CovCast: When Will Fuzzing Stop Paying Off? A Specialized Tabular Foundation Model for Coverage-Rate Forecasting

AMINE LBATH[1,2], [1]National Institute of Standards and Technology, USA and [2]Université Grenoble Alpes, CNRS, LIG, France

Fuzzing finds security vulnerabilities by executing a program on millions of randomly generated and mutated inputs, and a single campaign often runs for days on many machines. As fewer parts of the program remain unexplored, the probability of finding new bugs decreases, and practitioners must decide when to stop a campaign. This decision requires forecasting the coverage rate of a campaign, the number of new code elements it will cover per unit of time in the coming hours or days. Because fuzzing is random, coverage grows irregularly, alternating between long plateaus and sudden increases. Existing methods therefore fit statistical estimators to each campaign separately, and require per-element execution counts that most fuzzing setups do not record.

We introduce CovCast (**Cov**erage fore**Cast**er), a novel tabular foundation model specialized for forecasting fuzzing coverage. CovCast is pretrained once on synthetic fuzzing campaigns generated by a simulator, and forecasts the coverage rate of any new campaign from the coverage observed so far, without any re-training. On the FuzzTastic benchmark of 133 campaigns on five programs, CovCast reduces the median forecast error of the prior state of the art by 17%, and by 39% for forecasts made on the second and third days of a campaign, when stopping saves the most compute. Because it only needs the coverage over time that every fuzzer reports, CovCast also applies where the prior state of the art cannot. On 1,833 FuzzBench campaigns of 11 fuzzers, it reduces the median error of the best general-purpose foundation model by 23% and is the most accurate method for every fuzzer and every program. These results suggest that knowledge about fuzzing can be encoded in pretraining data rather than in estimators fitted to each campaign.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: greybox fuzzing, code coverage, coverage rate, stopping criterion, tabular foundation models, prior-data fitted networks, in-context learning

1 Introduction

Fuzzing remains one of the most effective techniques for finding security vulnerabilities [23]. A fuzzer executes a program on a large number of randomly generated inputs, and mutates the inputs that exercise new code to explore the program further. Google's OSS-Fuzz continuously fuzzes more than a thousand open-source projects [30], and a single FuzzBench experiment runs thousands of campaigns lasting one to seven days [24]. As a campaign progresses, the rate at which the fuzzer covers new code declines, and practitioners must decide when to stop it. Stopping a campaign too early may leave vulnerabilities undetected. Continuing a campaign that no longer covers new code wastes computing resources.

Such decisions require a forecast of the *coverage rate*, the number of new code elements that a campaign covers per unit of time, at future points of the campaign. With such a forecast, a campaign can be stopped once its predicted rate falls below a threshold, and machines can be moved to the campaigns that are predicted to keep finding new code [22]. Forecasting the coverage rate is inherently challenging because coverage grows in bursts. A single input that reaches a new function can cover many of its basic blocks at once, and two such bursts can be separated by hours without any new coverage.

Author's Contact Information: Amine Lbath[1,2], [1]National Institute of Standards and Technology, Gaithersburg, MD, USA and [2]Université Grenoble Alpes, CNRS, LIG, Grenoble, France.

Table 1. Forecasting the coverage rate of a real AFL++ campaign on freetype2 after 37.5 hours. Each row gives the state of the campaign at a time t, a later time θ, and the coverage rate at θ, measured as the number of new blocks covered per 15 minutes. Rows from the campaign's past are known; the rates of future rows must be forecast. CovCast reads the known rows as context.

Time t	Blocks covered	New blocks, last 4 h	Later time θ	Rate at θ	(true rate)
20 h	6,186	68	25 h	21.11	
25 h	6,308	120	30 h	2.93	
30 h	6,438	43	36 h	0.18	
… *up to 2,000 rows from the campaign's first 37.5 hours*					
37.5 h	6,466	1	56.5 h	?	5.96
37.5 h	6,466	1	75 h	?	0.17
37.5 h	6,466	1	112.5 h	?	0.08

As an example, consider a campaign fuzzing the font library freetype2 with the greybox fuzzer AFL++ [8] (Table 1). After 37.5 hours, the campaign has covered 6,466 basic blocks, but only one new block in the last four hours. Extrapolating this trend would suggest that the campaign is exhausted. Nineteen hours later, however, it covers about 6 new blocks of code every 15 minutes, before slowing down to 0.17 new blocks every 15 minutes after three days. Deciding whether to continue the campaign requires forecasting this evolution at 37.5 hours.

Prior work views fuzzing as sampling "species", the code elements, from a population, and uses species-richness estimators from ecology to predict how many new species further sampling will reveal [4, 7]. These estimators assume that inputs are sampled independently. Greybox fuzzers, however, deliberately mutate the inputs that covered new code, so that what they sample depends on what they have already found. This *adaptive bias* makes the estimators unreliable [5, 21]. The current state of the art, the extrapolator of Liyanage et al. [22], corrects for adaptive bias by simulating many independent-sampling campaigns from the collected coverage data, estimating their coverage rates, and extrapolating these estimates with a regression. It remains a statistical procedure fitted separately to each campaign from that campaign's data, and it requires per-element hit counts, which most fuzzing setups do not record.

We propose to forecast the coverage rate with a model that has learned how fuzzing campaigns behave. Tabular foundation models [13, 26] make this possible without training a model for each campaign. These transformer-based models are pretrained on millions of synthetic tables to predict the target column of a new table from a few labeled rows given as context. At inference time, they leverage the context in a single forward pass, without updating any of their parameters. Because pretraining teaches them to approximate Bayesian inference [26], they are particularly accurate when only a few labeled rows are available [13]. They predict a full probability distribution of the target, which quantifies the uncertainty of each prediction. General-purpose models such as TabPFN are pretrained on general synthetic tables, and therefore have no specific knowledge of how fuzzers cover code.

