Better Distributional Fit Does Not Improve Reachable Coverage Estimation in Fuzzing

Nelum Attanayake
The University of Sydney
neelum.attanayake@sydney.edu.au

Danushka Liyanage
The University of Sydney
danushka.liyanage@sydney.edu.au

Hoang Nguyen
The University of Sydney
hoang.nguyen@sydney.edu.au

Suranga Seneviratne
The University of Sydney
suranga.seneviratne@sydney.edu.au

Clement Canonne
The University of Sydney
clement.canonne@sydney.edu.au

Rahul Gopinath
The University of Sydney
rahul.gopinath@sydney.edu.au

***Abstract*—The effectiveness of a fuzzing campaign is assessed by program coverage. In complex real-world programs, the total reachable coverage is unknown, making it difficult to determine when fuzzing has sufficiently explored the program. Existing approaches use non-parametric estimators, but these can be inaccurate and unstable, often overestimating the coverage.**

Parametric estimators offer an alternative by assuming an underlying distribution for coverage discovery, potentially improving accuracy and stability, yet remain largely unexplored in fuzzing.

We evaluate parametric models for coverage discovery by fitting Poisson, Exponential, Gamma, Gamma–Poisson, Negative Binomial, and Zipf–Mandelbrot distributions on our real-world and synthetic benchmark programs. Our results show that coverage discovery is highly heterogeneous and heavy-tailed across the analyzed subjects. The Zipf–Mandelbrot model gives the best fit for eight of the twelve subjects, and the Gamma–Poisson model best fit for the remaining four. However, although Zipf–Mandelbrot and Gamma–Poisson fit better than other model families, their estimators show no accuracy advantage over the other non-parametric estimators used in the experiment. This is a negative result: modeling coverage discovery more accurately does not translate into better reachable-coverage estimates.

***Index Terms*—fuzzing, reachability, species estimation, parametric models, negative results**

I. Introduction

Fuzzing is an automated testing technique that uses coverage-guided feedback to explore program behavior [1]. The duration of a fuzzing campaign, and thus its stopping criterion, is typically governed by the maximum achievable coverage [2]. However, reaching 100% coverage in real-world programs is practically infeasible [3]–[5].

Unfortunately, quantifying the maximum reachable coverage is challenging, since static analysis techniques typically cannot accurately account for indirect control flow and incomplete knowledge of the program environment [6]. As a partial remedy, researchers have proposed statistical approaches [5] that model program behavior as a sampling process, an instance of the species estimation problem in biostatistics [7]. This process is then used to statistically estimate reachable coverage.

However, researchers have found these estimators to be imprecise, exhibiting wide confidence intervals and unreliable predictions [8], which limits their practical utility. In particular, they often overestimate the number of reachable program elements, yielding values that exceed the actual count present in the program. A potential reason is that these methods rely primarily on rare-event statistics and do not explicitly account for the underlying structure of the coverage discovery process. This prompted our first question:

***RQ1:** Does the frequency of covering program elements in real-world programs follow a known statistical distribution?* We fit six parametric models (Poisson, Exponential, Gamma, Gamma–Poisson, Negative Binomial, and Zipf–Mandelbrot) to empirical coverage data from seven real-world programs. Our results show that the Zipf–Mandelbrot distribution achieves the best fit for six of the seven real-world programs across all measures: AIC, BIC, and log-likelihood, with Gamma–Poisson the closest competitor.

Establishing the right distribution is a necessary first step, but evaluating estimator accuracy requires programs with known ground truth, which is unavailable for typical real-world software. This presents a chicken-and-egg problem. However, there is a class of programs where precise reachability is knowable: recursive-descent parsers generated from a grammar [9]. Given any grammar, we can translate it into a parser whose coverage space is fully derivable, while the input grammar can be made as large as necessary for statistical estimation to apply. Such parsers closely follow hand-written parsers in structure and are widely deployed in compiling, data extraction, and program analysis. Before using them as benchmarks, however, we must verify that they are statistically representative of real-world programs. Hence, this forms our second question:

***RQ2:** Do synthetic parser programs exhibit the same distributional properties as real-world programs?* We compare the goodness-of-fit of the same six distributions to empirical data from our seven real-world programs and our synthetic parsers. We find that Zipf–Mandelbrot and Gamma–Poisson are the two closest competitors in both settings, with the specific winner varying by subject (see Section IV for the full results).

Finding the right distribution, and confirming it applies to both program categories, allows us to ask the next question: ***RQ3:** Do the best-fit parametric models (Zipf–Mandelbrot and Gamma–Poisson) provide more accurate estimates of total reachable coverage compared to non-parametric estimators?*

Our results show that while these distributions provide the best characterization of coverage discovery, the corresponding estimators achieve no clear accuracy advantage over non-parametric methods. This is a negative result: the theoretically motivated, best-fitting parametric models provide no practical estimation benefit over existing non-parametric estimators, despite the strength of their statistical fit.

Contributions:

- We show that although Zipf–Mandelbrot fits the coverage discovery process better than competing models, neither the Zipf–Mandelbrot-based nor the Gamma–Poisson-based estimator outperforms established non-parametric species richness estimators — a negative result for the premise that better model fit yields better reachability estimates.
- We show that coverage discovery in fuzzing consistently follows a Zipf–Mandelbrot distribution. Specifically, the distribution models per-element detection probabilities.
- We demonstrate that recursive-descent parser programs exhibit the same distributional behavior as real-world fuzzing targets, validating their use as controlled benchmarks with exactly enumerable block totals.

II. Preliminaries

A. Coverage Discovery as a Statistical Process

The STADS framework [5] models fuzzing as a statistical sampling process analogous to species discovery in ecology. Coverage elements (basic blocks, branches, paths) correspond to species, and test inputs correspond to samples. If we partition the fuzzing campaign into T sampling units, we can construct an incidence matrix W where $W_{ij} = 1$ if coverage element i is observed in sampling unit j. From this we derive incidence frequency counts $(f_1, f_2, \ldots, f_T)$, where f_k is the number of elements observed in exactly k sampling units. The quantity f_0, which is the elements present in the program but never observed, is precisely what we wish to estimate: total reachable coverage is $S = S_{\text{obs}} + f_0$.

B. Approximating Coverage Discovery with Parametric Models

We consider six distributions, treated as a sequence of increasingly flexible hypotheses.

Poisson. The Poisson distribution is the natural null hypothesis. It assumes a *homogeneous* population in which every coverage element has the same detection rate λ across sampling units, so the per-element incidence count (the number of sampling units in which it is observed) is Poisson(λ) with a common rate and all elements are equally hard to reach. However, in most real programs this assumption is likely violated. Programs typically contain easy hot-path elements and rare error-handler elements, so detection rates vary widely across the element population. Hence, the Poisson distribution serves primarily as a falsifiable baseline. Rejecting it in favor of a more flexible model confirms that the population is heterogeneous and motivates the distributions below. Prior work that uses Chao1 [10] implicitly assumes a Poisson model [11]; our fitting makes this assumption explicit and testable.

Exponential. The Exponential distribution is the simplest heterogeneous rate prior: per-element detection rates λ_i are drawn from Exponential(β) across the element population, so most elements have very low detection rates and progressively higher rates are exponentially rarer. Under this prior, counts follow a Poisson(λ_i) model with element-specific rates λ_i. Since these rates are not directly observed, we account for their variability across elements by integrating over their distributions, which yields a Geometric distribution for the observed per-element incidence counts. Thus, the proportion of elements observed in exactly k sampling units decays geometrically in k.

