 10.1021/acs.orglett.9b03216. (Visited on 08/05/2024).

[6] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. "Coverage-based Greybox Fuzzing as Markov Chain". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, 1032–1043. ISBN: 9781450341394. DOI: 10.1145/2976749.2978428.

[7] Luca Borzacchiello, Emilio Coppa, and Camil Demetrescu. "FUZZOLIC: Mixing fuzzing and concolic execution". In: *Computers & Security* 108 (2021), p. 102368.

[8] George Bouliotis and Lucinda Billingham. "Crossing survival curves: alternatives to the log-rank test". In: *Trials* 12.S1 (Dec. 2011), A137. ISSN: 1745-6215. DOI: 10.1186/1745-6215-12-S1-A137. (Visited on 04/12/2024).

[9] Andy Bowden. *Fuzzing: Common Tools and Techniques*. URL: https://www.coalfire.com/the-coalfire-blog/fuzzing-common-tools-and-techniques?feed=blogs (visited on 03/07/2024).

[10] Sergey Bratus et al. “Exploit Programming: From Buffer Overflows to "Weird Machines" and Theory of Computation”. In: *login Usenix Mag.* 36.6 (2011). URL: https://www.usenix.org/publications/login/december-2011-volume-36-number-6/exploit-programming-buffer-overflows-weird.

[11] I. E. Bronshteyn. “Study of defects in a program code in Python”. In: *Programming and Computer Software* 39.6 (Nov. 2013), pp. 279–284. ISSN: 0361-7688, 1608-3261. DOI: 10.1134/S0361768813060017. (Visited on 03/08/2024).

[12] Marcel Böhme and Brandon Falk. “Fuzzing: on the exponential cost of vulnerability discovery”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event USA: ACM, Nov. 2020, pp. 713–724. ISBN: 978-1-4503-7043-1. DOI: 10.1145/3368089.3409729. (Visited on 03/19/2024).

[13] Harsh Chamarthi et al. “The "ACL2" Sedan Theorem Proving System”. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2011. DOI: 10.1007/978-3-642-19835-9_27.

[14] Harsh Raju Chamarthi. “Interactive non-theorem disproving”. PhD thesis. Northeastern University, 2016. DOI: 10.17760/D20467205. (Visited on 08/14/2024).

[15] Harsh Raju Chamarthi, Peter C. Dillinger, and Panagiotis Manolios. “Data Definitions in the ACL2 Sedan”. In: *Electronic Proceedings in Theoretical Computer Science* 152 (June 2014). arXiv:1406.1557 [cs], pp. 27–48. ISSN: 2075-2180. DOI: 10.4204/EPTCS.152.3. (Visited on 03/08/2024).

[16] Harsh Raju Chamarthi and Panagiotis Manolios. “Automated specification analysis using an interactive theorem prover”. In: *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11*. Ed. by Per Bjesse and Anna Slobodová. FMCAD Inc., 2011, pp. 46–53. URL: https://dl.acm.org/doi/10.5555/2157654.2157665.

[17] Harsh Raju Chamarthi and Panagiotis Manolios. “Automated specification analysis using an interactive theorem prover”. In: *2011 Formal Methods in Computer-Aided Design (FMCAD)*. 2011, pp. 46–53.

[18] Harsh Raju Chamarthi et al. “Integrating Testing and Interactive Theorem Proving”. In: *International Workshop on the ACL2 Theorem Prover and its Applications*. EPTCS. 2011. DOI: 10.4204/EPTCS.70.1.

[19] Harsh Raju Chamarthi et al. “Integrating Testing and Interactive Theorem Proving”. In: *Proceedings 10th International Workshop on the ACL2 Theorem Prover and its Applications*. Ed. by David S. Hardin and Julien Schmaltz. Vol. 70. EPTCS. 2011, pp. 4–19. DOI: 10.4204/EPTCS.70.1.

[20] Harsh Raju Chamarthi et al. "Integrating Testing and Interactive Theorem Proving". In: *Electronic Proceedings in Theoretical Computer Science* 70 (Oct. 2011). arXiv:1105.4394 [cs], pp. 4–19. ISSN: 2075-2180. DOI: 10.4204/EPTCS.70.1. (Visited on 08/14/2024).

[21] Jake Corina et al. "DIFUZE: Interface Aware Fuzzing for Kernel Drivers". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, 2123–2138. ISBN: 9781450349468. DOI: 10.1145/3133956.3134069.