We introduce CovCast (**Cov**erage fore**Cast**er), a novel specialized tabular foundation model for forecasting fuzzing coverage. Instead of general tables, CovCast is pretrained only on synthetic fuzzing campaigns generated by a simulator. The simulator encodes four properties of how greybox fuzzers cover code. Most code is reached early, the remaining code is reached in bursts, code becomes reachable only once the code guarding it is covered, and the throughput of the fuzzer decreases over time. Since the simulator can generate any number of campaigns, pretraining is not limited by the scarcity of real campaign data. To forecast a new campaign, CovCast turns the

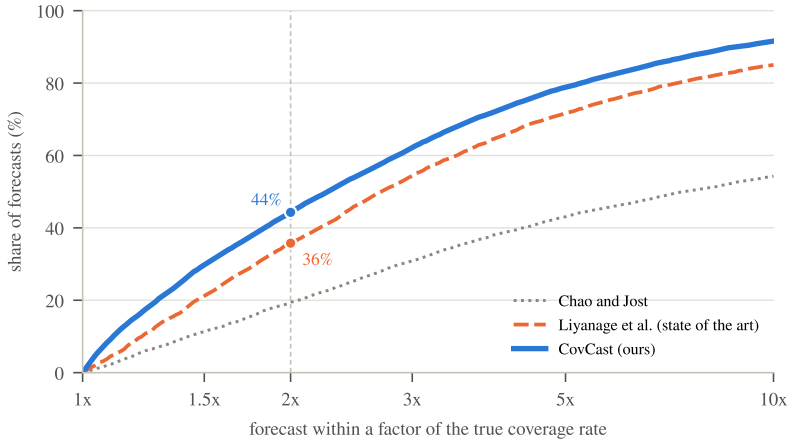


Fig. 1. Share of the 4,389 forecasts of the FuzzTastic benchmark that are within a given factor of the true coverage rate (higher is better). CovCast is more accurate than the state of the art at every tolerance. For example, 44% of its forecasts are within a factor of two of the true rate, against 36% for the extrapolator of Liyanage et al. [22]. CovCast is averaged over four pretraining runs.

coverage observed so far into a table like Table 1, provides it to the model as context, and obtains the forecast coverage rate in one forward pass, without any re-training.

We evaluate CovCast on two benchmarks. On the FuzzTastic benchmark of Liyanage et al., which contains 133 AFL++ campaigns of up to eight days on five programs, CovCast forecasts the coverage rate with a 17% lower median error than the prior state of the art (Fig. 1). Its advantage is largest for forecasts made on the second and third days of a campaign, when stopping saves the most compute, with a 39% lower median error. Stopping campaigns based on its forecasts saves almost four times as much compute as a fixed time budget for the same loss of coverage. Because CovCast only needs the coverage over time that every fuzzer reports, it also applies to campaigns for which the prior state of the art lacks the required data. On 1,833 FuzzBench campaigns of 11 fuzzers on 22 programs, it reduces the median error of the best general-purpose foundation model by 23% and is the most accurate method for every fuzzer and every program.

This paper makes four contributions: (i) we introduce CovCast, a novel specialized tabular foundation model and to our knowledge the first learned forecaster of fuzzing coverage, pretrained only on synthetic campaigns and applied to real campaigns without re-training; (ii) we design a simulator of greybox fuzzing campaigns and a representation of a campaign's coverage as context for a tabular foundation model; (iii) we evaluate CovCast on the benchmark and metric of the state of the art and on 1,833 FuzzBench campaigns, against statistical extrapolators, persistence baselines, general-purpose tabular and time-series foundation models, and models learned from real campaigns; and (iv) we release the simulator, the pretrained model and all forecasts.

2 Background and Related Work

We first define coverage and the coverage rate, then review how the coverage rate has been estimated, and finally introduce the tabular foundation models we build on.

Coverage and coverage rate. The *coverage* $S(t)$ of a campaign is the number of code elements, such as basic blocks, branches or edges, executed by at least one input generated up to time t. Greybox fuzzers aim to maximize coverage, and coverage is the standard measure of fuzzing progress. When fuzzers are compared, higher coverage is strongly correlated with more bugs

found [6]. The *coverage rate* $U(t)$ is the number of new elements covered per unit of time. Following Liyanage et al. [22], we measure time in 15-minute steps and define the true rate as a moving average of the coverage increments over ±1 hour. We forecast the coverage rate rather than the coverage itself, because most coverage is reached early. In the FuzzBench 23-hour experiment [24] (4,840 campaigns), a campaign reaches a median of 96.4% of its final coverage within the first hour, so that predicting no further growth is only 3.6% off the final coverage. The decision to stop, however, depends on the remaining growth, which an extrapolation of the first hour's coverage curve misses by a median of 89%. Table 1 illustrates the bursts and plateaus that make this growth difficult to forecast.

Estimating future coverage. The STADS framework [4] views fuzzing as species discovery. Good–Turing [10] estimates the probability that the next input covers a new element, species richness estimators estimate how many elements are reachable at all [21], and the extrapolator of Chao and Jost [7] predicts how many new elements further inputs will cover. All of them assume independent sampling, and adaptive bias makes them underestimate how much a greybox campaign has left to find [5, 21]. The extrapolator of Liyanage et al. [22] is the state of the art for forecasting the coverage rate. It builds independent-sampling versions of the greybox campaign from its per-element coverage data, estimates their rates, and extrapolates these estimates with a regression on time in log-log space. In its authors' evaluation, it reduces the bias of the extrapolator of Chao and Jost by at least an order of magnitude on four of five programs. More recent estimators [17, 18] improve the estimate of the probability that the *next* input covers something new, a different quantity from the rate at a future time, and also require per-input coverage data. Green Fuzzing [20] stops a campaign once the coverage of functions predicted to be vulnerable saturates. All of these methods are fitted to each campaign separately, from that campaign's data alone, and most require per-element coverage data.

Software reliability growth. Reliability growth models fit parametric curves to the failures observed during testing [9]. DSC-SRGM [14] pretrains a recurrent network on synthetic curves from such models and forecasts real failure curves. Like CovCast, it relies on synthetic pretraining. It produces point forecasts from a mixture of parametric curves, while CovCast predicts a full distribution by in-context learning, with a simulator of greybox fuzzing.

Tabular foundation models are transformers pretrained on millions of synthetic tables, which predict the target column of a new table from a few labeled rows given as context, without further training [12, 13, 26]. Because they are trained to predict across many tables drawn from a known distribution, they learn to approximate Bayesian prediction under that distribution [26]. TabPFN, TabICL [29] and Nori [31] are pretrained on general synthetic tables and lead general tabular benchmarks. Models of the same kind with specialized pretraining distributions have been built for extrapolating the learning curves of machine-learning models [1] and for Bayesian optimization [25], and time-series foundation models such as Chronos [3] forecast univariate series zero-shot. To our knowledge, none of these models has been specialized for fuzzing.

3 The CovCast Approach

CovCast has three elements (Fig. 2): ① a simulator of greybox fuzzing campaigns, ② a representation of a campaign's coverage history as a table, and ③ a model pretrained on tables built from simulated campaigns and applied, without re-training, to real ones.

3.1 ① Simulating Greybox Fuzzing Campaigns

The simulator generates random coverage curves that behave like those of real greybox campaigns (Algorithm 1). It encodes four observations about how greybox fuzzers cover code.