Gamma. The Gamma distribution generalizes the Exponential by adding a shape parameter α (i.e., Exponential corresponds to $\alpha = 1$), so per-element detection rates $\lambda_i \sim \text{Gamma}(\alpha, \beta)$ can span a wide range: from hot-path blocks covered by nearly every input to deeply nested branches reached only under rare conditions. Used as a rate prior on λ_i, the Gamma is the canonical model for heterogeneity in positive-valued detection rates across the element population, and its induced marginal on per-element incidence counts is distributed as a Negative Binomial (see below). Prior work on fuzzing has observed exponentially diminishing returns, where the rate of discovering new coverage elements decreased over time [12], which is consistent with a long-tailed Gamma rate distribution.

Negative Binomial and Gamma–Poisson. In the incidence setting, consider T sampling units. Each element i has a per-sampling-unit detection probability p_i, and the number of sampling units in which it appears follows Binomial(T, p_i). The principled parametric model for the distribution of p_i across the element population is the Beta distribution, with Beta–Binomial marginal for the per-element incidence count. When T is large and each p_i is small with $\lambda_i = Tp_i$ held finite, Binomial(T, p_i) $\approx$ Poisson(λ_i), and marginalizing this Poisson over a Gamma(α, β) prior on the rate λ_i yields a Negative Binomial for the per-element incidence count. The Negative Binomial and Gamma–Poisson models are thus two views of the same hierarchy: the Gamma is the continuous rate prior on λ_i, and the NB is its induced discrete marginal on incidence counts. We include the NB because it is the standard discrete companion to the Gamma in the species estimation literature [13] and to test empirically how well the Poisson approximation holds for the incidence counts observed in our data. For species richness estimation, however, the Gamma–Poisson framing is preferable because it exposes the rate distribution directly, from which the probability of an element being entirely unobserved, the key quantity for estimating f_0, is the zero-class probability of the induced Negative Binomial, namely $(\beta/(\beta+1))^{\alpha}$ [13].

Zipf–Mandelbrot. The Zipf–Mandelbrot law is a rank-frequency law which predicts the incidence count $f(r)$ of the element of rank r as

$$f(r) = C(r+q)^{-s}, \qquad C = \left(\sum_{r=1}^{S_{\text{obs}}} (r+q)^{-s} \right)^{-1},$$

We obtain the incidence count for each observed element as

the sum of the incidence indicators for that element across sampling units. We denote the total incidence count across all observed elements as J.

Then, the incidence count predicted by the model for the element of rank r is $\hat{x}_r = JC(r+q)^{-s}$.

The parameter estimates $\hat{s}$ and $\hat{q}$ are obtained from the nonlinear least squares fit to the observed and predicted rank-ordered incidence counts, which are computed with the help of the L-BFGS-B optimization algorithm with $s \in [10^{-6}, 10]$ and $q \in [0, 100]$.

Figure 2 and Figure 1 show the observed incidence counts ordered by ascending values, such that the rarest elements have ranks plotted on the left-hand sides of the graphs. For model fitting and estimation we use the descending rank ordering.

To estimate the total number of elements, we need to extrapolate the fitted rank–frequency curve beyond the observed ranks. We define the estimate of the total number of elements as the highest rank such that the corresponding predicted incidence count is at least one, or, alternatively, it is the rank right before the predicted incidence count falls below one.

C. Species Richness Estimators

Here, we only describe the estimators based on best-fitting models that are used in this paper.

Parametric estimators assume a specific form for the distribution of per-element detection probabilities and derive the number of unseen elements from the fitted model.

1) *Gamma-Poisson mixture:* As indicated in Section II-B, the Gamma-Poisson distribution assumes a rate for each element, $\lambda_i \sim \text{Gamma}(\alpha, \beta)$. Because, the data consist only of elements that have been observed at least once, we use maximum likelihood estimation to determine α and β based on the zero-truncated Gamma-Poisson likelihood. The estimated probability of non-occurrence of an element under the fitted model is
$$\hat{p}_0 = \left(\frac{\hat{\beta}}{\hat{\beta}+1}\right)^{\hat{\alpha}}.$$
Thus, the estimated probability of occurrence of an element is $1-\hat{p}_0$, and the total richness is estimated as $\hat{S} = \frac{S_{\text{obs}}}{1-\hat{p}_0}$.
2) *Zipf–Mandelbrot estimator:* The fitted rank–frequency curve $f(r) = C/(r+q)^s$ is extrapolated beyond the observed ranks. The estimated total number of elements is the largest rank $\hat{S}$ such that the predicted frequency $f(\hat{S}) \geq 1$, i.e., the rank at which the model predicts that elements begin to occur less than once on average and are therefore unobserved.

Given the distributional model above, species richness estimators use the observed frequency counts $(f_1, f_2, \ldots)$ to infer f_0 and hence total reachable coverage. Estimators divide into two families depending on whether they commit to a distributional assumption.

Non-parametric estimators: These estimators do not assume any specific distribution over per-element detection probabilities. They rely solely on observed frequency counts, in particular singletons (f_1) and doubletons (f_2), to extrapolate the number of unseen elements. We list below the commonly used non parametric species richness estimators applicable to fuzzing [5], [10], [14].

1) *Chao2* [15] is a non-parametric estimator that provides a lower-bound estimate of species richness by leveraging rare frequencies (singletons—f_1 and doubletons—f_2), to account for undetected species: $\hat{S} = S_{\text{obs}} + \frac{f_1^2}{2f_2}$, operating on presence–absence data across sampling units.
2) *Chao2_bc* [16] applies a bias correction to Chao2.
3) *Jackknife Estimators* [17] correct for singletons across sampling units (Jackknife 1): $\hat{S} = S_{\text{obs}} + f_1(T-1)/T$, where T is the number of sampling units. Second–order Jackknife (JK2) uses both singletons and doubletons to estimate species richness.
4) *Lanumteang–Böhning* [18] extends Chao's estimator, which uses only singletons and doubletons, by incorporating the first three frequency counts f_1, f_2, f_3 under a Gamma–Poisson mixture framework. It estimates the number of unobserved individuals via a log-linear approximation of the monotonic ratios between neighboring frequency counts, reducing bias and increasing efficiency over Chao's estimator in heterogeneous populations.
5) *Moment-based Chiu Estimator* [13] fits the Gamma–Poisson mixture using a moment estimation method rather than maximum likelihood, drawing on the Good–Turing frequency formula [19] to estimate the number of unseen species from the inferred zero-count probability. Using singletons, doubletons, and tripletons, rather than just the first two frequency counts, gives more stable, lower-variance estimates in heterogeneous populations.

III. Methodology

A. Research Questions

Our study is structured in two complementary phases. First, we collect coverage data from fuzzing campaigns and model the observed coverage discovery process using a range of statistical distributions. Second, we evaluate the behavior and reliability of coverage estimators under both controlled settings with known ground truth and realistic scenarios where the ground truth is unknown.

We formulate the following research questions:

- **RQ1:** To what extent can coverage discovery in real-world programs be characterized by known statistical distributions?
- **RQ2:** Do synthetic parser programs exhibit distributional properties consistent with those observed in real-world programs?
- **RQ3:** Does the best-fitting parametric model yield more accurate estimates of total reachable coverage compared to established non-parametric estimators?

B. Benchmark Programs

Our study uses two categories of programs: synthetic programs with known reachability and real-world programs without known reachability.