[22] Angel Cronin, Lu Tian, and Hajime Uno. "Strmst2 and Strmst2pw: New Commands to Compare Survival Curves Using the Restricted Mean Survival time". In: *The Stata Journal: Promoting communications on statistics and Stata* 16.3 (Sept. 2016), pp. 702–716. ISSN: 1536-867X, 1536-8734. DOI: 10.1177/1536867X1601600310. (Visited on 06/26/2024).

[23] Addison Crump et al. "LIBAFL LIBFUZZER: LIBFUZZER on Top of LIBAFL". In: *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. Melbourne, Australia: IEEE, May 2023, pp. 70–72. ISBN: 9798350301823. DOI: 10.1109/SBFT59156.2023.00021. (Visited on 07/31/2024).

[24] Sandeep Dalal and Rajender Singh Chhillar. "Empirical study of root cause analysis of software failure". In: *ACM SIGSOFT Software Engineering Notes* 38.4 (July 2013), pp. 1–7. ISSN: 0163-5948. DOI: 10.1145/2492248.2492263. (Visited on 03/08/2024).

[25] Neil T. Dantam. *CL-FUZZ*. 2018. URL: https://github.com/ndantam/cl-fuzz (visited on 08/20/2024).

[26] Demetris T Christopoulos. *Reliable computations of knee point for a curve and introduction of a unit invariant estimation*. 2014. DOI: 10.13140/2.1.3111.5844.

[27] Anna van Deusen. *Python Popularity: The Rise of a Global Programming Language*. Jan. 2023. URL: https://flatironschool.com/blog/python-popularity-the-rise-of-a-global-programming-language/ (visited on 06/27/2024).

[28] Luca Di Grazia and Michael Pradel. "The evolution of type annotations in python: an empirical study". In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Singapore Singapore: ACM, Nov. 2022, pp. 209–220. ISBN: 978-1-4503-9413-0. DOI: 10.1145/3540250.3549114. (Visited on 03/08/2024).

[29] Will Dietz et al. "Understanding Integer Overflow in C/C++". In: *ACM Trans. Softw. Eng. Methodol.* 25.1 (Dec. 2015). ISSN: 1049-331X. DOI: 10.1145/2743019.

[30] Peter C Dillinger et al. "ACL2s: "The ACL2 Sedan"". In: *Electronic Notes in Theoretical Computer Science* 174.2 (2007), pp. 3–18.

[31] Yiran Ding et al. *LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens*. arXiv:2402.13753 [cs]. Feb. 2024. URL: http://arxiv.org/abs/2402.13753 (visited on 09/07/2024).

[32] Julian Dolby et al. "Ariadne: analysis for machine learning programs". In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. MAPL 2018. Philadelphia, PA, USA: Association for Computing Machinery, 2018, 1–10. ISBN: 9781450358347. DOI: 10.1145/3211346.3211349.

[33] Ran Dubin. "Disarming Attacks Inside Neural Network Models". In: *IEEE Access* 11 (2023), pp. 124295–124303. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3330141.

[34] Aryaz Eghbali and Michael Pradel. "DynaPyt: a dynamic analysis framework for Python". In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Singapore Singapore: ACM, Nov. 2022, pp. 760–771. ISBN: 978-1-4503-9413-0. DOI: 10.1145/3540250.3549126. (Visited on 08/29/2024).

[35] J. Emmerson and J.M. Brown. "Understanding Survival Analysis in Clinical Trials". In: *Clinical Oncology* 33.1 (Jan. 2021), pp. 12–14. ISSN: 09366555. DOI: 10.1016/j.clon.2020.07.014. (Visited on 08/07/2024).

[36] Andrea Fioraldi et al. "LibAFL: A Framework to Build Modular and Reusable Fuzzers". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. Los Angeles CA USA: ACM, Nov. 2022, pp. 1051–1065. ISBN: 978-1-4503-9450-5. DOI: 10.1145/3548606.3560602. (Visited on 07/31/2024).

[37] Gordon Fraser and Andrea Arcuri. "Whole Test Suite Generation". In: *IEEE Transactions on Software Engineering* 39.2 (Feb. 2013), pp. 276–291. ISSN: 0098-5589. DOI: 10.1109/TSE.2012.14. (Visited on 03/09/2024).

