# Finding Bugs Efficiently

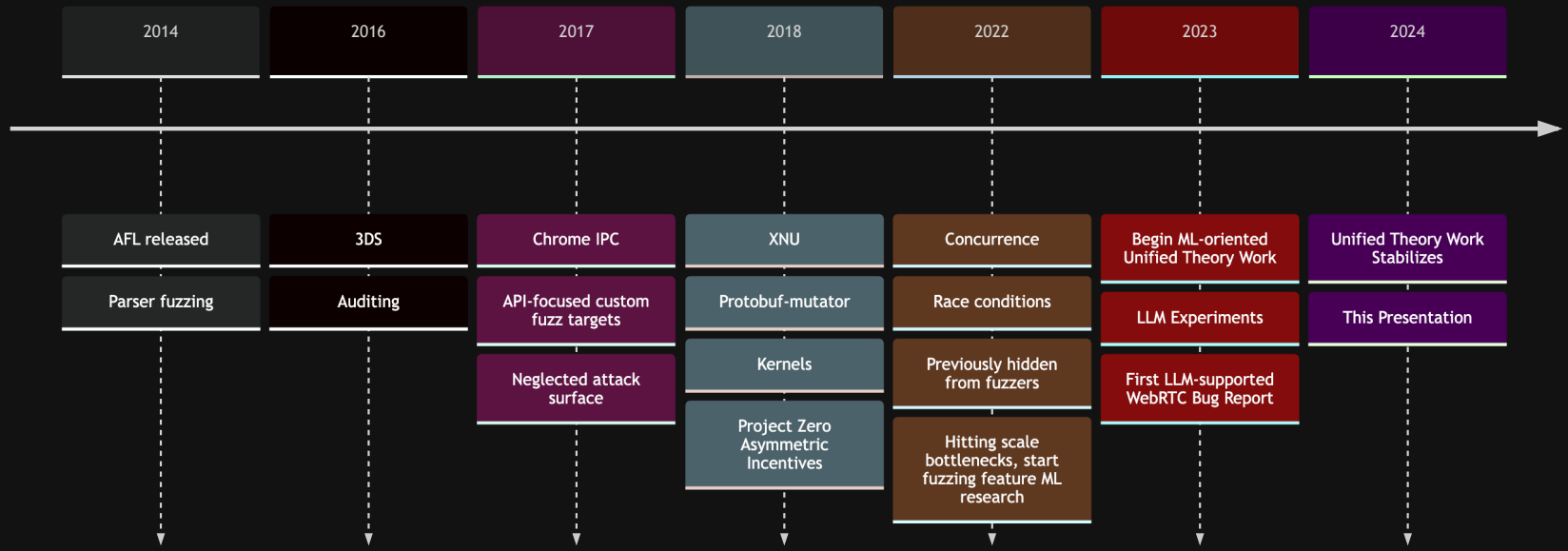
## A Practitioner's Model of Program Analysis

Ned Williamson
ASU Applied Vulnerability Research, 2024

# About Me

- Recently at Google Project Zero
- More recently funemployed
- Focus on fuzzing research & systems security
- Work on developing strategic directions for automated testing
- Experience with:
  - Chrome IPC fuzzing
  - iOS/XNU research
  - Syscall fuzzing at scale