TABLE I
STRUCTURAL AND COVERAGE CHARACTERISTICS OF THE SYNTHETIC PARSER SUBJECTS

Synth. Pg	$\|N\|$	$\|P\|$	LOC	#Basic Blks	Covered Blks
ParserA	55	728	55,245	104,952	9,995
ParserB	47	529	103,014	194,852	19,502
ParserC	49	744	70,282	136,455	8,897
ParserD	51	1,025	118,588	231,510	12,236
ParserE	59	1,534	169,435	331,301	14,186

$|N|$: number of non-terminals; $|P|$: number of production rules. $\#BasicBlks$: number of basic blocks in the instrumented binary (upper bound on reachable coverage.)

TABLE II
REAL-WORLD SUBJECT PROGRAMS

Real-World Pg	LOC	# Basic Blks	Covered Blks
jsoncpp	8,437	5,962	1,108
libtiff	27,351	15,548	6,157
libxml2	74,672	66,106	12,877
libpcap	45,939	6,456	1,723
jasper	25,096	14,422	4,582
freetype2	89,262	27,527	7,031
zlib	11,555	3,289	632

Synthetic programs with known reachability. We use synthetic parser generation to build large fuzzing benchmarks with rich control-flow structure. Any structured program can be represented as a static control-flow graph annotated with data-flow and constraints, so generating arbitrary control-flow graphs yields an unbiased population of program skeletons covering all feasible path diversity.

From Control-Flow to Grammar. A control-flow graph is built from statement sequences, conditionals (`if-then-else`), loops (`while`), and subroutine calls, each of which maps directly to an LL(1) grammar construct. A conditional `if C: A else: B` becomes $\langle \bullet \rangle \to \langle C \rangle\ \langle IF^i \rangle$ with $\langle IF^i \rangle \to$ `'T'` $\langle A \rangle \mid$ `'F'` $\langle B \rangle$; a loop `while C: B` becomes $\langle \bullet \rangle \to \langle C \rangle\ \langle L^i \rangle$ with $\langle L^i \rangle \to$ `'T'` $\langle B \rangle\ \langle C \rangle\ \langle L^i \rangle \mid$ `'F'`; and each procedure definition becomes a nonterminal. LL(1) grammars are thus isomorphic to structured program control-flow, and a well-known construction [20] turns any LL(1) grammar into a recursive-descent parser. Since handwritten parsers are common fuzzing subjects, we generate arbitrary LL(1) grammars, convert them into recursive-descent parsers, and use the resulting programs as evaluation subjects.

Grammars and Parser Construction. We use standard context-free grammar notation with *terminals* (language symbols) and *nonterminals* expanded by *production rules*. LL(1) grammars additionally have no left recursion, are left-factored, and permit single-token lookahead to choose a rule. From such a grammar we construct a recursive-descent parser by turning each nonterminal into a procedure that dispatches on the lookahead token, with tail recursion rewritten as a `while` loop. For each grammar, during construction, we ensure that every nonterminal defined is reachable from the start symbol. That is, the reachability of the recursive–descent parser is 100%.

TABLE III
MODEL FIT COMPARISON FOR REAL-WORLD PROGRAMS

	Model	AIC	BIC	LogLik
jsoncpp	Exp.	1.63×10^4	1.63×10^4	-8.13×10^3
	Gamma	1.53×10^4	1.53×10^4	-7.64×10^3
	Gamma–P.	1.53×10^4	1.53×10^4	-7.64×10^3
	N. Bin.	1.53×10^4	1.53×10^4	-7.64×10^3
	Poisson	9.98×10^4	9.98×10^4	-4.99×10^4
	Zipf–M.	$\mathbf{1.43 \times 10^4}$	$\mathbf{1.43 \times 10^4}$	$\mathbf{-7.17 \times 10^3}$
libtiff	Exp.	8.25×10^6	8.25×10^6	-4.12×10^6
	Gamma	8.25×10^6	8.25×10^6	-4.12×10^6
	Gamma–P.	8.31×10^4	8.31×10^4	-4.15×10^4
	N. Bin.	8.31×10^4	8.31×10^4	-4.15×10^4
	Poisson	1.63×10^6	1.63×10^6	-8.14×10^5
	Zipf–M.	$\mathbf{8.19 \times 10^4}$	$\mathbf{8.19 \times 10^4}$	$\mathbf{-4.10 \times 10^4}$
libxml2	Exp.	1.73×10^7	1.73×10^7	-8.66×10^6
	Gamma	1.73×10^7	1.73×10^7	-8.66×10^6
	Gamma–P.	$\mathbf{1.77 \times 10^5}$	$\mathbf{1.77 \times 10^5}$	$\mathbf{-8.83 \times 10^4}$
	N. Bin.	1.77×10^5	1.77×10^5	-8.83×10^4
	Poisson	4.75×10^6	4.75×10^6	-2.37×10^6
	Zipf–M.	1.82×10^5	1.82×10^5	-9.11×10^4
libpcap	Exp.	2.49×10^4	2.49×10^4	-1.24×10^4
	Gamma	2.40×10^4	2.40×10^4	-1.20×10^4
	Gamma–P.	2.40×10^4	2.40×10^4	-1.20×10^4
	N. Bin.	2.40×10^4	2.40×10^4	-1.20×10^4
	Poisson	1.96×10^5	1.96×10^5	-9.78×10^4
	Zipf–M.	$\mathbf{2.22 \times 10^4}$	$\mathbf{2.22 \times 10^4}$	$\mathbf{-1.11 \times 10^4}$
jasper	Exp.	6.66×10^4	6.66×10^4	-3.33×10^4
	Gamma	6.63×10^4	6.63×10^4	-3.31×10^4
	Gamma–P.	6.63×10^4	6.63×10^4	-3.31×10^4
	N. Bin.	6.63×10^4	6.63×10^4	-3.31×10^4
	Poisson	7.25×10^5	7.25×10^5	-3.62×10^5
	Zipf–M.	$\mathbf{6.09 \times 10^4}$	$\mathbf{6.09 \times 10^4}$	$\mathbf{-3.04 \times 10^4}$
freetype2	Exp.	1.03×10^5	1.03×10^5	-5.14×10^4
	Gamma	1.03×10^5	1.03×10^5	-5.13×10^4
	Gamma–P.	1.02×10^5	1.03×10^5	-5.12×10^4
	N. Bin.	1.02×10^5	1.03×10^5	-5.12×10^4
	Poisson	1.19×10^6	1.19×10^6	-5.96×10^5
	Zipf–M.	$\mathbf{9.47 \times 10^4}$	$\mathbf{9.47 \times 10^4}$	$\mathbf{-4.74 \times 10^4}$
zlib	Exp.	9.46×10^3	9.47×10^3	-4.73×10^3
	Gamma	6.83×10^3	6.83×10^3	-3.41×10^3
	Gamma–P.	6.79×10^3	6.80×10^3	-3.39×10^3
	N. Bin.	6.79×10^3	6.80×10^3	-3.39×10^3
	Poisson	7.54×10^3	7.55×10^3	-3.77×10^3
	Zipf–M.	$\mathbf{6.46 \times 10^3}$	$\mathbf{6.47 \times 10^3}$	$\mathbf{-3.23 \times 10^3}$

Controlling Program Complexity. Grammar generation exposes several knobs for precise complexity control: the number of nonterminals (procedures), the number and length of production rules (branching complexity), the depth and type of recursion (direct, indirect, or linear), and the fraction of unreachable nonterminals simulating dead code. Together these parameters yield large structured parser programs with complex control-flow. For this study, we generate synthetic programs with 55K to 169K lines of code, with 47 to 59 nonterminals