[38] Gordon Fraser and Andreas Zeller. "Mutation-Driven Generation of Unit Tests and Oracles". In: *IEEE Transactions on Software Engineering* 38.2 (Mar. 2012), pp. 278–292. ISSN: 0098-5589. DOI: 10.1109/TSE.2011.93. (Visited on 03/09/2024).

[39] *Fuzzing*. URL: https://owasp.org/www-community/Fuzzing (visited on 06/27/2024).

[40] *Fuzzing*. URL: https://firefox-source-docs.mozilla.org/tools/fuzzing/index.html (visited on 03/07/2024).

[41] *General Python FAQ*. URL: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place (visited on 06/27/2024).

[42] Leon Jay Gleser. "Bootstrap Confidence Intervals: Comment". In: *Statistical Science* 11.3 (Aug. 1996), pp. 219–221. URL: https://www.jstor.org/stable/2246113.

[43] Liang Gong et al. "DLint: Dynamically checking bad coding practices in JavaScript". In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 2015, pp. 94–105.

[44] Google. *How To Split A Fuzzer-Generated Input Into Several*. URL: https://github.com/google/fuzzing/blob/master/docs/split-inputs.md (visited on 03/25/2024).

[45] Google Information Security team. *Announcing the Atheris Python Fuzzer*. URL: https://opensource.googleblog.com/2020/12/announcing-atheris-python-fuzzer.html (visited on 04/06/2024).

[46] Robert J. Grissom and John J. Kim. *Effect sizes for research: univariate and multivariate applications*. 2nd ed. New York: Routledge, 2012.

[47] Samuel Groß et al. "FUZZILLI: Fuzzing for JavaScript JIT Compiler Vulnerabilities". In: *Proceedings 2023 Network and Distributed System Security Symposium*. San Diego, CA, USA: Internet Society, 2023. ISBN: 978-1-891562-83-9. DOI: 10.14722/ndss.2023.24290. (Visited on 03/10/2024).

[48] Emre Güler et al. "Atropos: Effective fuzzing of web applications for server-side vulnerabilities". In: *USENIX Security Symposium*. 2024.

[49] Yimeng Guo et al. "Generating Python Type Annotations from Type Inference: How Far Are We?" In: *ACM Transactions on Software Engineering and Methodology* 33.5 (June 2024), pp. 1–38. ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3652153. (Visited on 06/22/2024).

[50] Csaba Györgyi, Sándor Laki, and Stefan Schmid. "Toward Highly Reliable Programmable Data Planes: Verification of P4 Code Generation". In: *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. Madrid, Spain: IEEE, June 2023, pp. 1–5. ISBN: 9798350399806. DOI: 10.1109/NetSoft57336.2023.10175397. (Visited on 03/08/2024).

[51] Kyunghwa Han and Inkyung Jung. "Restricted Mean Survival Time for Survival Analysis: A Quick Guide for Clinical Researchers". In: *Korean Journal of Radiology* 23.5 (2022), p. 495. ISSN: 1229-6929, 2005-8330. DOI: 10.3348/kjr.2022.0061. (Visited on 04/05/2024).

[52] Quinn Hanam et al. "Finding patterns in static analysis alerts: improving actionable alert ranking". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, 152–161. ISBN: 9781450328630. DOI: 10.1145/2597073.2597100.

[53] Stefan Hanenberg et al. "An empirical study on the impact of static typing on software maintainability". In: *Empirical Software Engineering* 19.5 (Oct. 2014), pp. 1335–1382. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-013-9289-1. (Visited on 03/08/2024).

[54] Xiaoyu He et al. "SoFi: Reflection-Augmented Fuzzing for JavaScript Engines". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, 2229–2242. ISBN: 9781450384544. DOI: 10.1145/3460120.3484823.

[55] Sarah Heckman and Laurie Williams. "A systematic literature review of actionable alert identification techniques for automated static code analysis". In: *Inf. Softw. Technol.* 53.4 (Apr. 2011), 363–387. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2010.12.007.

[56] Sarah Heckman and Laurie Williams. "On establishig a benchmark for evaluating static analysis alert prioritization and classification techniques". In: *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '08. New York, NY, USA: Association for Computing Machinery, 2008, 41–50. ISBN: 9781595939715. DOI: 10.1145/1414004.1414013.

