Test Parameter Tuning with Blackbox Optimization: A Simple Yet Effective Way to Improve Coverage

†Qijing Huang, Hamid Shojaei, Fred Zyda, Azade Nazi,
Shobha Vasudevan, Sat Chatterjee, Richard Ho
Google Inc., †*UC Berkeley*
{hamids, zyda, azade, shovasu, schatter, riho}@google.com, †qijing.huang@berkeley.edu

Abstract

Constrained random verification in industrial settings involves parameterized tests. The parameters used to control the test stimuli generation are typically set when the test is first written, and seldomly varied later on in nightly regressions. In this work, we formulate test parameter configuration as a blackbox optimization problem and introduce Smart Regression Planner (SRP), an approach that automatically configures the tunable test parameters to better explore the input space and accelerate convergence towards coverage. The optimizer in SRP can drive the parameters update with a Bayesian optimization technique that uses coverage from nightly regressions as feedback. Our evaluation on open-source as well as larger industrial designs demonstrates that SRP leads to up to 9.84% higher average coverage over 100 nights than the human baseline. Importantly, it converges to coverage milestones up to 20× faster than the human baseline. With high-level test parameter optimization, we introduce a problem space and an opportunity to achieve categorically higher coverage in industrial settings with very low overhead. Furthermore, through two practical use cases, we show that employing multi-objective optimization and transfer learning can further accelerate the verification process.

Constrained random verification (CRV) is the de-facto standard in industrial design verification. Central to this process is the design of an elaborate testbench that applies pseudorandom stimulus to the design-under-test (DUT) downstream. The testbench typically consists of *parameterized tests* that are manually crafted for verifying functionality. Each parameter acts as a high-level knob to control stimulus generation, and the testbench then generates a family of related stimuli based on these configurable parameters. Coverage that determines the comprehensiveness of tests is recorded after each regression. Coverage holes found in a nightly regression are addressed by changing parameter configurations, adding new parameters and/or new tests. This verification process is followed iteratively until coverage closure is achieved.

Configuration of test parameters has a significant effect on the coverage. In the RISC-V verification platform [1], we find that random perturbation of test parameters results in ∼60% functional coverage difference between the best and worst configurations. However, verification engineers rarely explore the large space of parameters systematically. Common practice typically relies on increased coverage over time simply through variations in the random seed. Furthermore, the number of parameters increases with increasing design complexity, making it harder for a human to reason about the higher dimensional parameter space effectively. Manual analysis may miss trends among parameter values or draw incorrect conclusions, resulting in sub-optimal coverage and runtime for tests. This adds to the cost of verification and coverage closure which is already a costly, inefficient process taking multiple person-years for industrial designs.

In this work, we investigate the value of *automatically configuring test parameters* towards increased coverage. Given the prevalence of blackbox machine learning (ML) optimization algorithms for hyperparameter tuning, we asked ourselves the following question: *Is it possible to leverage state-of-the-art hyper-parameter exploration techniques to improve coverage by exploring the space of test parameters?*

A. Solution and contributions

We introduce Smart Regression Planner (SRP), an approach to automatically optimize test parameters with the goal of quick coverage convergence. Traditional research on input stimulus generation is at the Boolean input level, instruction (or transaction) level, or at the constraint level [2], [3], [4]. Searching the space of input stimulus directly suffers from combinatorial explosion. In contrast, SRP works at a higher level of abstraction that naturally has much fewer configurable inputs (< 100); nonetheless it directly impacts verification coverage. The more tractable input dimensions allow the application of powerful optimization methods to this problem. In SRP, we formulate test parameter configuration as a blackbox optimization problem with an objective to maximize coverage. We first employ a simple random search to configure test parameters as a baseline. We then apply ML-based Bayesian optimization methods that can leverage coverage feedback from past regression tests and learn near optimal parameter configurations. Bayesian optimization [5] is agnostic to structure and flexible enough to adapt to changes in an evolving design. While random search relies purely on exploration, Bayesian optimization exploits learning through feedback. We also investigate use cases of (1) simultaneously minimizing runtime and maximizing coverage using multi-objective Bayesian optimization, and (2) transfer learning, or the ability to transfer learned heuristics from one set of parameters to another through design evolution.