# Research Journey




*Let's start with a recent case that highlights why we need a new approach…*

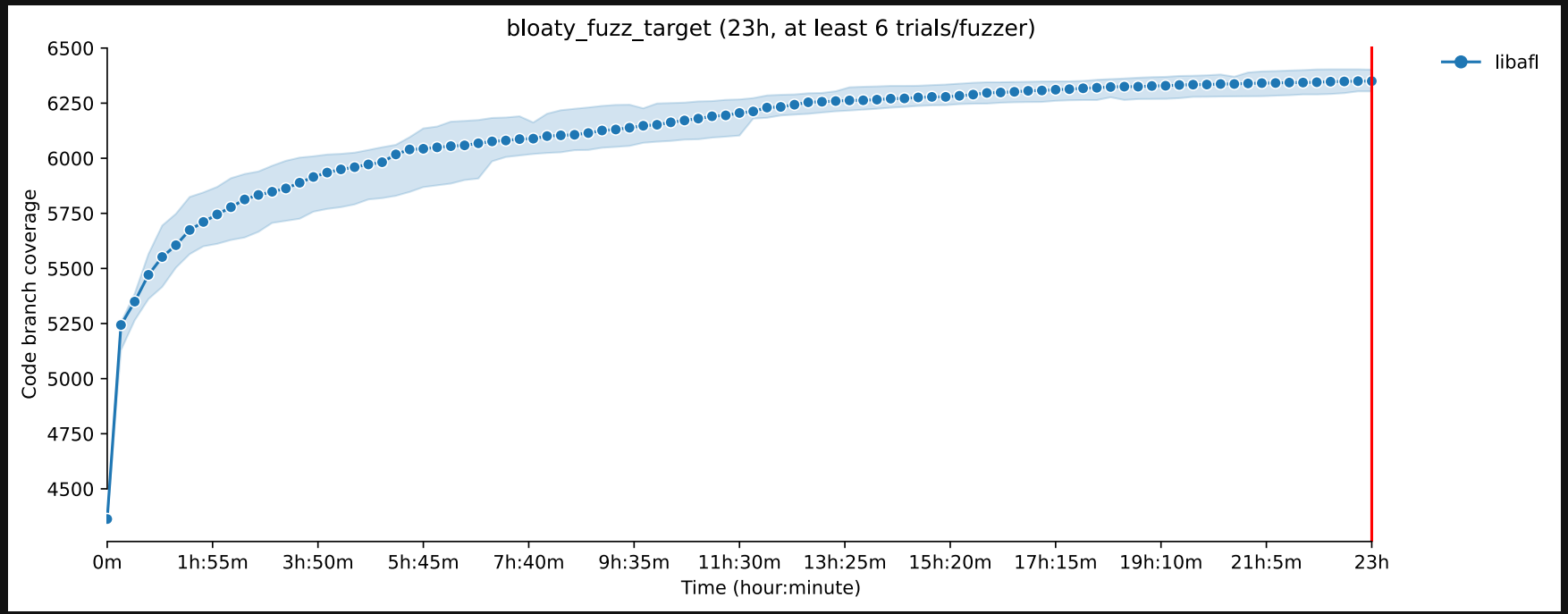
# The WebP Wake-Up Call (CVE-2023-4863)

- Missed by state-of-the-art testing
- Complex Huffman table interactions
- Affected major browsers and mobile devices globally
- Observed in-the-wild

# Why Traditional Tools Failed

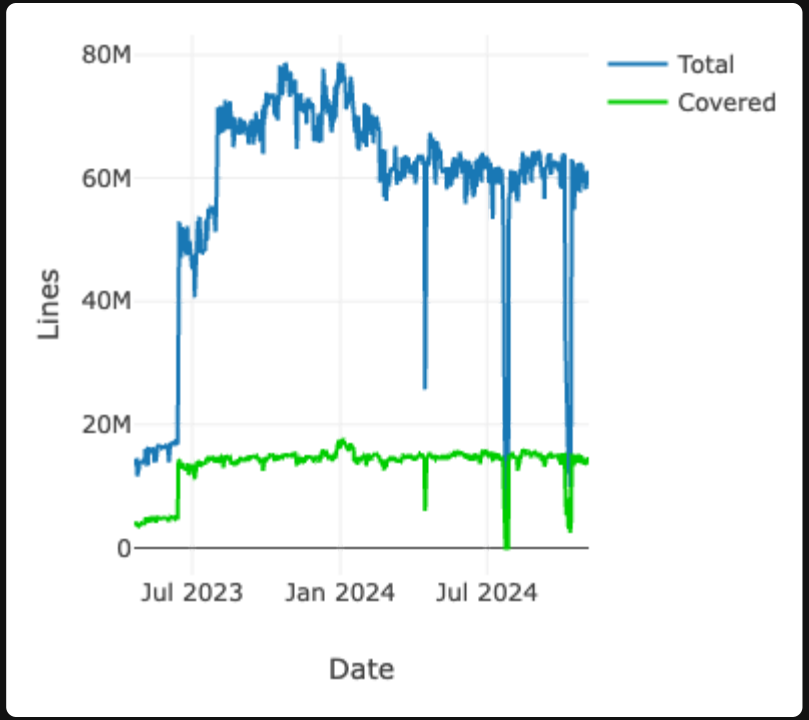
- Coverage metrics showed thorough testing
- Fuzzed for years by OSS-Fuzz
- Crashing test case reproducible with existing fuzz targets
- Critical state interactions missed
- Tools lacked "understanding" of relevant search space