[57] Adrian Herrera, Mathias Payer, and Antony L. Hosking. "DatAFLow : Toward a Data-Flow-Guided Fuzzer". In: *ACM Transactions on Software Engineering and Methodology* 32.5 (Sept. 2023), pp. 1–31. ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3587156. (Visited on 03/08/2024).

[58] Adrian Herrera et al. *Corpus Distillation for Effective Fuzzing: A Comparative Evaluation*. arXiv:1905.13055 [cs]. Sept. 2020. URL: http://arxiv.org/abs/1905.13055 (visited on 03/09/2024).

[59] Adrian Herrera et al. "Seed selection for successful fuzzing". In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. Virtual Denmark: ACM, July 2021, pp. 230–243. ISBN: 978-1-4503-8459-9. DOI: 10.1145/3460319.3464795. (Visited on 03/08/2024).

[60] Sture Holm. "A Simple Sequentially Rejective Multiple Test Procedure". In: *Scandinavian Journal of Statistics* 6.2 (1979), pp. 65–70. ISSN: 03036898, 14679469. URL: http://www.jstor.org/stable/4615733 (visited on 06/04/2024).

[61] Kyriakos Ispoglou et al. "FuzzGen: Automatic Fuzzer Generation". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2271–2287. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/ispoglou.

[62] Vivek Jain et al. "TIFF: Using Input Type Inference To Improve Fuzzing". In: *Proceedings of the 34th Annual Computer Security Applications Conference*. San Juan PR USA: ACM, Dec. 2018, pp. 505–517. ISBN: 978-1-4503-6569-7. DOI: 10.1145/3274694.3274746. (Visited on 03/08/2024).

[63] Hong Jin Kang, Khai Loong Aw, and David Lo. "Detecting false alarms from automatic static analysis tools: how far are we?" In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, 698–709. ISBN: 9781450392211. DOI: 10.1145/3510003.3510214.

[64] E. L. Kaplan and Paul Meier. "Nonparametric Estimation from Incomplete Observations". In: *Journal of the American Statistical Association* 53.282 (June 1958), pp. 457–481. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.1958.10501452. (Visited on 03/20/2024).

[65] Adhishree Kathikar et al. "Assessing the Vulnerabilities of the Open-Source Artificial Intelligence (AI) Landscape: A Large-Scale Analysis of the Hugging Face Platform". In: *2023 IEEE International Conference on Intelligence and Security Informatics (ISI)*. Oct. 2023, pp. 1–6. DOI: 10.1109/ISI58743.2023.10297271.

[66] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. 2000.

[67] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: Case Studies*. Kluwer Academic Publishers, July 2000. DOI: 10.1007/978-1-4757-3188-0.

[68] Matt Kaufmann and J Strother Moore. "ACL2 homepage". In: *See URL http://www. cs. utexas. edu/users/moore/acl2* (2024).

[69] Faizan Khan et al. "An Empirical Study of Type-Related Defects in Python Projects". In: *IEEE Transactions on Software Engineering* 48.8 (Aug. 2022), pp. 3145–3158. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2021.3082068. (Visited on 03/08/2024).

[70] George Klees et al. "Evaluating Fuzz Testing". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Toronto Canada: ACM, Oct. 2018, pp. 2123–2138. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243804. (Visited on 03/08/2024).

[71] Herb Krasner. *The Cost of Poor Software Quality in the US: A 2020 Report*. Tech. rep. Consortium for Information & Software Quality, Jan. 2021. URL: https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf (visited on 03/07/2024).

[72] Wen Li et al. "PyRTFuzz: Detecting Bugs in Python Runtimes via Two-Level Collaborative Fuzzing". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, 1645–1659. ISBN: 9798400700507. DOI: 10.1145/3576915.3623166.

[73] Li, Wen and Ruan, Jinyang and Yi, Guangbei and Cheng, Long and Luo, Xiapu and Cai, Haipeng. "PolyFuzz: Holistic greybox fuzzing of multi-language systems". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023. ISBN: 978-1-939133-37-3.

[74] *libFuzzer - a library for coverage-guided fuzz testing*. URL: https://llvm.org/docs/LibFuzzer.html (visited on 03/09/2024).

[75] Barbara H Liskov and Alan Snyder. "Exception handling in CLU". In: *IEEE transactions on software engineering* 6 (1979), pp. 546–558.