One advantage of our proposal is that it provides an easy way to leverage the tests written by the verification engineers without requiring *any* additional effort from the verification engineers beyond setting up the system. We have implemented SRP in an industrial nightly regression flow where the test parameters can be reconfigured for every test run. SRP fits into the flow with low overhead. Furthermore, when regressions are run in the cloud, SRP can opportunistically exploit idle capacity to increase exploration, leading to faster coverage improvement. In addition, we leverage advanced machine learning techniques to tackle real-world deployment problems in verification, such as improving not just the coverage, but also the runtime (by having multiple objectives), and efficiently handling rapidly evolving testbenches and designs (through transfer learning).

We evaluate SRP on two sets of benchmarks: the open-source RISCV [1] and IBEX [6] designs, as well as the industrial TPU [7] design. Since the state of practice in CRV is to use fixed human-specified parameters, we use this as our baseline. Our comprehensive set of experiments categorically show that test parameter optimization consistently provides significant value to CRV. In practical settings, even a 1-2% coverage improvement can substantially save human effort and time to closure. We measure coverage computed nightly (point-in-time coverage) and the cumulative coverage over many nights (accumulated coverage). Improvement in point-in-time coverage enables more timely discovery of bugs while faster convergence on accumulated coverage shortens the chip verification cycles for signoff. Our proposed flow always achieves the maximal point-in-time coverage percentage (up to 9.84 higher than baseline) as well as the maximal accumulated coverage percentage (up to 5.58 higher than baseline) across 100 nights on all designs. Our flow also converges much faster than the human baseline on the accumulated coverage. Over the IBEX design, it takes SRP up to at least 95 fewer nights to reach the last 5% of max attainable coverage, showing a 20× speedup in coverage convergence. Overall, SRP tests can detect more issues (6.86 failures per 1000 tests) than human baseline (5.94 failures per 1000 tests) on TPU. Finally, we show multi-objective optimization in SRP improves the runtime of simulations by 15% without taking a hit on coverage.

In summary, our contributions are as follows.

- We identify a simple yet effective method for faster coverage convergence with minimal human intervention by automatically configuring test parameters in constrained random verification using standard ML-based blackbox optimization algorithms
- We demonstrate, with comprehensive evaluation for an industrial design and open-source designs, that test parameter optimization provides an increase in point-in-time, and fast convergence to accumulated coverage, which are highly valuable during coverage closure and bug detection.
- We show that blackbox Bayesian optimization is ideally suited to configure test parameters to achieve higher functional as well as code coverage.
- We show value in the Bayesian algorithms that simultaneously optimize runtime and coverage with multi-objective optimization as well as transfer learning to deal with evolving design and testbenches.

I. Our Approach: Smart Regression Planner

A. Problem Formulation

Consider a *parameterized test* $T(p_1, p_2, \ldots p_n)$ where p_i is test parameter that can be numerical, categorical, or Boolean as shown in the example of Fig. 1. Given an assignment of values v_i to each p_i, simulation returns the point-in-time coverage C_{PIT} which is a real number between 0 and 100%. We call the tuple of test parameter values $v = (v_1, v_2, .., v_n)$ a *test configuration* for T. The objective of regression planning is to find a test configuration v^* that maximizes C_{PIT}. This problem is stochastic as C_{PIT} is affected by randomness. Since the function f that maps a test configuration v to C_{PIT} does not have any obvious structure to be exploited for optimization (such as convexity or smoothness), it is natural to consider black-box techniques.

```
test: riscv single_rand_test
iterations: 15
gen test: riscv_rand_instr_test
gen opts: >
  +instr cnt=10000
  +num of sub program=5
  +illegal instr ratio=10
  +hint instr ratio=20
  +no ebreak=1
  +no wfi=0
  +stream name 1=riscv_loop_instr
  +stream freq 1=40
  +stream name 2=riscv_hazard_instr_stream
  +stream freq 2=10
  +stream name 3=riscv_jal_instr
  +stream_freq_3=10
```

Fig. 1: Example test parameters for IBEX [6]. There are four categories of configurable parameters in this design, including integer ($instr\_cnt$, $num\_of\_sub\_program$), ratio ($illegal\_instr\_ratio$, $hint\_instr\_ratio$), boolean (no_ebreak, no_wfi), and frequency ($stream\_freq\_N$).