TABLE IV
MODEL FIT COMPARISON FOR SYNTHETIC SUBJECTS

	Model	AIC	BIC	LogLik
ParserA	Exp.	1.33×10^7	1.33×10^7	-6.63×10^6
	Gamma	1.41×10^5	1.41×10^5	-7.03×10^4
	Gamma–P.	1.41×10^5	1.41×10^5	-7.02×10^4
	N. Bin.	1.41×10^5	1.41×10^5	-7.02×10^4
	Poisson	2.07×10^6	2.07×10^6	-1.03×10^6
	Zipf–M.	$\mathbf{1.33 \times 10^5}$	$\mathbf{1.33 \times 10^5}$	$\mathbf{-6.66 \times 10^4}$
ParserB	Exp.	2.59×10^7	2.59×10^7	-1.29×10^7
	Gamma	2.59×10^7	2.59×10^7	-1.29×10^7
	Gamma–P.	$\mathbf{2.55 \times 10^5}$	$\mathbf{2.55 \times 10^5}$	$\mathbf{-1.28 \times 10^5}$
	N. Bin.	2.55×10^5	2.55×10^5	-1.28×10^5
	Poisson	5.12×10^6	5.12×10^6	-2.56×10^6
	Zipf–M.	2.56×10^5	2.56×10^5	-1.28×10^5
ParserC	Exp.	1.18×10^7	1.18×10^7	-5.90×10^6
	Gamma	1.23×10^5	1.23×10^5	-6.13×10^4
	Gamma–P.	1.23×10^5	1.23×10^5	-6.13×10^4
	N. Bin.	1.23×10^5	1.23×10^5	-6.13×10^4
	Poisson	1.47×10^6	1.47×10^6	-7.36×10^5
	Zipf–M.	$\mathbf{1.13 \times 10^5}$	$\mathbf{1.13 \times 10^5}$	$\mathbf{-5.66 \times 10^4}$
ParserD	Exp.	1.62×10^7	1.62×10^7	-8.11×10^6
	Gamma	1.62×10^7	1.62×10^7	-8.11×10^6
	Gamma–P.	$\mathbf{1.59 \times 10^5}$	$\mathbf{1.59 \times 10^5}$	$\mathbf{-7.96 \times 10^4}$
	N. Bin.	1.59×10^5	1.59×10^5	-7.96×10^4
	Poisson	4.65×10^6	4.65×10^6	-2.33×10^6
	Zipf–M.	1.66×10^5	1.66×10^5	-8.31×10^4
ParserE	Exp.	1.88×10^7	1.88×10^7	-9.41×10^6
	Gamma	1.88×10^7	1.88×10^7	-9.41×10^6
	Gamma–P.	$\mathbf{1.80 \times 10^5}$	$\mathbf{1.80 \times 10^5}$	$\mathbf{-8.98 \times 10^4}$
	N. Bin.	1.80×10^5	1.80×10^5	-8.98×10^4
	Poisson	5.57×10^6	5.57×10^6	-2.78×10^6
	Zipf–M.	1.93×10^5	1.93×10^5	-9.66×10^4

per parser and unrestricted recursion depth. (Table I).

Real-world programs with unknown reachability. To complement the synthetic evaluation, we include widely used fuzzing targets with unknown reachable coverage, with diverse input formats and behaviors. These are described in Table II.

C. Data Collection

We adopt the STADS sampling framework (Section II-A [5]). Using AFL++ [21], we conduct $K = 3$ independent seven-day trials per subject to capture late-stage coverage behaviour where discovery rates stabilize and statistical properties become more reliable. Coverage observations are aggregated into sampling units over fixed time intervals to construct the incidence matrix W and frequency counts $(f_1, f_2, \ldots)$ as defined in Section II-A.

D. Statistical Distribution Fitting

We fit each of the six parametric models introduced in Section II-B to the observed incidence frequency data from each subject. Rate-based probabilistic models are estimated using maximum likelihood. The Zipf–Mandelbrot rank–frequency model uses a nonlinear least squares fitting method (Section II-B). Fit is evaluated using log-likelihood and model selection criteria AIC and BIC. This analysis is performed independently on real-world programs and synthetic parser benchmarks.

E. Coverage Estimators

Based on our model fit, we use an estimator based on a Gamma–Poisson mixture, and another based on Zipf–Mandelbrot, as described in Section II-C.

Non-parametric estimators, including Chao2, Jackknife1, and LB, rely only on observed incidence frequencies and do not assume a specific distribution.

Parametric estimators use fitted statistical models to extrapolate beyond observed data and estimate the number of unseen coverage elements.

Estimator accuracy is evaluated on synthetic programs, where ground truth is available. For real-world programs, evaluation focuses on the stability and consistency of estimates.

F. Evaluation Metrics

Estimator accuracy. For synthetic programs, where ground truth S is known, we measure accuracy using mean bias over K trials, *mean bias*$(t) = \frac{1}{KS} \sum_{i=1}^{K} \left(\hat{S}_i(t) - S \right)$. We also report variance across trials and confidence interval coverage.

Estimator stability. For real-world programs with unavailable ground truth, we evaluate the consistency of estimates across runs and over time. Stable estimators are expected to show low variability and consistent trends as more data is observed.

Model selection. To compare parametric distribution fits across models, we use the Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) and maximised log-likelihood $\ell(\hat{\theta})$. Given the number of parameters k, and the number of observed data points n, these are defined as $\ell(\hat{\theta}) = \sum_{i=1}^{n} \log P(x_i \mid \hat{\theta})$, $AIC = 2k - 2\,\ell(\hat{\theta})$, and $BIC = k \ln(n) - 2\,\ell(\hat{\theta})$.

Where x_i is the i-th observed value (incidence frequency), k is the number of parameters, and $n = S_{obs}$ is the number of covered blocks. A model is the best fit when it achieves the lowest AIC or BIC. For log-likelihood, the highest value indicates the best fit.

IV. RESULTS

A. RQ1: Goodness of Fit of Distributional Models for Coverage Discovery in Real-World Programs

We test whether coverage discovery in fuzzing follows a simple, uniform pattern or shows uneven, varying behavior.

H0: Coverage discovery follows memoryless processes, and no model consistently provides superior fit across subjects.

H1: Coverage discovery is uneven and is better modeled by flexible distributions.

To evaluate this hypothesis, we fit six parametric distributions to coverage discovery data across all subjects: Poisson, Exponential, Gamma, Gamma–Poisson, Negative Binomial, and Zipf–Mandelbrot. Model fit is assessed using AIC, BIC, and log-likelihood.

Table III shows the goodness of fit of the six parametric models fitted to coverage discovery data from seven real-world

TABLE V
FITTED ZIPF–MANDELBROT DISTRIBUTION PARAMETERS PER SUBJECT

	Subject	C	s	q
Real-world Pg	jsoncpp	4.55×10^{-3}	0.258	100
	libtiff	6.19×10^{-3}	0.479	100
	libxml2	1.25×10^{-3}	0.334	100
	libpcap	4.68×10^{-3}	0.316	100
	jasper	1.66×10^{-3}	0.274	100
	freetype2	1.16×10^{-3}	0.270	100
	zlib	2.24×10^{-3}	0.059	100
Synthetic Pg	ParserA	1.22×10^{-3}	0.307	100
	ParserB	2.28×10^{-3}	0.437	100
	ParserC	1.71×10^{-3}	0.340	100
	ParserD	3.45×10^{-3}	0.454	100
	ParserE	3.34×10^{-3}	0.461	100

programs. The Zipf–Mandelbrot distribution achieves the lowest AIC and BIC and the highest log-likelihood across six of the seven real-world subjects; for libxml2, the Gamma–Poisson model achieves a marginally better fit (1.77×10^5 vs. 1.82×10^5), indicating a consistently superior fit. The fitted Zipf–Mandelbrot parameters are given in Table V.