[76] Stephan Lukasczyk and Gordon Fraser. "Pynguin: automated unit test generation for Python". In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, 168–172. ISBN: 9781450392235. DOI: 10.1145/3510454.3516829.

[77] Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser. "Automated Unit Test Generation for Python". In: *Search-Based Software Engineering*. Ed. by Aldeida Aleti and Annibale Panichella. Cham: Springer International Publishing, 2020, pp. 9–24. ISBN: 978-3-030-59762-7.

[78] Ivan do Carmo Machado, Paulo Anselmo da Mota Silveira Neto, and Eduardo Santana de Almeida. "Towards an integration testing approach for software product lines". In: *2012 IEEE 13th International Conference on Information Reuse & Integration (IRI)*. 2012, pp. 616–623. DOI: 10.1109/IRI.2012.6303066.

[79] Panagiotis Manolios and Daron Vroon. "Algorithms for Ordinal Arithmetic". In: *19th International Conference on Automated Deduction (CADE)*. Ed. by Franz Baader. Vol. 2741. Lecture Notes in Computer Science. Springer, 2003, pp. 243–257. DOI: 10.1007/978-3-540-45085-6_19.

[80] Panagiotis Manolios and Daron Vroon. "Integrating Reasoning About Ordinal Arithmetic into ACL2". In: *5th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*. Ed. by Alan J. Hu and Andrew K. Martin. Vol. 3312. Lecture Notes in Computer Science. Springer, 2004, pp. 82–97. DOI: 10.1007/978-3-540-30494-4_7.

[81] Panagiotis Manolios and Daron Vroon. "Ordinal Arithmetic: Algorithms and Mechanization". In: *Journal of Automated Reasoning* 34.4 (2005), pp. 387–423. DOI: 10.1007/s10817-005-9023-9.

[82] Panagiotis Manolios and Daron Vroon. "Termination Analysis with Calling Context Graphs". In: *Computer Aided Verification*. Ed. by Thomas Ball and Robert B. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 401–414. ISBN: 978-3-540-37411-4.

[83] N. Mantel. "Evaluation of survival data and two new rank order statistics arising in its consideration". In: *Cancer Chemotherapy Reports* 50.3 (Mar. 1966), pp. 163–170. ISSN: 0069-0112.

[84] Valentin Manès, Marcel Böhme, and Sang Kil Cha. *FSE2020 - Boosting Fuzzer Efficiency An Information-Theoretic Perspective*. Artwork Size: 639622034 Bytes Pages: 639622034 Bytes. 2020. DOI: 10.6084/M9.FIGSHARE.12415622.V2. (Visited on 04/06/2024).

[85] Henrique Marques, Nuno Laranjeiro, and Jorge Bernardino. "Injecting software faults in Python applications: The OpenStack case study". In: *Empirical Software Engineering* 27.1 (Jan. 2022), p. 20. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-021-10047-9. (Visited on 03/09/2024).

[86] Glenford J. Myers, Corey Sandler, and Tom. Badgett. *The Art of Software Testing*. 3rd ed. Hoboken, N.J: John Wiley & Sons, 2012. ISBN: 9781118133132.

[87] Abhinav Nagpal and Goldie Gabrani. "Python for Data Analytics, Scientific and Technical Applications". In: *2019 Amity International Conference on Artificial Intelligence (AICAI)*. 2019, pp. 140–145. DOI: 10.1109/AICAI.2019.8701341.

[88] Sebastian Neef, Lorenz Kleissner, and Jean-Pierre Seifert. "What All the PHUZZ Is About: A Coverage-guided Fuzzer for Finding Vulnerabilities in PHP Web Applications". In: *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. Singapore Singapore: ACM, July 2024, pp. 1523–1538. ISBN: 9798400704826. DOI: 10.1145/3634737.3661137. (Visited on 08/20/2024).

[89] Geoffrey Neumann, Mark Harman, and Simon Poulding. "Transformed Vargha-Delaney Effect Size". In: Bergamo, Italy, 2015, pp. 318–324. DOI: 10.1007/978-3-319-22183-029. (Visited on 03/22/2024).

[90] OpenAI. *GPT-4o*. May 2024. URL: https://openai.com/index/hello-gpt-4o/ (visited on 09/10/2024).

[91] John-Paul Ore, Carrick Detweiler, and Sebastian Elbaum. "An Empirical Study on Type Annotations: Accuracy, Speed, and Suggestion Effectiveness". In: *ACM Trans. Softw. Eng. Methodol.* 30.2 (Feb. 2021). ISSN: 1049-331X. DOI: 10.1145/3439775.