# Typical FuzzBench Report




Source: https://www.fuzzbench.com/reports/2024-10-21-2028-libafl/index.html

# Lines of Code Covered by OSS-Fuzz




Source: https://introspector.oss-fuzz.com/

ML researchers can’t have all the fun! We need our own scaling laws.

# The Evolution of Frontier Bug Finding Automation

1. **Grind Era (2014-2017)**
   - AFL/libFuzzer and grammar-based fuzzing @ Home
   - Symbolic execution research @ CMU, ForAllSecure
   - Focus on specific attack surfaces
2. **Scaling Era (2018-2021)**
   - Exploring grammar and fuzz-target improvements @ Google
     - Pushing target complexity limits with SockFuzzer
     - Concurrence for race condition discovery
3. **Facing Diminishing Expected Value (2022+)**
   - Shifting landscape signals need for fundamental design work
   - Need for theoretical framework

# Why We Need a New Approach

- Limited ceiling using traditional fuzzers
  - Coverage saturation requiring constant grammar updates
  - Fixed representations failing to capture bug patterns
  - Poor support for more complex systems (i.e. V8 engine)
- Memory safety tools (MTE, Rust) changing vulnerability patterns
  - Need new bug classes, potentially new feedback approaches

# From Practice to Theory

- Each phase of research revealed limitations
- Traditional approaches hitting fundamental barriers
- Need unified framework to understand:
  - Why tools miss bugs
  - How different approaches complement each other
  - Where to focus research efforts
- Information-economic framework emerging from practical experience
- Bridging gap between tools and human intuition

# Towards a Unified Framework

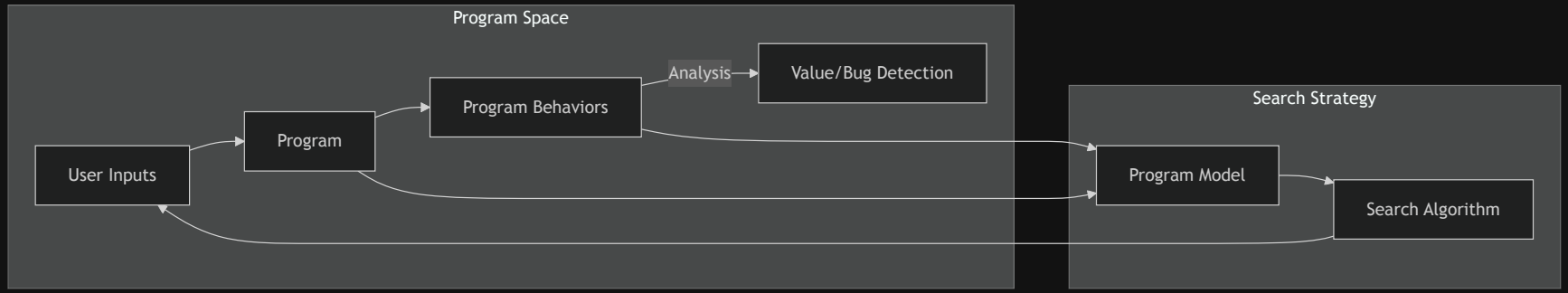
- Building from Experience
  - Existing approach: Generalize fuzz targets from known bugs
  - New direction: Understand search strategies themselves
- Key Elements
  - Representative Hard Problems
    - WebP, V8, CPU vulnerabilities
    - Complex interaction patterns
  - Leverage knowledge of existing approaches
    - Auditing
    - Fuzzing
    - Symbolic execution

# Bug Finding as Search

- The Core Challenge
  - Programs can exhibit infinite possible behaviors
  - Bugs are specific patterns within these behaviors
  - We need efficient ways to find them
- Simple Example: Buffer Overflow
  - Program: Like we saw in the WebP case
  - Behaviors: All possible access patterns
  - Bug: Access beyond array bounds
  - Search: How do we find these cases efficiently?