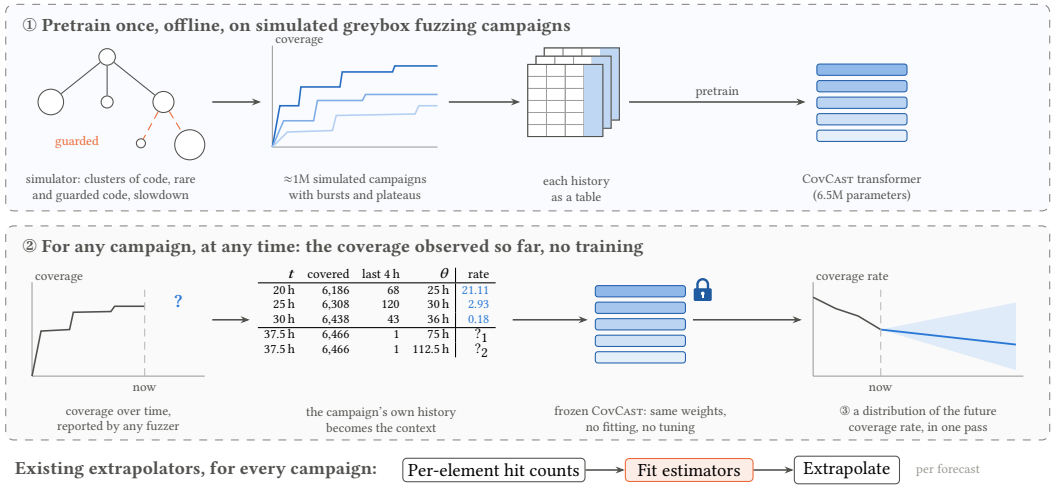


Fig. 2. Overview of CovCast. ① A transformer-based model is pretrained once, offline, on about a million simulated greybox fuzzing campaigns. ② For any campaign and at any time, the coverage observed so far becomes the context of the frozen model, which ③ forecasts the distribution of the future coverage rate in one forward pass. Existing extrapolators instead fit statistical estimators to the per-element hit counts of each campaign (bottom).

Most code is easy to reach; some is very hard. Code elements differ by orders of magnitude in how often random inputs reach them. Each group of elements receives a hit rate drawn from a log-normal distribution, so that most code is covered in the first hours and a long tail is covered late or never.

Coverage arrives in bursts. An input that reaches a new function covers many of its blocks at once. Elements are grouped into 30 to 1,000 clusters of varying size, and a cluster is covered all at once. This produces the bursts and long plateaus illustrated in Table 1.

Code is guarded by other code. Code behind a branch becomes reachable only once the branch is covered. Greybox fuzzers exploit this dependency, which also makes their sampling adaptive. Clusters are arranged in up to eight levels, and the clock of a cluster starts only when a cluster of the level above is covered.

Fuzzers slow down. The number of executions per second decreases as the corpus grows. Simulated effort is converted into wall-clock time through a throughput that decays as $(1 + \theta/\tau)^{-\alpha}$, with random τ and α.

Every simulated campaign is observed in the same way as a real one, every 15 minutes from 12 hours to 8 days. The simulator uses no data from real campaigns, and every campaign is generated from randomly drawn parameters. Their ranges are deliberately broad, covering programs of a few hundred to tens of thousands of code elements and campaigns that saturate within hours or keep growing for days, so that the model can be applied to campaigns of any fuzzer and program.

3.2 ② The Campaign History as Context

A tabular foundation model predicts the target column of query rows from labeled context rows. We build both from the campaign itself, as in Table 1. Each row describes the campaign as observed up to a time t, together with a later time θ; its label is the logarithm of the coverage rate at θ. The description consists of nine numbers: the coverage rates over the last 1, 4 and 12 hours, the slope and intercept of a power law fitted to the rate over the second half of the campaign so far, the

Algorithm 1: Sampling one campaign from the simulator

$K \sim \mathrm{LogUniform}(30, 1000)$ clusters; sizes $s_j = \lceil e^{z_j} \rceil$, $z_j \sim \mathcal{N}(\log \bar{s}, \varsigma)$, $\bar{s} \sim \mathcal{U}(1, 20)$, $\varsigma \sim \mathcal{U}(0.3, 1.5)$;
levels $L \sim \mathcal{U}\{1..8\}$; depth $d_j \sim \min(\mathrm{Geom}(g), L) - 1$, $g \sim \mathcal{U}(0.15, 0.9)$;
hit rates $\log \lambda_j \sim \mathcal{N}(-\mu, \sigma)$, $\mu \sim \mathcal{U}(\log 0.2, \log 10)$, $\sigma \sim \mathcal{U}(1, 4)$;
efforts $e_j \sim \mathrm{Exp}(\lambda_j)$;
for *level* $d = 1..L - 1$ **do**
  $e_j \leftarrow e_j + e_{\mathrm{parent}(j)}$ for clusters j at depth d, parent at depth $d - 1$;
wall-clock $\theta_j \leftarrow \tau\big((1 + e_j(1 - \alpha)/\tau)^{1/(1-\alpha)} - 1\big)$, $\tau \sim \mathrm{LogUniform}(1, 500)$, $\alpha \sim \mathcal{U}(0, 0.9)$;
return $S(\theta) = \sum_j s_j \, [\theta_j \le \theta]$ *for* $\theta = 12\,h, 12.25\,h, \ldots, 8\,d$

extrapolation of that power law to θ, the elapsed time t, the coverage $S(t)$, and the ratio θ/t (all on a log scale). Context rows use earlier times t, one per hour from 15 hours on, and later times θ whose rate is already known when the forecast is made at θ_0. Query rows use $t = \theta_0$ and the future times to forecast. The context thus shows the model how this campaign's rate evolved after moments that looked like the present one, and the query asks for the same relation one step further.

The rows are strictly causal. The true rate at θ averages coverage increments up to one hour after θ, so context labels stop one hour before θ_0. The moving average also bridges long stalls by extending its window to the next coverage increment, so we compute all context rows from the curve truncated at θ_0, which prevents a stall from being bridged by an increment observed after the forecast. We use at most 2,000 context rows. If a campaign has not covered anything new since 15 hours, the context is empty and every learned method falls back to the power-law extrapolation.

3.3 ③ Pretraining and Forecasting

CovCast uses the architecture of tabular foundation models: a transformer that alternates attention across the nine values of a row and attention across rows, in which query rows attend only to context rows. Its output is a probability distribution over 5,000 bins of the (standardized) log rate. We pretrain it from scratch on tables built from simulated campaigns (Algorithm 1) as in Section 3.2, at a random forecast time between 25 and 125 hours. Every batch is freshly simulated, so the model never sees the same campaign twice. The model has 10 layers, width 192 and 6.5 million parameters. We pretrain it with AdamW on four A100 GPUs, with a learning rate of $3 \cdot 10^{-4}$, 1,000 warm-up steps, and cosine decay over 8,000 steps. We select the checkpoint on 200 FuzzBench campaigns that are not part of any evaluation (Section 4); the selected checkpoint was trained for 4,000 steps, about 70 minutes, on roughly a million simulated campaigns. To forecast, CovCast returns the median of the predicted distribution, averaged over eight random orderings of the nine columns.