[92] John-Paul Ore et al. "Assessing the type annotation burden". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. Montpellier France: ACM, Sept. 2018, pp. 190–201. ISBN: 978-1-4503-5937-5. DOI: 10.1145/3238147.3238173. (Visited on 08/07/2024).

[93] Sarah ORourke. *How the Atheris Python Fuzzer Works*. Copyright - Copyright Newstex Dec 9, 2020; Last updated - 2020-12-09. Dec. 2020. URL: https://link.ezproxy.neu.edu/login?url=https://www.proquest.com/blogs-podcasts-websites/how-atheris-python-fuzzer-works/docview/2468364629/se-2.

[94] OWASP Foundation. *OWASP Top Ten*. 2021. URL: https://owasp.org/www-project-top-ten/ (visited on 09/09/2024).

[95] Carlos Pacheco et al. "Feedback-Directed Random Test Generation". In: *29th International Conference on Software Engineering (ICSE'07)*. 2007, pp. 75–84. DOI: 10.1109/ICSE.2007.37.

[96] Joonyoung Park, Inho Lim, and Sukyoung Ryu. "Battles with false positives in static analysis of JavaScript web applications in the wild". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, 61–70. ISBN: 9781450342056. DOI: 10.1145/2889160.2889227.

[97] Parquery AG. *icontract*. Nov. 2023. URL: https://pypi.org/project/icontract/ (visited on 09/09/2024).

[98] *pickle — Python object serialization*. URL: https://docs.python.org/3/library/pickle.html (visited on 03/07/2024).

[99] *pytype*. URL: https://github.com/google/pytype (visited on 06/28/2024).

[100] Hongjin Qian et al. *Are Long-LLMs A Necessity For Long-Context Tasks?* arXiv:2405.15318 [cs]. May 2024. URL: http://arxiv.org/abs/2405.15318 (visited on 09/07/2024).

[101] Ingkarat Rak-amnouykit et al. "Python 3 types in the wild: a tale of two type systems". In: *Proceedings of the 16th ACM SIGPLAN International Symposium on Dynamic Languages*. Virtual USA: ACM, Nov. 2020, pp. 57–70. ISBN: 978-1-4503-8175-8. DOI: 10.1145/3426422.3426981. (Visited on 03/08/2024).

[102] Sebastian Raschka, Joshua Patterson, and Corey Nolet. "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence". In: *Information* 11.4 (2020). ISSN: 2078-2489. DOI: 10.3390/info11040193.

[103] Sanjay Rawat et al. "VUzzer: Application-aware Evolutionary Fuzzing". In: *Proceedings 2017 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2017. ISBN: 978-1-891562-46-4. DOI: 10.14722/ndss.2017.23404. (Visited on 08/08/2024).

[104] Alexandre Rebert et al. "Optimizing seed selection for fuzzing". In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, pp. 861–875.

[105] *RecursionError in pyclbr.readmodule_ex*. Apr. 2023. URL: https://github.com/python/cpython/issues/103864 (visited on 07/13/2024).

[106] Albert Reuther et al. "Interactive supercomputing on 40,000 cores for machine learning and data analysis". In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE. 2018, pp. 1–6.

[107] Robust Intelligence. *Pickle Serialization in Data Science: A Ticking Time Bomb*. URL: https://www.robustintelligence.com/blog-posts/pickle-serialization-in-data-science-a-ticking-time-bomb.

[108] Guido van Rossum, Jukka Lehtosalo, and Łukasz Langa. *PEP 484 - Type Hints*. Sept. 2014. URL: 10.1145/3426422.3426981 (visited on 03/07/2024).

[109] Ori Roth. *Python Type Hints are Turing Complete*. arXiv:2208.14755 [cs]. Aug. 2022. URL: http://arxiv.org/abs/2208.14755 (visited on 03/08/2024).

[110] Patrick Royston and Mahesh Kb Parmar. "Restricted mean survival time: an alternative to the hazard ratio for the design and analysis of randomized trials with a time-to-event outcome". In: *BMC Medical Research Methodology* 13.1 (Dec. 2013), p. 152. ISSN: 1471-2288. DOI: 10.1186/1471-2288-13-152. (Visited on 04/05/2024).