# Core Concepts

## Programs as Behavior Spaces



# Program & Behavior Space

- Program $P$: Maps inputs $I$ to behaviors $B$
- Behavior Space $B$:
  - High-dimensional space of program states
  - Includes traces, coverage, states
- Value Function $V : B \to O$
  - Maps behaviors to an ordered space $O$
  - Measures desirability of behavior in search
  - Examples of $O$: crashes, ASan issues, severity levels

# Real Example: WebP Case

- Input Space: strings (WebP files)
- Behaviors:
  - Huffman table processing
  - Memory allocation patterns
  - Decoding states
- Value Function:
  - ASan

# Search Strategy & Models

- Search Strategy $S$:
  - Selects inputs $i \in I$
  - Explores behavior space $B$
  - AFL: Evolutionary algorithm (novelty search)
  - Auditing: Code review, manual experiment
- Model $m$ (optional):
  - Guides input selection
  - Conditions on $P$ and observed behaviors $(I, B)$
  - AFL: Coverage maps
- Resource Constraints:
  - Compute costs
  - Memory limitations

# Behaviors can encode all bugs

- Program traces contain all observable behaviors (state)
  - Information includes both static program and semantics
- We can’t compute every trace/find all bugs exhaustively (halting problem, NP-hard, etc.)
- Search is necessary, but we have compute and memory budgets
- How do we pick inputs efficiently?

# What do we learn from traces?

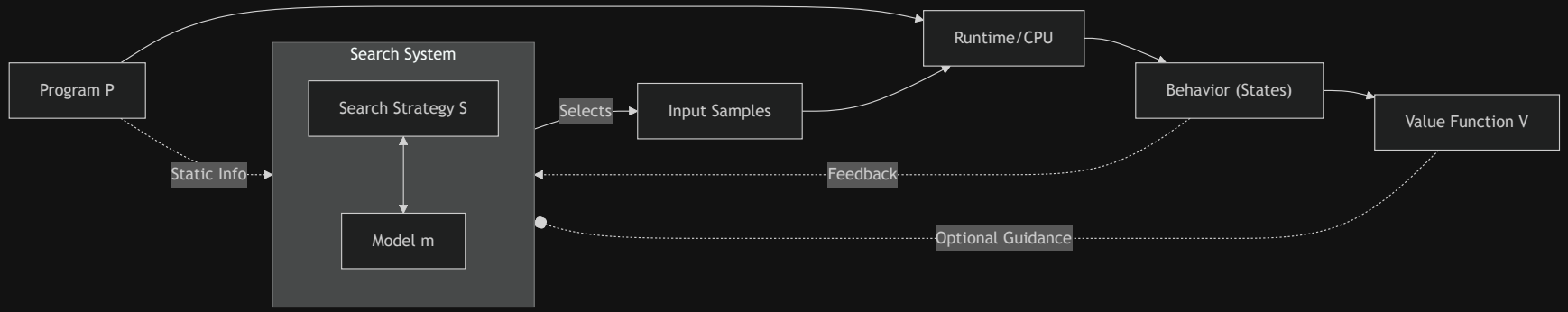
- AFL uses coverage maps (compressed traces) to help maintain diverse behavioral inputs
- Lossy coverage is clearly a bottleneck in the space of all search approaches
- What's the ideal compression?