Coverage discovery in fuzzing is best modeled by the Zipf–Mandelbrot distribution, which achieves the lowest AIC and BIC, and the highest log-likelihood, for six of the seven real-world programs evaluated. Gamma-Poisson is the best-fit model for the remaining subject. These findings indicate strong heterogeneity in heavy-tailed coverage frequencies.

B. RQ2: Do Synthetic Parser Programs Exhibit the Same Distributional Properties as Real-World Programs?

Table IV shows the same model-fit comparison for five synthetic parser-based benchmarks. The pattern broadly mirrors the real-world results, but not exactly: Zipf–Mandelbrot achieves the best fit for two of the five parsers (ParserA, ParserC), while Gamma–Poisson achieves the best fit for the remaining three (ParserB, ParserD, ParserE), with margins ranging from negligible (0.4% for ParserB) to substantial (7.2% for ParserE). In both settings, Zipf–Mandelbrot and Gamma–Poisson are each other's closest competitors, and both consistently and substantially outperform Poisson, Exponential, and plain Gamma. This supports the use of synthetic parsers as controlled benchmarks whose statistical behavior is representative of real-world fuzzing targets, though it also shows that the ranking between the two best-fitting models is not fixed and can vary by subject. Parameter s controls the rate at which the frequency of the count decreases with rank. Parameter q determines the curvature of the relationship between rank and frequency. The constant C acts as a normalizing constant to make $(r+q)^{-s}$ sum up to 1. Multiplying by the total number of incidence count J normalizes these rank weights such that the total number of predicted incidence will sums up to J.

Synthetic parser programs exhibit the same qualitative distributional behavior as real-world programs: coverage discovery is heavy-tailed and heterogeneous in both settings, with Zipf–Mandelbrot and Gamma–Poisson as the two closest competitors. Which of the two achieves the best fit varies by subject (Zipf–Mandelbrot for two of five parsers, Gamma–Poisson for the other three), validating parsers as realistic benchmarks while showing that no single model dominates uniformly.

C. RQ3: Do the Zipf–Mandelbrot and Gamma–Poisson Estimators Outperform Non-Parametric Estimators?

Table VI gives a per-subject breakdown of known total blocks, observed coverage, individual estimates, and relative error for Gamma–Poisson, Zipf–Mandelbrot, and the five non-parametric estimators, for all five synthetic parsers evaluated (ParserA–ParserE). For the synthetic parsers, the enumerated total number of basic blocks is taken as the common reference upper bound. The main focus of this comparison is the estimation process. All techniques are applied to the same observations with the same block total.

Furthermore, Table VII reports the bias, variance, and relative error of every estimator evaluated.

Comparing the estimators, most achieve nearly identical results on parsers. The Jackknife1 estimator achieves the smallest mean bias magnitude (186,687) of all seven estimators evaluated, the best result overall. The estimates from the parametric-model-based estimators are unremarkable: despite the advantage of a fitted parametric model, both Zipf–Mandelbrot and Gamma–Poisson produce estimates that are very close to those of the non-parametric estimators. Notably, Zipf–Mandelbrot has the largest bias magnitude (186,851) of any estimator evaluated, parametric or non-parametric, despite being the best-fitting distribution for the majority of subjects.

Neither of the two evaluated parametric estimators achieves a clear accuracy advantage over the evaluated non-parametric methods. Parametric and non-parametric estimators are comparatively indistinguishable in accuracy across evaluated synthetic parsers.

V. DISCUSSION

What the Zipf–Mandelbrot fit tells us about coverage discovery. Zipf–Mandelbrot provides the best fit for the large majority of subjects: six of the seven real-world programs (Table III) and two of the five synthetic parsers (Table IV). For the remaining subjects, Gamma–Poisson achieves the better fit instead: libxml2 among real-world programs, and ParserB, ParserD, and ParserE among synthetic parsers. The margins range from negligible (0.4% for ParserB) to substantial (7.2% for ParserE, comparable to Zipf–Mandelbrot's own margin of victory elsewhere). The Zipf–Mandelbrot plots for real-world programs (Figure 1) and parsers (Figure 2) are provided. Both Zipf–Mandelbrot and Gamma–Poisson are heavy-tailed