[111] *Rust Programming Language*. URL: https://www.rust-lang.org/ (visited on 08/16/2024).

[112] Rizel Scarlett. *Why Python keeps growing, explained*. Mar. 2023. URL: https://github.blog/2023-03-02-why-python-keeps-growing-explained/ (visited on 06/27/2024).

[113] Matt Schwager, Dominik Klemba, and Josiah Dykstra. "Design and Implementation of a Coverage-Guided Ruby Fuzzer". In: *Proceedings of the 17th Cyber Security Experimentation and Test Workshop*. Philadelphia PA USA: ACM, Aug. 2024, pp. 27–33. ISBN: 9798400709579. DOI: 10.1145/3675741.3675749. (Visited on 08/20/2024).

[114] Agnia Sergeyuk et al. *Using AI-Based Coding Assistants in Practice: State of Affairs, Perceptions, and Ways Forward*. arXiv:2406.07765 [cs]. June 2024. URL: http://arxiv.org/abs/2406.07765 (visited on 09/07/2024).

[115] Naomi Seyfer. *What to do about setters of a different type than their property?* Mar. 2017. URL: https://github.com/python/mypy/issues/3004 (visited on 08/08/2024).

[116] Bhargava Shastry et al. "Static Program Analysis as a Fuzzing Aid". In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Marc Dacier et al. Cham: Springer International Publishing, 2017, pp. 26–47. ISBN: 978-3-319-66332-6.

[117] Jeremy Siek. *What is Gradual Typing*. Mar. 2014. URL: https://wphomes.soic.indiana.edu/jsiek/what-is-gradual-typing/ (visited on 03/07/2024).

[118] A Singhal, T Winograd, and K A Scarfone. *Guide to secure web services*. Tech. rep. NIST SP 800-95. Edition: 0. Gaithersburg, MD: National Institute of Standards and Technology, 2007, NIST SP 800–95. DOI: 10.6028/NIST.SP.800-95. (Visited on 08/29/2024).

[119] Nick Stephens et al. "Driller: Augmenting Fuzzing Through Selective Symbolic Execution". In: *Proceedings 2016 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2016. ISBN: 978-1-891562-41-9. DOI: 10.14722/ndss.2016.23368. (Visited on 08/02/2024).

[120] Evan Sultanik. *Never a dill moment: Exploiting machine learning pickle files*. Mar. 2021. URL: https://blog.trailofbits.com/2021/03/15/never-a-dill-moment-exploiting-machine-learning-pickle-files/ (visited on 03/07/2024).

[121] Gil Tayar et al. *Github - tc39/proposal-type-annotations: ECMAScript proposal: Type Annotations*. URL: https://github.com/tc39/proposal-type-annotations (visited on 08/05/2024).

[122] The Associated Press. "British Airways computer problem strands 20,000 travelers; 100+ London flights canceled". In: *USA Today* (Aug. 2019). URL: https://www.usatoday.com/story/travel/airline-news/2019/08/07/british-airways-computer-glitch-strands-20-000-100-london-flights-canceled/1942804001/ (visited on 03/07/2024).

[123] The Clang Team. *SanitizerCoverage*. URL: https://clang.llvm.org/docs/SanitizerCoverage.html (visited on 04/05/2024).

[124] *The LLVM Compiler Infrastructure*. URL: https://llvm.org/ (visited on 07/20/2024).

[125] the mypy project. *mypy*. URL: https://mypy-lang.org/ (visited on 03/07/2024).

[126] *The top programming languages*. 2022. URL: https://octoverse.github.com/2022/top-programming-languages (visited on 03/07/2024).

[127] Lionel Nganyewou Tidjon and Foutse Khomh. *Threat Assessment in Machine Learning based Systems*. arXiv:2207.00091 [cs]. June 2022. URL: http://arxiv.org/abs/2207.00091 (visited on 03/15/2024).

[128] *Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX*. URL: https://github.com/huggingface/transformers (visited on 03/25/2024).

[129] *TypeScript: JavaScript with syntax for types*. URL: https://www.typescriptlang.org/ (visited on 08/05/2024).

[130] *typing — Support for type hints*. URL: https://docs.python.org/3/library/typing.html (visited on 03/13/2024).