# The Ideal Trace Compression is the Original Program

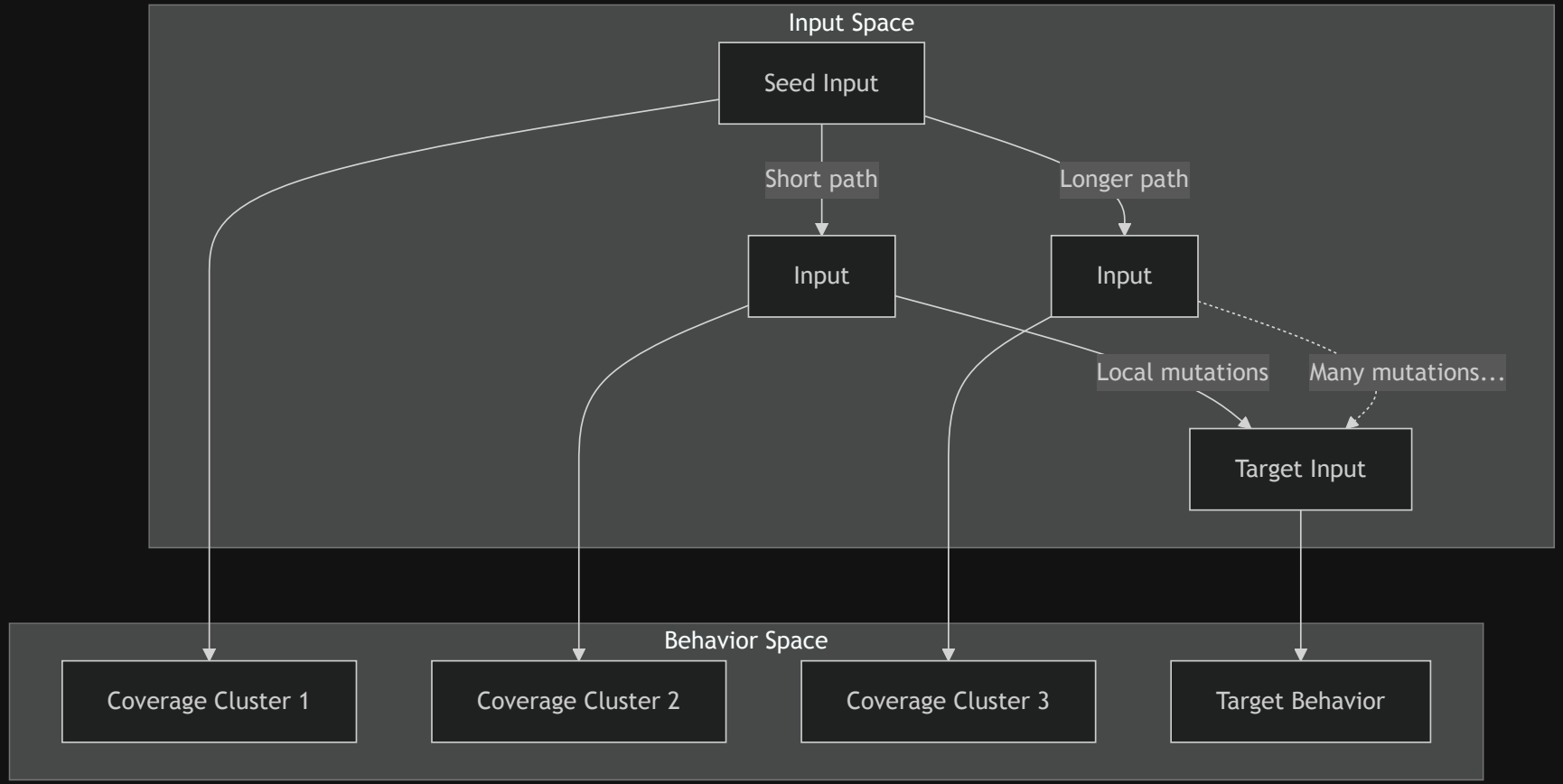
- (Program, Input) $\longrightarrow$ Trace
- Compressing across all traces reduces to optimal program (or optimized version of it)
- Then what is the point of running a test case?

# Compute Requirements

- Fundamentally, compute is required to realize a state
- You can't avoid running the program unless you can prove an optimization
- AFL exploits symmetries between input space and behavior (coverage, trace) space
  - Input mutations are likely to lead to behavioral mutations that are similarly local
  - Preserved inputs are like memoization for spent compute resources
- Program $P$ is therefore an excellent and rich source of information
  - Rather than compress traces, let's transform $P$ (compiler instrumentation!)