Use in practice. During a campaign, a forecast can be requested at any time from the coverage over time that every fuzzer reports. Each forecast builds a new table and calls the pretrained model once. Since no model is trained for a specific program, fuzzer or campaign, CovCast requires no tuning or maintenance. Because the model predicts a distribution, a stopping policy can also take the uncertainty of the forecast into account.

4 Evaluation

We evaluate CovCast through five research questions:

> **RQ1:** How accurately does CovCast forecast the coverage rate, compared with the state of the art and with general-purpose foundation models?

RQ2: How robust is CovCast across programs, fuzzers and stages of a campaign?
RQ3: Why does CovCast work, i.e., what matters in its pretraining data?
RQ4: Do the forecasts of CovCast support stopping decisions, and are they well calibrated?
RQ5: How expensive is CovCast?

4.1 Experimental Setup

Benchmarks. We use two benchmarks. The FuzzTastic benchmark of Liyanage et al. [22] contains AFL++ [8] campaigns on freetype2, gif2png, jasper, jsoncpp and readelf, with coverage measured in basic blocks by FuzzTastic [19] every 15 minutes for up to eight days. After removing the outlier runs excluded by the authors, 133 campaigns remain. Following the original study, forecasts are made every 12.5 hours, at elapsed times θ_0 from 25 to 125 hours. Horizons range from 0.5 to 3 times the elapsed time and must end within the campaign. This yields 4,389 forecast windows. To keep the cost of the general-purpose models manageable, every method is scored on every second 15-minute step of each window. The FuzzBench 7-day experiment [24] contains 2,420 campaigns of 11 fuzzers (AFL, AFLFast, AFL++, AFLSmart, Eclipser, Entropic, FairFuzz, honggfuzz, LAF-Intel, libFuzzer and MOpt) on 22 programs, with coverage measured in edges. We forecast the 2,220 campaigns that were not used to select the checkpoint, after 25, 50 and 75 hours, for horizons up to the elapsed time; 1,833 campaigns have at least one window in which the true rate is not zero (5,498 windows).

Metric. We use the metric of the original study: the *bias* of a window is the average, over its steps, of the difference between the logarithm of the forecast rate and the logarithm of the true rate. We report the median and mean absolute bias over windows. An absolute bias of 0.69 means that a forecast is off by a factor of two on average. Windows of the same campaign overlap and are therefore not independent, so we compare methods at the level of campaigns. We report the relative reduction of the median absolute bias with a 95% confidence interval obtained by resampling campaigns (2,000 bootstrap samples), the share of campaigns in which CovCast has the lower mean absolute bias, and the p-value of a paired Wilcoxon signed-rank test over campaign means. We base our conclusions on the confidence intervals; p-values are not adjusted for multiple comparisons.

Methods. We compare CovCast with the extrapolators of Liyanage et al. and of Chao and Jost [7], using the predictions published by Liyanage et al., so that both are evaluated as by their authors. We also compare it with three simple extrapolations of the campaign's own history: a power law fitted to its rate (the last of our nine columns), and the rate of the last 4 or 12 hours carried forward. We further compare with three general-purpose tabular foundation models, TabPFN v2 [13], TabICL [29] and Nori [31], which receive the same tables as CovCast and also return the median of their predicted distribution, and with the time-series foundation model Chronos-Bolt [3], which forecasts the campaign's (causal, smoothed) rate history zero-shot. All methods use only the observed coverage of the campaign being forecast. FuzzBench publishes only the amount of code covered over time, whereas the extrapolators of Liyanage et al. and of Chao and Jost estimate from per-element hit counts; replaying the campaigns with hit-count instrumentation would make them applicable, at substantial cost. On FuzzBench, we therefore compare CovCast with TabPFN, Chronos-Bolt, the two persistence baselines and the power law, which corresponds to the regression step of Liyanage et al. without its bias correction.

Protocol. No part of CovCast is fitted to the benchmark. The architecture and pretraining recipe were fixed in advance, the pretraining data are generated by the simulator alone, without any real campaign, and the checkpoint was selected on 200 randomly chosen campaigns of the FuzzBench 7-day experiment [24], which uses other fuzzers, other programs and edge coverage. Since pretraining is stochastic, we run it four times with different random seeds and the same

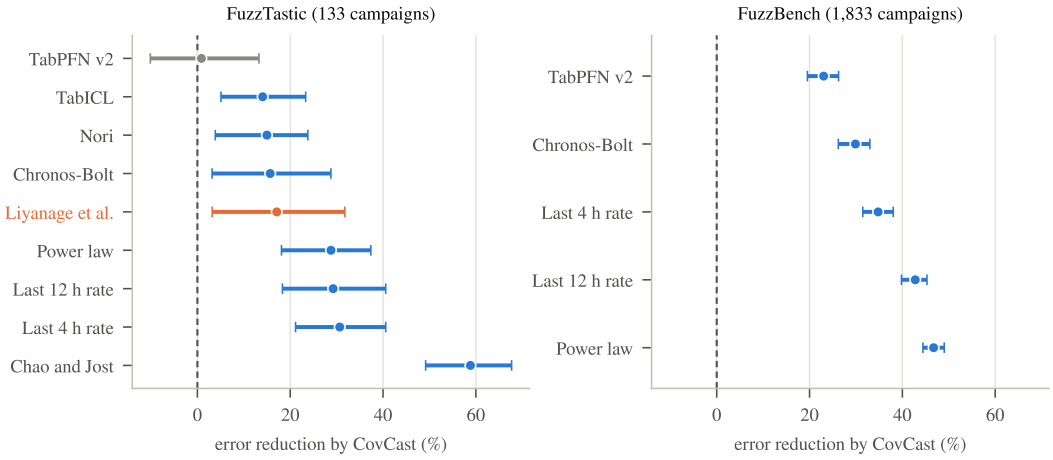


Fig. 3. Reduction of the median absolute bias by CovCast (higher is better) against every baseline, with 95% confidence intervals from a bootstrap over campaigns. CovCast is more accurate than every baseline on FuzzBench and, on FuzzTastic, than all baselines except TabPFN, which it matches overall and outperforms early in campaigns.

Table 2. Forecast accuracy on the FuzzTastic benchmark (4,389 windows of 133 campaigns), with CovCast averaged over four pretraining runs. Median and mean absolute bias: lower is better. Reduction (higher is better): relative reduction of the median absolute bias by CovCast, with a 95% confidence interval from a bootstrap over campaigns. Won: share of campaigns in which CovCast has the lower mean absolute bias; p: Wilcoxon signed-rank test over campaign means.