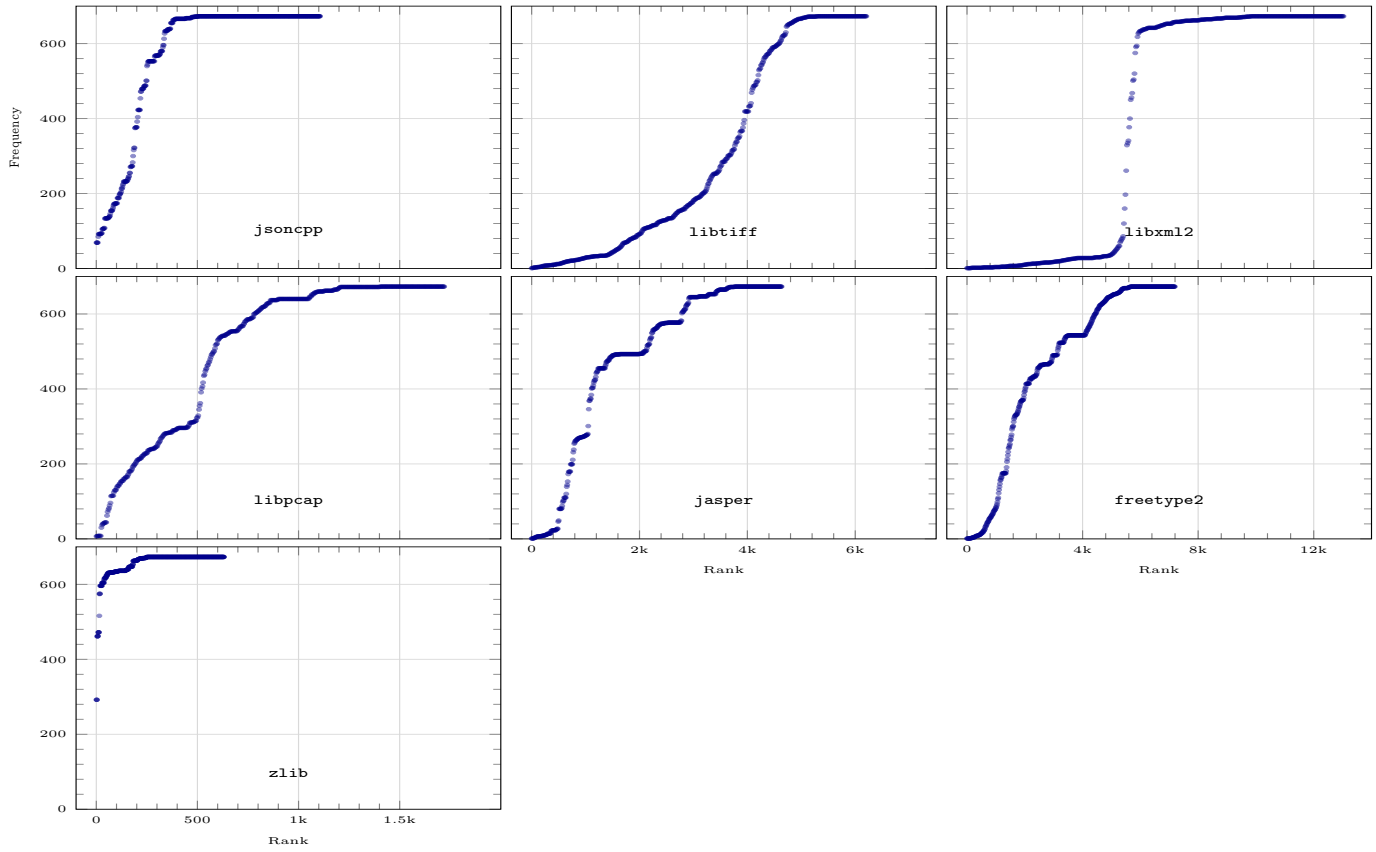

Fig. 1. The rank-frequency curve for real-world programs to allow us to understand the bi-modal distribution. For visualization, coverage elements are arranged in ascending order of their incidence counts across all sampling units. The x-axis is the rank. That is, if we arrange all coverage elements in the ascending order based on their occurrence in all sampling units, number $n = 1$ in X axis means that has the smallest occurrence count out of all coverage elements, and $n = 10$ in X axis means it has the 10th smallest occurrence. The y-axis is the frequency of occurrence. That is, How many elements are there has the nth smallest coverage reported.

alternatives to the homogeneous Poisson and Exponential baselines, which are consistently and substantially worse across every subject. This indicates that coverage discovery is strongly heterogeneous and heavy-tailed. It follows a power law, with a small fraction of easy-to-reach elements and a long tail of increasingly rare elements. This quantitatively confirms the thesis of Boehme [12]: fuzzing campaigns face exponentially diminishing returns because the remaining elements are much harder to cover, not merely because coverage is running out. That the identity of the best-fitting model varies by subject, rather than a single model dominating uniformly, is itself informative: it suggests that no single two-parameter heavy-tailed family fully captures coverage discovery.

Even for subjects where it wins, the Zipf–Mandelbrot fit is imperfect in a specific way. When examining the coverage frequency data on a linear scale, it exhibits a saturation plateau. For example, in the freetype2 subject, 37.7% of elements are observed in at least 90% of sampling units, while 9.3% of elements are observed as rare. At the same time, a large portion of elements are never observed. This suggest two distinct regions in the distribution, with a dense cluster at high frequency and a long sparse tail, separated by a clear structural break. A single smooth power-law cannot capture both regions simultaneously. It must either follow the plateau or the tail, but not both. Therefore, the Zipf–Mandelbrot curve does not align with the high-frequency region. Since the model treats a bimodal distribution as if it were unimodal, the fitted parameters are dominated by the large saturated cluster. As a result, unseen elements are not well captured. In the fitted curve, the sparse tail, where the unseen elements reside is poorly represented. Because this tail determines the number of undiscovered elements, all unimodal models struggle to produce accurate reachability estimates. Bridging the gap between model fit and estimation accuracy likely requires models that explicitly separate the saturated region from the sparse tail.

Distributional fit does not translate into estimation accuracy. The key finding of this study is that while Zipf–Mandelbrot provides the best characterization of coverage discovery for the majority of subjects, neither the Zipf–Mandelbrot- nor the Gamma–Poisson-based estimator achieves a measurable accuracy advantage over non-parametric methods. We postulate that the reason is that while these distributions characterize the coverage discovery process quite well, the estimators derived from them may not be optimal, in particular because they may not be robust to some or all assumptions in the underlying models. Taken together, our results show that goodness of fit

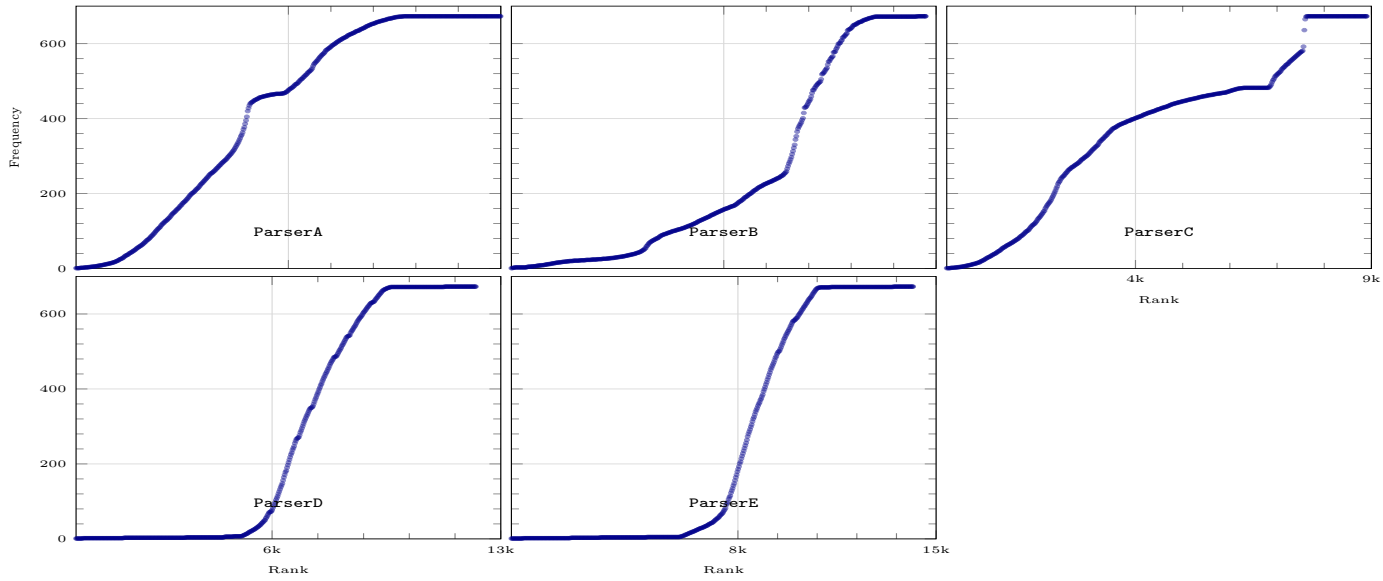

Fig. 2. The rank-frequency curve for synthetic programs to allow us to understand the bi-modal distribution. For visualization, coverage elements are arranged in ascending order of their incidence counts across all sampling units. The x-axis is the rank. That is, if we arrange all coverage elements in the ascending order based on their occurrence in all sampling units, number $n = 1$ in X axis means that has the smallest occurrence count out of all coverage elements, and $n = 10$ in X axis means it has the 10th smallest occurrence. The y-axis is the frequency of occurrence. That is, How many elements are there has the nth smallest coverage reported.

and reachability estimation are related but distinct problems. A model may explain the observed discovery process well while still providing limited leverage on the unseen portion of the coverage space, which is precisely the quantity required for maximum reachability estimation.

Why better fit does not imply recovery of the unseen tail. In all five simulated parsers, the estimated sum using the Zipf-Mandelbrot estimator is equal to the observed coverage given in Table VI. This means that the estimated rank–frequency distribution does not predict any further unseen items beyond the estimator's threshold. Although the model fits the distribution of incidences very well, reachability significantly depend on the sparse tail of the distribution.

Implications for benchmark design. The similar distributional behavior of synthetic parser benchmarks and real-world programs supports the use of parser-based programs as ground-truth evaluation subjects. This is practically important: evaluating estimator accuracy requires ground truth, which is unavailable for typical real-world software. The synthetic parsers produce programs whose coverage space is both fully enumerable and statistically representative, making them a principled substitute for the large, opaque programs that would otherwise be needed for credible evaluation.