[131] András Vargha and Harold D. Delaney. "A Critique and Improvement of the *CL* Common Language Effect Size Statistics of McGraw and Wong". In: *Journal of Educational and Behavioral Statistics* 25.2 (June 2000), pp. 101–132. ISSN: 1076-9986, 1935-1054. DOI: 10.3102/10769986025002101. (Visited on 03/19/2024).

[132] Till Varoquaux and Konstantin Kashin. *PEP 593 - Flexible function and variable annotations*. Apr. 2019. URL: https://peps.python.org/pep-0593/ (visited on 06/27/2024).

[133] Vasudev Vikram, Rohan Padhye, and Koushik Sen. "Growing a Test Corpus with Bonsai Fuzzing". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Madrid, ES: IEEE, May 2021, pp. 723–735. ISBN: 978-1-66540-296-5. DOI: 10.1109/ICSE43902.2021.00072. (Visited on 03/09/2024).

[134] Alexey Vishnyakov et al. "Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle". In: *arXiv.org* (2023). ISSN: 2331-8422.

[135] Andrew T. Walter, David Greve, and Panagiotis Manolios. "Enumerative Data Types with Constraints". In: *2022 Formal Methods in Computer-Aided Design (FMCAD)*. 2022, pp. 189–198. DOI: 10.34727/2022/isbn.978-3-85448-053-2_25.

[136] Andrew T. Walter, Ankit Kumar, and Panagiotis Manolios. *Calculational Proofs in ACL2s*. arXiv:2307.12224 [cs]. July 2023. URL: http://arxiv.org/abs/2307.12224 (visited on 08/14/2024).

[137] Andrew T. Walter and Panagiotis Manolios. "ACL2s Systems Programming". In: *Proceedings of the Seventeenth International Workshop on the ACL2 Theorem Prover and its Applications*. EPTCS. 2022. DOI: 10.4204/EPTCS.359.12.

[138] Junjie Wang et al. "Skyfire: Data-Driven Seed Generation for Fuzzing". In: *2017 IEEE Symposium on Security and Privacy (SP)*. San Jose, CA, USA: IEEE, May 2017, pp. 579–594. ISBN: 978-1-5090-5533-3. DOI: 10.1109/SP.2017.23. (Visited on 03/08/2024).

[139] Junjie Wang et al. *Superion: Grammar-Aware Greybox Fuzzing*. arXiv:1812.01197 [cs]. Jan. 2019. URL: http://arxiv.org/abs/1812.01197 (visited on 03/08/2024).

[140] Mingzhe Wang et al. "SAFL: increasing and accelerating testing coverage with symbolic execution and guided fuzzing". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. Gothenburg Sweden: ACM, May 2018, pp. 61–64. ISBN: 978-1-4503-5663-3. DOI: 10.1145/3183440.3183494. (Visited on 07/31/2024).

[141] Yuli Wang et al. "JSTIFuzz: Type-Inference-based JavaScript Engine Fuzzing". In: *2020 International Conference on Networking and Network Applications (NaNA)*. 2020, pp. 381–387. DOI: 10.1109/NaNA51271.2020.00071.

[142] Colin Winter and Tony Lownds. *PEP 3107 - Function Annotations*. Dec. 2006. URL: https://peps.python.org/pep-3107/ (visited on 06/27/2024).

[143] Yi Yang, Ana Milanova, and Martin Hirzel. "Complex Python features in the wild". In: *Proceedings of the 19th International Conference on Mining Software Repositories*. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, 282–293. ISBN: 9781450393034. DOI: 10.1145/3524842.3528467.

[144] Insu Yun et al. "QSYM: A practical concolic execution engine tailored for hybrid fuzzing". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 745–761.

[145] Andreas Zeller et al. "Fuzzing with Grammars". In: *The Fuzzing Book*. CISPA Helmholtz Center for Information Security, 2024. URL: https://www.fuzzingbook.org/html/Grammars.html.

[146] Andreas Zeller et al. "Mutation-Based Fuzzing". In: *The Fuzzing Book*. CISPA Helmholtz Center for Information Security, 2023. URL: https://www.fuzzingbook.org/html/MutationFuzzer.html.

[147] Hongyu Zhai, Casey Casalnuovo, and Prem Devanbu. "Test Coverage in Python Programs". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. Montreal, QC, Canada: IEEE, May 2019, pp. 116–120. ISBN: 978-1-72813-412-3. DOI: 10.1109/MSR.2019.00027. (Visited on 03/09/2024).