Method	Median |bias|	Mean |bias|	Reduction [95% CI]	Won	p
Pretrained on simulated campaigns					
CovCast	**0.824**	**1.024**	–	–	–
General-purpose tabular foundation models, same tables					
TabPFN v2	0.831	1.051	1% [−10, 13]	57%	0.07
TabICL	0.959	1.218	14% [5, 23]	67%	$3 \cdot 10^{-7}$
Nori	0.969	1.220	15% [4, 24]	68%	10^{-6}
Statistical extrapolators					
Liyanage et al. [22]	0.994	1.283	17% [3, 32]	50%	0.04
Power law	1.157	1.620	29% [18, 37]	83%	$6 \cdot 10^{-14}$
Chao and Jost [7]	2.002	3.402	59% [49, 68]	85%	$8 \cdot 10^{-18}$
Time-series foundation model, zero-shot on the rate history					
Chronos-Bolt [3]	0.977	1.162	16% [3, 29]	60%	0.01
Persistence: the recent rate carried forward					
Last 12 hours	1.165	1.341	29% [18, 41]	74%	$9 \cdot 10^{-8}$
Last 4 hours	1.189	1.317	31% [21, 41]	71%	$3 \cdot 10^{-7}$

schedule, stopping each run at the selected step. We report the error of CovCast averaged over the four runs, which is the expected error of one pretraining run.

Table 3. Forecast accuracy on 1,833 held-out FuzzBench 7-day campaigns of 11 fuzzers on 22 programs, with edge coverage (5,498 windows), with CovCast averaged over four pretraining runs. Median and mean absolute bias: lower is better. Reduction and Won (higher is better) as in Table 2, over campaigns; Programs: number of programs on which CovCast has the lower median absolute bias.

Method	Median |bias|	Mean |bias|	Reduction [95% CI]	Won	Programs
CovCast	**0.715**	**0.938**	–	–	–
TabPFN v2	0.929	1.105	23% [20, 26]	65%	22/22
Chronos-Bolt	1.020	1.182	30% [26, 33]	66%	22/22
Last 4 hours	1.097	1.304	35% [31, 38]	68%	22/22
Last 12 hours	1.250	1.423	43% [40, 45]	73%	22/22
Power law	1.343	1.696	47% [44, 49]	84%	22/22

4.2 RQ1: Accuracy

On the FuzzTastic benchmark (Table 2 and Fig. 3), CovCast reduces the median forecast error of the extrapolator of Liyanage et al., the state of the art, by 17% (0.82 against 0.99, 95% CI [3, 32]), and that of Chronos-Bolt, TabICL and Nori by 14 to 16%. Its gains are also much larger than its losses. On the campaigns where CovCast is more accurate than Liyanage et al., its mean error is lower by 0.87 on average, and on the others it is higher by only 0.36. CovCast is also less often far off. Its forecasts are off by more than a factor of five in 21% of the windows, against 28% for Liyanage et al. CovCast is as accurate as TabPFN overall (0.82 against 0.83), but, as RQ2 shows, considerably more accurate early in a campaign. Simple extrapolations of the campaign's own history, such as carrying the recent rate forward or fitting a power law, have a 40 to 44% higher median error than CovCast. Chronos-Bolt, although pretrained on a large corpus of real and synthetic time series, is only as accurate as the extrapolator of Liyanage et al., which indicates that general knowledge of time series does not capture how fuzzing campaigns evolve.

On the 1,833 held-out FuzzBench campaigns (Table 3), which come from 11 fuzzers and measure coverage in edges, CovCast is clearly the most accurate method. It reduces the median error of TabPFN by 23%, that of Chronos-Bolt by 30%, and that of the persistence baselines and the power law by 35 to 47%, with narrow confidence intervals, and it is more accurate than every baseline on each of the 22 programs. These are campaigns for which the extrapolator of Liyanage et al. cannot produce a forecast at all, since FuzzBench does not record per-element hit counts.

Answer to RQ1: CovCast reduces the median error of the state-of-the-art extrapolator by 17% on FuzzTastic, and is the most accurate forecaster on FuzzBench, with a 23% lower median error than the best general-purpose foundation model and a lower error than every baseline on all 22 programs.

4.3 RQ2: Robustness Across Programs, Fuzzers and Campaign Stages

Programs. A practitioner usually does not know in advance which forecasting method suits a program, so a method must not fail on any of them. On the FuzzTastic benchmark, CovCast is the most robust method (Table 4), with a median error of at most 1.03 on every program. The extrapolator of Liyanage et al. is off by a factor of almost six on gif2png and jsoncpp (errors of 1.76 and 1.78), and TabPFN reaches 1.33 on jasper. CovCast is significantly more accurate than Liyanage et al. on gif2png and jsoncpp, where it is more accurate in 81% and 100% of the campaigns, and significantly more accurate than TabPFN on jasper and readelf. On FuzzBench, CovCast has the lowest median error on each of the 22 programs, against every baseline.

Table 4. Median absolute bias per program (lower is better; bold: lowest), with CovCast averaged over four pretraining runs, and, for CovCast against Liyanage et al. and against TabPFN, the share of campaigns in which CovCast is more accurate (higher is better), with the p-value of a Wilcoxon test over campaigns.

Program (campaigns)	Liyanage et al.	TabPFN	CovCast	Won vs. Liyanage et al.	Won vs. TabPFN
freetype2 (30)	0.811	**0.715**	0.996	30% ($p = 0.01$)	30% ($p = 0.002$)
gif2png (27)	1.755	**0.728**	0.858	81% ($p = 6 \cdot 10^{-4}$)	41% ($p = 0.2$)
jasper (26)	**1.019**	1.328	1.032	35% ($p = 0.4$)	85% ($p = 7 \cdot 10^{-5}$)
jsoncpp (20)	1.778	0.442	**0.319**	100% ($p = 2 \cdot 10^{-6}$)	55% ($p = 0.3$)
readelf (30)	**0.595**	1.017	0.775	23% ($p = 0.009$)	77% ($p = 3 \cdot 10^{-4}$)
Worst program	1.778	1.328	**1.032**		

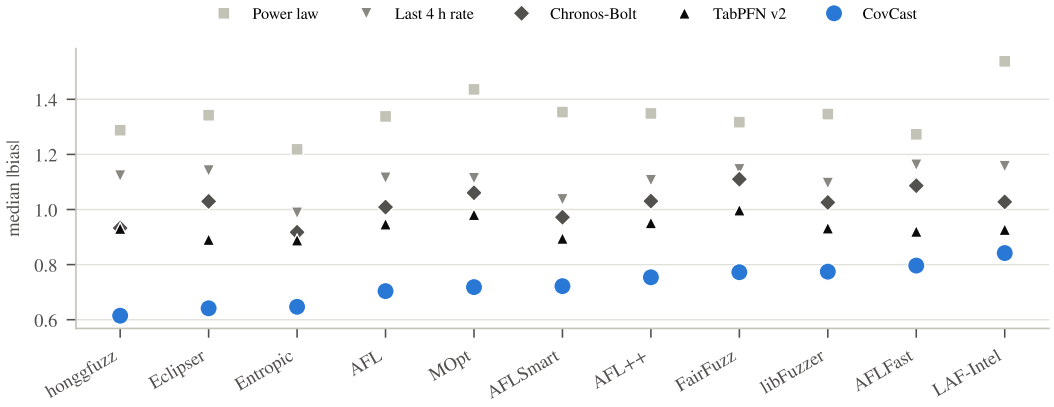


Fig. 4. Median absolute bias per fuzzer on the held-out FuzzBench campaigns (lower is better), with CovCast averaged over four pretraining runs. CovCast is the most accurate method for each of the 11 fuzzers.