Program P
Search System
Search Strategy S
Model m
Static Info
Selects
Input Samples
Runtime/CPU
Behavior (States)
Value Function V
Feedback
Optional Guidance

# AFL: Mutation Dynamics



# Humans

- Rich, slow online learning of $B$
- Model $m$
  - Complex
  - Hierarchical
  - Generalizable
  - Supports tool use
- Value function $V$: Rich understanding leveraging world model
  - Allows for much smaller search space for high value issues
  - Translates to very small and efficient m where possible
- Compute budget $C$/memory $M$: Severely limited
- Combined $m$ and $V$ permit search at very high levels of abstraction

# Exploiting the Framework

- Transformer-based VAE to learn better feature representations
  - Could potentially learn "real" approximation
  - Use model uncertainty to characterize novelty
  - Learn from all executions, even failed ones
  - Learn richer m from coverage, ideally full trace
  - Sample without mutation constraint
  - Needs more design work

# LLMs: AFL Regime

- Leverage to transform $P$ more flexibly than existing compiler instrumentation for target-specific feedback
  - Scales well w.r.t. code size, like coverage
  - Can benefit from upstream model improvements
  - Consistent with information-theoretic density of static program
  - Clean separation of classical compute
- Better mutations
  - Improve grammars
  - Use an agent to review code coverage reports
  - Much broader/higher level structural feedback loop
- Exploit real-world symmetries that evade symbolic systems
  - Knowledge of file formats and protocols
    - Used in my WebRTC fuzzing work
  - Implement $V$

# LLMs: Agent Regime, e.g. Google's Big Sleep

- "World model" provides ability to condition on information from the security community
- Consistent with recent Google Project Zero/DeepMind agent use to find a SQLite bug
  - Echoes my training for fuzzing: variants are excellent supervision signals
    - Initial capability training
    - Exploiting world knowledge as regular part of workflow
- Improving fuzz targets that missed known bugs (webp!)
  - AI labs and research groups investigating program analysis should consider this approach

# WebP Bug Through Framework

- $m$: Coverage too coarse
- $S$: Wrong mutation strategy
- $V$: ASan still worked! Not the problem.

# Kolmogorov Complexity Insights

For a bug $b$: $K(b) = \text{length of shortest program describing } b$

If the bug is human-discoverable:

- $K(b)$ must be bounded by human cognitive limits.
- An efficient description space must exist.
- The challenge is to find the right abstraction level for $m$.
- I've secretly been exploiting this for a decade whenever someone said a bug was "unfuzzable."

# WebP Through Kolmogorov Complexity

- Input space
  - Valid WebP file with 5 Huffman tables, first 4 at max size, 5th table exceeds buffer limit
- Value-aware behavior space
  - If the first Huffman table level is shallow and the second is densely populated, memory allocation for Huffman decoding may exceed the buffer limit
- Intuitively, we can see that the search depth is not excessive w.r.t. reasonable description languages

# Key Implications

- Human discovery implies tractable search path
- Concise description suggests reasonable $K(b)$
- LLMs provide evidence that efficient feature representations are within reach
- For example, an LLM wrote the descriptions on the previous slide
  - Input: Ben Hawkes' blog post
  - Behavioral: @mistymtncop and Ben's reproduction code and upstream WebP source
- Given compute and memory advantages of computers, once they can search sufficiently broadly, they will surpass human depth and discover bugs/paths of depths we can't achieve, just like chess engines
  - You don't need to beat the halting problem
  - You just need to beat humans and their tools
  - But I can't wait to have an eval bar when I'm streaming bug hunting ranked!

# AGI (You Can Stop Listening Now)

- Full V needs world model understanding
  - (Multi-modal) LLMs are already learning a text/image/etc. projection of this
  - ASan implies exploitability implies social impact, the underlying measure of bug value
  - Investing in highly specialized $V$ unlikely to outscale general approaches derived from world models
- Optimal pre-compute/inference model may be discovered
  - Look up active inference for interesting intuitive view about this
- Security reduces to general intelligence
- Implications
  - Work on search problem likely more fruitful in near to mid term until AGI progress converges
  - Very long (?) term, national security may reduce to energy production scale and efficiency if general capabilities are broadly available
    - Humans will probably be more energy efficient for a while, but we are nowhere near as scalable

# Thanks


- Professor Doupé and ASU for hosting me
- Sergei Glazunov
- Mark Brand
- Kostya Serebryany
- Ivan Fratric
- fluorescence
- lokihardt
- Tim Becker
- Tyler Nighswander
- Dominik Christian Maier
- Ian Beer
- Jann Horn
- Brandon Azad
- Tavis Ormandy
- David Aslanian
- Wei Wang

# Q&A