Why coverage falls off a cliff before reaching a majority of the code. What remains to be explained is why both real-world programs and parsers appear to fall off a cliff in coverage without reaching a majority of their code. We note that this shared behavior is itself evidence that parsers are representative of real-world programs.

We have a hypothesis. Parsers have traditionally been among the most difficult subjects for fuzzers, which is precisely why there is a line of research on specification inference for parsers [22]–[24]. Our results suggest that the coverage space is potentially bi-modal, consisting of a small fraction of trivially covered lines followed by a much larger region of substantially harder-to-cover code. Under such a regime, neither parametric nor non-parametric estimators would accurately model coverage discovery until fuzzing has progressed significantly further than what current campaigns achieve. The AFL++ fuzzer is able to explore the program elements that are close to the provided seed corpus, but is unable to generalize further to the complete input domain within the time provided. Indeed this is what we see in our real-world programs (Figure 1) where you can distinguish the slowly decaying head corresponding to trivially covered blocks, followed by a knee representing when trivial coverage is exhausted, followed by the long decay of the tail representing harder to cover blocks. The initial decay and the knee is also visible in parsers (Figure 2).

VI. Related Work

Determining reachable coverage is a fundamental challenge in software testing and fuzzing. Recent work has investigated statistical approaches that leverage observed execution data [14]. A simple non-statistical baseline treats the observed coverage at the end of a fuzzing campaign as ground truth once it appears to plateau. However, apparent plateaus can be unreliable [5], and the exponential difficulty of discovering new behaviours means that observed coverage may still underestimate the true maximum [12]. For sufficiently small programs, reachable coverage may saturate within a practical time, allowing ground truth to be established empirically [14]; however, results from such programs may not generalize to complex real-world

TABLE VI
PER-SUBJECT ESTIMATOR COMPARISON FOR SYNTHETIC PARSER BENCHMARKS

	Estimator	Known Total	Observed Coverage	Estimated Max. Cov.	Rel. Error (%)
ParserA	Gamma–P.	104,952	9,995	10,011	90.46
	Zipf–M.			9,995	90.48
	Chao2			10,029	90.44
	Chao2_bc			10,028	90.45
	Jackknife1			10,077	90.40
	Chiu			9,996	90.48
	Lanumteang–Böhning			10,043	90.43
ParserB	Gamma–P.	194,852	19,502	19,568	89.96
	Zipf–M.			19,502	89.99
	Chao2			19,510	89.99
	Chao2_bc			19,510	89.99
	Jackknife1			19,542	89.97
	Chiu			19,502	89.99
	Lanumteang–Böhning			19,532	89.98
ParserC	Gamma–P.	136,455	8,897	8,905	93.47
	Zipf–M.			8,897	93.48
	Chao2			8,933	93.45
	Chao2_bc			8,932	93.45
	Jackknife1			8,977	93.42
	Chiu			8,898	93.48
	Lanumteang–Böhning			8,940	93.45
ParserD	Gamma–P.	231,510	12,236	12,308	94.68
	Zipf–M.			12,236	94.71
	Chao2			12,252	94.71
	Chao2_bc			12,252	94.71
	Jackknife1			12,438	94.63
	Chiu			12,236	94.71
	Lanumteang–Böhning			12,242	94.71
ParserE	Gamma–P.	331,301	14,186	14,331	95.67
	Zipf–M.			14,186	95.72
	Chao2			14,237	95.70
	Chao2_bc			14,237	95.70
	Jackknife1			14,599	95.59
	Chiu			14,186	95.72
	Lanumteang–Böhning			14,198	95.71

Columns show the known total blocks for each parser, the observed coverage (rounded to the nearest integer), the estimated maximum coverage (rounded to the nearest integer), and the relative error for Gamma–P. , Zipf–M. , and five non-parametric estimators. All five synthetic parsers with known-total ground truth (ParserA, ParserB, ParserC, ParserD, ParserE) are shown

software. Among statistical approaches, bootstrapping from empirical campaign data has been used to create a synthetic ground-truth asymptote for evaluating estimators when true reachable coverage is unknown [14]. Building on this statistical perspective, species richness estimators from biostatistics provide a theoretically grounded approach to estimating unseen program elements by treating coverage elements as species and test inputs as samples [5], [25].

The *STADS* framework [5], models software testing as a statistical sampling process. By drawing an analogy between program behaviors and biological species, STADS enables the application of ecological estimators to infer the total number of reachable program elements from partial observations.

Recent work [14] studies estimating of maximum reachable coverage and fuzzing saturation. Reachable coverage is formulated as a statistical estimation problem over an unknown search space, where estimates improve with more coverage observations. These estimates quantify testing completeness, remaining unexplored code, and fuzzing effectiveness.

Several studies examine adaptive bias and propose mitigation methods. Boehme et al. [26] study residual risk estimation, showing limitations of traditional estimators under adaptive sampling. Liyanage et al. [27] sutdy coverage-rate extrapolation to mitigate bias via simulating blackbox campaigns and regression-based prediction, improving long-term accuracy.

Recent work examines the reliability of statistical estimators

TABLE VII
ESTIMATOR PERFORMANCE (BIAS AND RELATIVE ERROR)

Estimator	Bias		Rel. Error (%)	
	Mean	SD	Mean	SD
Parametric				
Gamma–Poisson	-186789.49	86711.42	92.85	2.54
Zipf–Mandelbrot	-186850.80	86764.63	92.88	2.54
Non-Parametric				
Chao2	-186821.76	86759.39	92.86	2.55
Chao2_bc	-186822.19	86759.10	92.86	2.55
Chiu	-186850.49	86764.81	92.88	2.54
Jackknife1	**-186687.44**	**86627.30**	**92.80**	**2.51**
Lanumteang–Böhning	-186823.05	86780.84	92.86	2.56

in fuzzing contexts. Empirical evaluations indicate that widely used estimators can exhibit substantial bias and variability depending on the underlying distribution of coverage elements and program structure [8]. Furthermore, statistical reachability analysis has been proposed to better model heterogeneous discovery processes and overcome limitations of structure-agnostic estimation methods [28].

Accurate estimation depends on reliable coverage measurement and evaluation infrastructure. Tools such as FuzzTastic provide fine-grained, time-resolved coverage data and address biases in traditional approaches, enabling more robust statistical analysis of fuzzing progress [29]. Additionally, alternative coverage metrics and fuzzing strategies, such as n-gram coverage, improve exploration effectiveness and better capture program behavior [30].

While synthetic benchmarks have been proposed before for controlled evaluation, existing approaches such as maze-based programs [31] provide limited structural complexity and do not adequately reflect real-world software behavior.

Despite these advances, a key gap remains: all existing reachability estimators for fuzzing are either non-parametric or implicitly assume distributions without empirical validation. No prior work identifies the distribution governing coverage discovery, validates it against programs with known ground truth, or evaluates whether correct distributional assumptions improve estimation.

In this work, we directly address these open questions. We are the first to empirically characterize the distribution of coverage discovery across multiple real-world programs, to verify that synthetic parser benchmarks replicate the observed distribution, and to evaluate whether parametric estimators derived from the identified distribution outperform the best non-parametric alternatives. Our finding that distributional fit and estimation accuracy are largely decoupled shows that improving the model of the *observed* discovery process does not automatically improve prediction of the *unseen* portion, and points toward the structural gap that future estimators must close.