Fuzzers. Fuzzers differ in how they mutate inputs and schedule their corpus, and therefore in how their coverage grows. Although CovCast was pretrained without any knowledge of specific fuzzers, it is the most accurate method for each of the 11 fuzzers of FuzzBench (Fig. 4), with median errors from 0.62 to 0.84, against 0.89 to 1.00 for the best baseline.

Campaign stages. A forecast is most valuable early, since a campaign stopped on day two saves six days of compute and one stopped on day five only three. This is where CovCast is strongest (Fig. 5). For forecasts made at every elapsed time up to 62.5 hours, CovCast has a lower median error than both the extrapolator of Liyanage et al. and TabPFN. In the second and third days of a campaign (37.5 to 62.5 hours), it reduces the error of Liyanage et al. at every horizon, by up to 52%, including forecasts three times further ahead than the campaign has run. Over the 2,128 windows of this period, nearly half of the benchmark, its median error is 0.68, against 1.12 for Liyanage et al. (39% lower, 95% CI [25, 50], $p = 3 \cdot 10^{-5}$) and 0.90 for TabPFN (25% lower, [10, 35], $p = 3 \cdot 10^{-5}$). Later in a campaign, once days of history are available, the other methods catch up on this benchmark. On FuzzBench, CovCast remains the most accurate method at every elapsed time (0.77 against 0.99 for TabPFN after 25 hours, 0.72 against 0.91 after 50 hours, and 0.66 against 0.88 after 75 hours).

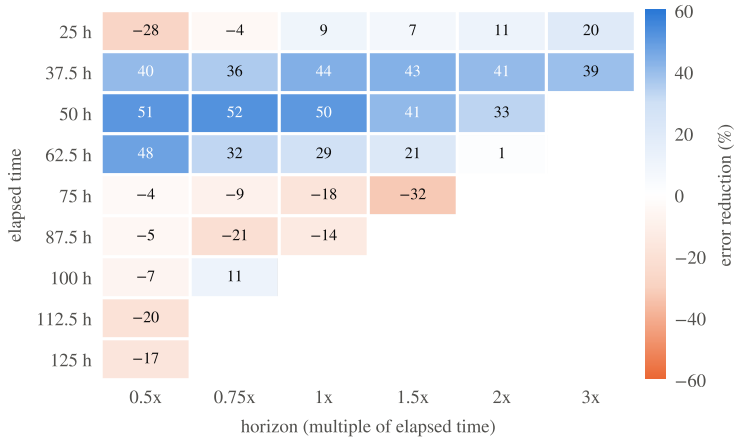


Fig. 5. Reduction of the median absolute bias by CovCast against the extrapolator of Liyanage et al. on the FuzzTastic benchmark (higher is better; blue: CovCast more accurate, orange: less accurate), by elapsed time when the forecast is made (rows) and forecast horizon (columns). Long horizons exist only for early forecasts, because campaigns end after eight days.

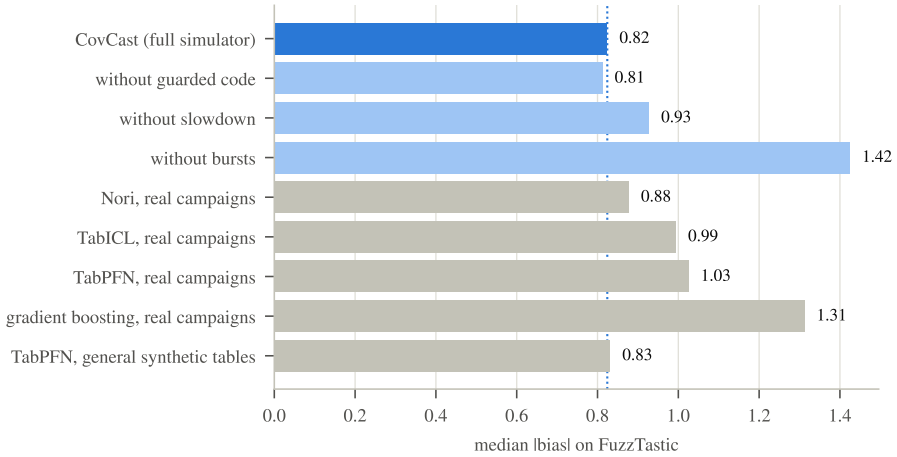


Fig. 6. Median absolute bias on the FuzzTastic benchmark (lower is better) depending on what the model learns from. All models read the same nine columns. Blue: CovCast (dark) and variants of its simulator with one element removed (light), each averaged over its pretraining runs; grey: models that learn from the real campaigns of the other four programs, and a model pretrained on general synthetic tables.

Answer to RQ2: CovCast is the most robust method. Its error on its worst program is 1.03, against 1.78 for the state of the art and 1.33 for TabPFN. It is the most accurate method for all 22 FuzzBench programs and all 11 fuzzers, and on the second and third days of a campaign, when forecasts save the most compute, it reduces the median error of the state of the art by 39% and that of TabPFN by 25%.

4.4 RQ3: What Matters in the Pretraining Data

CovCast differs from general-purpose foundation models only in its pretraining data, so we ask which parts of that data make it accurate. We pretrain variants of CovCast with the same architecture, schedule and number of steps, each with one element of the simulator removed, with two pretraining runs each. Without bursts, every code element is covered on its own, which produces smooth curves. Without guarded code, all code is reachable from the start. Without slowdown, the fuzzer's throughput is constant. We also ask whether real campaigns could replace simulated ones. Real campaign data is scarce in the number of programs rather than in the number of campaigns, so we use a leave-one-program-out protocol. For each program of the FuzzTastic benchmark, models learn from the 103 to 113 campaigns of the other four programs, either as the context of a general-purpose foundation model or as the training data of gradient-boosted trees, and then forecast the held-out program from the same nine columns.

Bursts are by far the most important element of the simulator (Fig. 6). Without them, the median error rises by 73% on the FuzzTastic benchmark (0.82 to 1.43) and by 61% on the FuzzBench selection campaigns (0.68 to 1.09), and the model becomes less accurate than every baseline except Chao and Jost. A model pretrained on smooth campaigns has never seen a long plateau followed by a breakthrough, and therefore cannot anticipate one. The slowdown of the fuzzer also matters (12% higher error on FuzzTastic, 8% on FuzzBench). Removing guarded code has no measurable effect on either benchmark. Both the bursts and the slowdown effects are significant at the level of campaigns ($p < 10^{-15}$) and consistent across pretraining runs.

Learning from real campaigns does not match pretraining on simulated ones. Models that learn from the real campaigns of the other four programs are less accurate than CovCast, with median errors of 0.88 to 1.31 against 0.82. CovCast reduces their median error by 6 to 37%, with confidence intervals that exclude zero for three of the four models, the exception being Nori ([−13, 24]%). The campaigns of four programs do not cover the behavior of a fifth, while the simulator covers a much wider range of behaviors. Since all these models read the same nine columns as CovCast, and the columns alone, such as the power-law extrapolation or the recent rate, are far less accurate (RQ1), the differences stem from what the models learned before seeing the campaign rather than from the design of the table.

Answer to RQ3: The accuracy of CovCast comes from its simulated pretraining data, and above all from simulating coverage bursts, without which its error rises by 61 to 73%. Pretraining on simulated campaigns is also more accurate than learning from the real campaigns of other programs.

4.5 RQ4: Stopping Decisions and Uncertainty

A forecast is useful if it leads to better decisions. We therefore simulate a simple stopping policy on the FuzzTastic benchmark. At each decision point, every 12.5 hours from 25 to 125 hours, a campaign is stopped as soon as its forecast coverage rate over the next half of the elapsed time falls below a threshold. Varying the threshold trades compute saved against coverage lost, measured relative to running every campaign to 187.5 hours. We compare this policy with fixed time budgets, which stop all campaigns at the same time, and with the same policy driven by the true future rate (oracle).

Forecast-based stopping with CovCast saves substantially more compute than fixed budgets. When at most 1% of the final coverage may be lost, it saves 31% of the compute, almost four times as much as the best fixed budget (8%), and close to the oracle (36%). With a loss of at most 1.5%, it saves 39%, against 23% for a fixed budget. For this coarse decision, the forecasts of CovCast are as

useful as those of the extrapolator of Liyanage et al. (32% and 39% at the same losses), although CovCast needs only the coverage over time, and more useful than those of TabPFN (24% and 33%).

CovCast also quantifies the uncertainty of its forecasts, which the extrapolator of Liyanage et al. and the persistence baselines do not. Over all forecast steps of the FuzzTastic benchmark for which CovCast produces a forecast from context, and over its four pretraining runs, its central 80% predictive intervals contain the true rate in 81.3% of the cases, and its 90% intervals in 90.2% of the cases, with 78.6 to 91.4% and 88.0 to 94.1% across programs. The forecasts are therefore well calibrated, which allows stopping policies to account for the risk of a wrong forecast.

Answer to RQ4: Stopping campaigns based on CovCast's forecasts saves almost four times as much compute as a fixed budget for the same coverage loss, and CovCast's predictive intervals are well calibrated.

4.6 RQ5: Cost

Pretraining CovCast took about 70 minutes on four A100 GPUs. It is performed offline and uses no real campaign data. A forecast at one point of a campaign, after 75 hours with 2,000 context rows and 226 future steps, takes 90 milliseconds on one A100 GPU, including building the table, against 842 milliseconds for TabPFN. The power law takes 4 milliseconds. A forecast every 15 minutes therefore adds a negligible cost to a campaign that runs for days. Its cost can be reduced further without loss of accuracy. On the 200 selection campaigns, with forecasts made up to 125 hours, using 1,000 or 4,000 context rows instead of 2,000 changes its median error by at most 0.02. The extrapolator of Liyanage et al., in contrast, simulates many independent-sampling campaigns from per-element hit counts for every forecast.

Answer to RQ5: CovCast is pretrained in about an hour, produces a forecast in 90 milliseconds, and needs only the coverage over time.

5 Discussion

Why does pretraining on simulated campaigns help? A general-purpose foundation model must infer from the table alone how the coverage rate of this campaign evolves. CovCast has been pretrained on about a million campaigns whose rates evolve as in fuzzing, where a plateau is often followed by a burst and the rate decays roughly as a power law with occasional jumps. It can therefore recognize a plateau as a pause rather than an end, as in the campaign of Table 1. This knowledge matters most when few observations are available, which is also when a forecast is most useful (RQ2). Extending the simulator's late phase may extend this advantage to the end of long campaigns as well.

Using more information. CovCast and the baselines of RQ1 and RQ2 use only the observed coverage of the campaign being forecast, as the extrapolator of Liyanage et al. does. In practice, several campaigns often run in parallel on the same program: FuzzBench runs 10 to 20 per fuzzer and program, and OSS-Fuzz runs several fuzzers. Simulating groups of parallel campaigns would allow CovCast to also learn from these campaigns.

From coverage to bugs. Coverage is a proxy for the ability to find bugs. The same approach could forecast bug discovery, for example with the ground-truth bugs of Magma [11], but bugs are rare. In the FuzzBench bug experiment, fewer than half of the 23-hour campaigns crash at all, so that bug curves are much shorter and noisier than coverage curves.

Encoding domain knowledge in pretraining data. CovCast encodes knowledge about fuzzing in its pretraining data rather than in a hand-crafted estimator. Improving CovCast therefore amounts to improving the simulator, whose realism can be checked against real campaigns statistic

by statistic. Synthetic training data has also proven effective for other software engineering tasks where real data are scarce, such as bug detection and repair with artificially injected bugs [2, 28] and vulnerability detection and repair with generated or injected vulnerabilities [15, 16, 27]. We expect the same recipe to apply to other software engineering problems with scarce data but well-understood generating processes.

6 Threats to Validity

Construct validity. We use the definition of the coverage rate, the bias metric and the published predictions of Liyanage et al., so that the comparison is on their terms. Coverage is a proxy for finding bugs [6], but it is the standard measure of fuzzing progress.

Internal validity. The pretraining data contain no real campaign, and the checkpoint was selected on FuzzBench campaigns disjoint from all test campaigns, with a selection rule fixed in advance. While designing the simulator, we only checked that its parameter ranges span three summary statistics of real campaigns, including FuzzTastic campaigns, without using any forecasts or evaluation windows. The general-purpose foundation models receive the same tables as CovCast. To account for the randomness of pretraining, we report CovCast averaged over four pretraining runs. Each run is more accurate than the extrapolator of Liyanage et al. on FuzzTastic and than every baseline on FuzzBench.

External validity. The FuzzTastic benchmark contains five programs and one fuzzer. The FuzzBench campaigns extend the evaluation to 11 fuzzers, 22 programs and edge coverage. Recent estimators of the probability that the next input covers new code [17, 18] do not forecast the rate at a future time and need per-input coverage data that neither benchmark provides, so we do not compare with them.