VII. THREATS TO VALIDITY

Internal validity. Fuzzing campaigns depend on factors such as seed selection, mutation strategies, and scheduling algorithms that may influence coverage discovery. Other fuzzers could exhibit different incidence distributions. To reduce variability, we perform multiple independent campaigns per subject and aggregate results across runs.

Finally, we mitigated machine-load effects by controlling load during experiments.

External validity. Our parsers provide a controlled setting for validating estimator accuracy. However, they may not fully capture the diversity and complexity of arbitrary software systems. All our experiments are conducted using AFL++, which primarily targets C/C++ programs. Therefore, their effectiveness may not generalize to other fuzzers or languages with different execution characteristics.

Construct validity. Our cross-domain conclusion is indirect: the argument is that if the best-fit distribution is the same in real-world and generated programs, then the same estimator should apply. In practice, this holds for the majority but not all of our subjects: Gamma–Poisson outperforms Zipf–Mandelbrot for one real-world program and three of five parsers. We acknowledge this threat.

VIII. CONCLUSION

This paper reports a negative result for the hypothesis that better distributional fit of the two evaluated parametric models necessarily improves reachable-coverage estimation in fuzzing.

Our results show that the Zipf–Mandelbrot distribution provides the best fit to coverage discovery data for the majority of subjects, across both real-world programs and synthetic benchmarks, with Gamma–Poisson the closest competitor where it does not. This suggests that fuzzing is governed by strongly heterogeneous discovery rates, with many hard-to-reach elements forming a long tail. Parser-based synthetic programs exhibit the same distributional behavior as real-world targets, supporting their use as controlled benchmarks with known ground truth. Despite this fit advantage, however, neither the Zipf–Mandelbrot- nor the Gamma–Poisson-based estimator offers a clear prediction advantage over the best non-parametric estimators. Hence, the central hypothesis motivating this study does not hold. This highlights a fundamental distinction: a model can characterize observed data well without improving extrapolation to the unseen tail.

Our findings concern these two parametric estimators and should not be generalized to parametric estimation as a whole; mixture, piecewise, time-dependent, and structure-aware approaches remain unexplored.

Translating improved distributional fit into better reachability estimates remains an open problem, likely requiring structural and temporal properties of the fuzzing process.

IX. DATA AVAILABILITY

The parser generator and analysis pipeline are available at

https://github.com/Nelum123/issre2026-rene.git

https://doi.org/10.5281/zenodo.22004432

REFERENCES

[1] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as Markov chain," in *ACM Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 1032–1043.

[2] J. Fell, "A review of fuzzing tools and methods," *PenTest*, 2017.

[3] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE TSE*, vol. 39, no. 2, pp. 276–291, 2013.

[4] J. R. Horgan, S. London, and M. R. Lyu, "Achieving software quality with testing coverage measures," *Computer*, vol. 27, no. 9, 1994.

[5] M. Böhme, "STADS: Software testing as species discovery," *ACM TOSEM*, vol. 27, no. 2, pp. 7:1–7:52, 2018.

[6] K. Zhu, Y. Lu, H. Huang, L. Yu, and J. Zhao, "Constructing more complete control flow graphs utilizing directed gray-box fuzzing," *Appl. Sci.*, vol. 11, no. 3, p. 1351, 2021.

[7] A. Chao, C.-H. Chiu *et al.*, "Species richness: Estimation and comparison," *Wiley StatsRef: Stat. Ref. Online*, vol. 1, p. 26, 2016.

[8] K. Kuznetsov, A. Gambi, S. Dhiddi, J. Hess, and R. Gopinath, "Empirical evaluation of frequency based statistical models for estimating killable mutants," in *ACM/IEEE ESEM*, 2024, pp. 61–71.

[9] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Pearson, 2007.

[10] D. Liyanage, N. Attanayake, Z. Luo, and R. Gopinath, "Assessing reliability of statistical maximum coverage estimators in fuzzing," *arXiv preprint arXiv:2507.17093*, 2025.

[11] J. E. Schmitz and S. Rahmann, "A comprehensive review and evaluation of species richness estimation," *Briefings in Bioinformatics*, 2025.

[12] M. Böhme and B. Falk, "Fuzzing: On the exponential cost of vulnerability discovery," in *ACM ESEC/FSE*, 2020, pp. 713–724.

[13] C.-H. Chiu, "A more reliable species richness estimator based on the Gamma–Poisson model," *PeerJ*, vol. 11, p. e14540, 2023.

[14] D. Liyanage, M. Böhme, C. Tantithamthavorn, and S. Lipp, "Reachable coverage: Estimating saturation in fuzzing," in *IEEE/ACM ICSE*, 2023.

[15] A. Chao, "Estimating the population size for capture-recapture data with unequal catchability," *Biometrics*, pp. 783–791, 1987.

[16] A. Chao, R. L. Chazdon, R. K. Colwell, and T.-J. Shen, "A new statistical approach for assessing similarity of species composition with incidence and abundance data," *Ecol. Lett.*, vol. 8, no. 2, pp. 148–159, 2005.

[17] K. P. Burnham and W. S. Overton, "Estimation of the size of a closed population when capture probabilities vary among animals," *Biometrika*, pp. 625–633, 1978.

[18] K. Lanumteang and D. Böhning, "An extension of Chao's estimator of population size based on the first three capture frequency counts," *Comput. Stat. Data Anal.*, vol. 55, no. 7, pp. 2302–2311, 2011.

[19] I. J. Good and G. H. Toulmin, "The number of new species, and the increase in population coverage, when a sample is increased," *Biometrika*, 1956.

[20] V. R. Pratt, "Top down operator precedence," in *ACM SIGACT-SIGPLAN SPPL*, 1973, pp. 41–51.

[21] M. van Hauser *et al.*, "AFLplusplus," 2025. [Online]. Available: https://aflplus.plus/

[22] L. Bettscheider and A. Zeller, "Inferring input grammars from code with symbolic parsing," *ACM TOSEM*, 2025.

[23] M. Schröder and J. Cito, "Static inference of regular grammars for ad hoc parsers," *Proc. ACM Program. Lang.*, no. OOPSLA2, 2025.

[24] R. Gopinath, B. Mathis, and A. Zeller, "Mining input grammars from dynamic control flow," in *ACM ESEC/FSE*, 2020, pp. 172–183.

[25] A. Chao and R. K. Colwell, "Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling," *SORT*, 2017.

[26] M. Böhme, D. Liyanage, and V. Wüstholz, "Estimating residual risk in greybox fuzzing," in *ACM ESEC/FSE*, 2021, pp. 230–241.

[27] D. Liyanage, S. Lee, C. Tantithamthavorn, and M. Böhme, "Extrapolating coverage rate in greybox fuzzing," in *IEEE/ACM ICSE*, 2024.

[28] S. Lee and M. Böhme, "Statistical reachability analysis," in *ACM ESEC/FSE*, 2023.

[29] S. Lipp, D. Elsner, T. Hutzelmann, S. Banescu, A. Pretschner, and M. Böhme, "FuzzTastic: A fine-grained, fuzzer-agnostic coverage analyzer," in *IEEE/ACM ICSE Companion*, 2022, pp. 75–79.

[30] X. Peng, P. Jia, X. Fan, and J. Liu, "ENZZ: Effective N-gram coverage assisted fuzzing with nearest neighboring branch estimation," *Inf. Softw. Technol.*, vol. 177, p. 107582, 2025.

[31] H. Lee, S. Kim, and S. K. Cha, "Fuzzle: Making a puzzle for fuzzers," in *IEEE/ACM ASE*, 2022, pp. 1–12.