7 Conclusion

We introduced CovCast, a novel tabular foundation model pretrained only on synthetic fuzzing campaigns, which forecasts the coverage rate of a real campaign from the coverage observed so far, without any re-training. On the FuzzTastic benchmark, its median forecast error is 17% lower than that of the prior state of the art, and 39% lower on the second and third days of a campaign, when stopping decisions matter most. Because it only needs the coverage over time, it also applies to campaigns for which the prior state of the art lacks the required data, and on FuzzBench campaigns of 11 fuzzers it is the most accurate method for every fuzzer and every program. These results suggest that knowledge about fuzzing belongs in pretraining data rather than in estimators fitted to each campaign. We envision forecasts that are available for any campaign from its first hours, so that compute is spent where fuzzing still finds new code.

Data Availability

The simulator, the pretraining and evaluation code, the pretrained model and all forecasts are available at https://github.com/AmL-Dev/CovCast. The evaluation data are the public artifact of Liyanage et al. [22] and the public FuzzBench experiment data [24].

References

[1] Steven Adriaensen, Herilalaina Rakotoarison, Samuel Müller, and Frank Hutter. 2023. Efficient Bayesian Learning Curve Extrapolation using Prior-Data Fitted Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[2] Miltiadis Allamanis, Henry Jackson-Flux, and Marc Brockschmidt. 2021. Self-Supervised Bug Detection and Repair. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[3] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao

Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. 2024. Chronos: Learning the Language of Time Series. *Transactions on Machine Learning Research* (2024).

[4] Marcel Böhme. 2018. STADS: Software Testing as Species Discovery. *ACM Transactions on Software Engineering and Methodology* 27, 2 (2018). doi:10.1145/3210309

[5] Marcel Böhme, Danushka Liyanage, and Valentin Wüstholz. 2021. Estimating Residual Risk in Greybox Fuzzing. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. doi:10.1145/3468264.3468570

[6] Marcel Böhme, László Szekeres, and Jonathan Metzman. 2022. On the Reliability of Coverage-Based Fuzzer Benchmarking. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. doi:10.1145/3510003.3510230

[7] Anne Chao and Lou Jost. 2012. Coverage-based Rarefaction and Extrapolation: Standardizing Samples by Completeness rather than Size. *Ecology* 93, 12 (2012). doi:10.1890/11-1952.1

[8] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. AFL++: Combining Incremental Steps of Fuzzing Research. In *14th USENIX Workshop on Offensive Technologies (WOOT)*.

[9] Amrit L. Goel and Kazu Okumoto. 1979. Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. *IEEE Transactions on Reliability* R-28, 3 (1979).

[10] I. J. Good. 1953. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika* 40, 3–4 (1953).

[11] Ahmad Hazimeh, Adrian Herrera, and Mathias Payer. 2020. Magma: A Ground-Truth Fuzzing Benchmark. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 3 (2020). doi:10.1145/3428334

[12] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. In *International Conference on Learning Representations (ICLR)*.

[13] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. 2025. Accurate Predictions on Small Data with a Tabular Foundation Model. *Nature* 637 (2025). doi:10.1038/s41586-024-08328-6

[14] Taehyoun Kim, Duksan Ryu, and Jongmoon Baik. 2025. DSC-SRGM: A Deep Synthetic Cross-Project Software Reliability Growth Model. *Software Testing, Verification and Reliability* (2025). doi:10.1002/stvr.70028

[15] Amine Lbath, Massih-Reza Amini, Aurelien Delaitre, and Vadim Okun. 2025. AVIATOR: Towards AI-Agentic Vulnerability Injection Workflow for High-Fidelity, Large-Scale Code Security Dataset. arXiv:2508.20866.

[16] Amine Lbath, Manan Suri, Aurelien Delaitre, Vadim Okun, Massih-Reza Amini, Ram D. Sriram, and Dinesh Manocha. 2026. CyberForge: Verified Vulnerability Injection at Repository Level for Cybersecurity Agent Training. arXiv:2608.06471.

[17] Seongmin Lee and Marcel Böhme. 2025. How Much is Unseen Depends Chiefly on Information About the Seen. In *International Conference on Learning Representations (ICLR)*.

[18] Seongmin Lee and Marcel Böhme. 2026. Dependency-aware Residual Risk Analysis. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*.

[19] Stephan Lipp, Daniel Elsner, Thomas Hutzelmann, Sebastian Banescu, Alexander Pretschner, and Marcel Böhme. 2022. FuzzTastic: A Fine-grained, Fuzzer-agnostic Coverage Analyzer. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. doi:10.1145/3510454.3516847

[20] Stephan Lipp, Daniel Elsner, Severin Kacianka, Alexander Pretschner, Marcel Böhme, and Sebastian Banescu. 2023. Green Fuzzing: A Saturation-Based Stopping Criterion using Vulnerability Prediction. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. doi:10.1145/3597926.3598043

[21] Danushka Liyanage, Marcel Böhme, Chakkrit Tantithamthavorn, and Stephan Lipp. 2023. Reachable Coverage: Estimating Saturation in Fuzzing. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*.

[22] Danushka Liyanage, Seongmin Lee, Chakkrit Tantithamthavorn, and Marcel Böhme. 2024. Extrapolating Coverage Rate in Greybox Fuzzing. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*. doi:10.1145/3597503.3639198

[23] Valentin J. M. Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J. Schwartz, and Maverick Woo. 2021. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering* 47, 11 (2021).

[24] Jonathan Metzman, László Szekeres, Laurent Simon, Read Sprabery, and Abhishek Arya. 2021. FuzzBench: An Open Fuzzer Benchmarking Platform and Service. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. doi:10.1145/3468264.3473932

[25] Samuel Müller, Matthias Feurer, Noah Hollmann, and Frank Hutter. 2023. PFNs4BO: In-Context Learning for Bayesian Optimization. In *International Conference on Machine Learning (ICML)*.

[26] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. 2022. Transformers Can Do Bayesian Inference. In *International Conference on Learning Representations (ICLR)*.

[27] Yu Nong, Yuzhe Ou, Michael Pradel, Feng Chen, and Haipeng Cai. 2023. VulGen: Realistic Vulnerability Generation Via Pattern Mining and Deep Learning. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. doi:10.1109/ICSE48619.2023.00211

[28] Michael Pradel and Koushik Sen. 2018. DeepBugs: A Learning Approach to Name-Based Bug Detection. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018). doi:10.1145/3276517

[29] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. 2025. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. In *International Conference on Machine Learning (ICML)*.

[30] Kostya Serebryany. 2017. OSS-Fuzz – Google's Continuous Fuzzing Service for Open Source Software. USENIX Security Symposium, invited talk.

[31] Synthefy, Po-han Li, Aditya Narayanan, Sai Shankar Narasimhan, Raghav Mallampalli, Aahan Agrawal, Bekzat Ajan, Raimi Shah, and Shubhankar Agarwal. 2026. Synthefy Nori: Tabular Foundation Model for Regression. doi:10.5281/zenodo.20710462