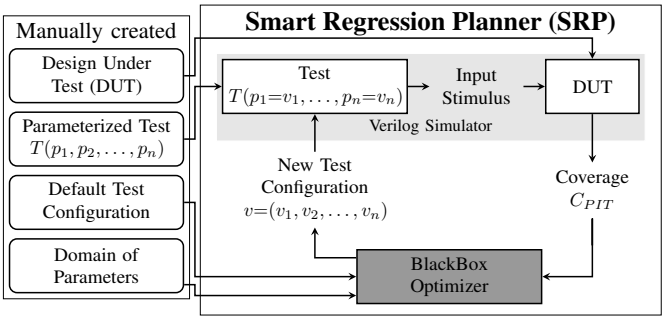


Fig. 2: The Smart Regression Planner (SRP) framework.

B. SRP Framework

Fig. 2 shows the proposed SRP framework to leverage ML-based black box optimization for improving coverage closure in simulation. In addition to the default test configuration, the test uses the parameter configuration provided by the blackbox optimizer to generate inputs for the DUT. The point-in-time coverage C_{PIT} is computed by the Verilog simulator that simulates the test and the DUT, and this value is fed to the blackbox optimizer. The optimizer then generates a new value for each test parameter p_i from its valid parameter domain and feeds the parameters back to the test. The test and the design are re-simulated with the new configuration. The optimizer tracks coverage results as a function of the test configuration. Over a series of simulation runs, the optimizer learns which combinations of test parameter values lead to maximum coverage. The cycle continues until coverage closure is achieved or the maximum number of simulation runs allowed is reached.

In real CRV deployment, more than one set of tests can be invoked every night. Instead of running with the default parameters multiple times with different random seeds, we propose to run the optimizer suggested parameters *in addition* to the default (verification engineer-specified) parameters in SRP. In addition, the optimizer's suggestions are automatically checked in to source control alongside the parameterized tests in order to track and diagnose regression failures with the existing tools.

The main goal of the SRP framework is to optimize the point-in-time coverage C_{PIT} and achieve more comprehensive nightly bug detection during the hardware development. However, since the blackbox optimization algorithm in SRP frequently perturbs the test parameters which enables more exploration, our proposed flow subsequently impacts the accumulated coverage C_{ACC} for a sequence of test configurations. Therefore, we also evaluate the impact of different approaches on the accumulated coverage C_{ACC} since it is a critical metric for coverage closure and sign-off.

C. Blackbox Optimization in SRP

We formulate regression planning as an optimization problem. The goal is to find $v^* = \text{argmax}_v f(v)$, where v^* is test configuration that maximizes the C_{PIT}. Several classes of algorithms have been proposed from a simple RANDOM-SEARCH to more powerful techniques like Gaussian Process Bandits (GP-BANDIT) [5].

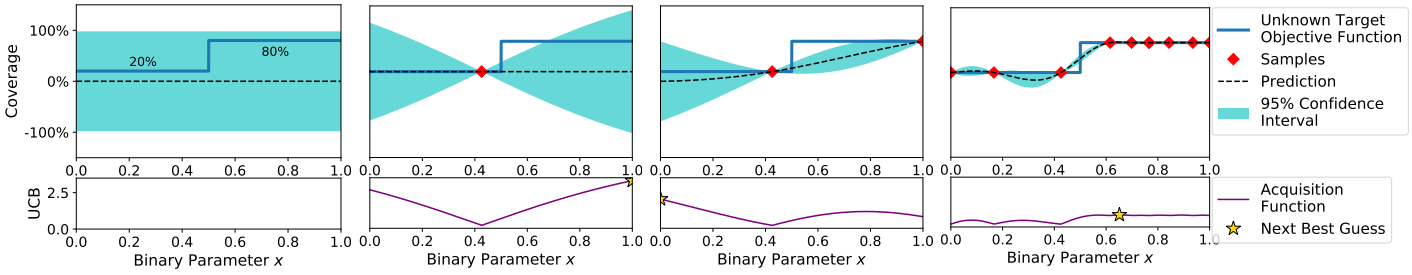
RANDOM-SEARCH: This approach selects v_t uniformly at random at time step t independent of the previous points selected and does not require any coverage feedback.

GP-BANDIT: In this approach, we formulate the problem as *multi-armed bandit* [8], [5] and apply GP-BANDIT [5] to find v^* . GP-BANDIT chooses a limited set of configurations to maximize the expected value of the blackbox objective function. It attempts to learn the properties of $f(v)$ and optimize an expected reward (e.g. $E[f(v)]$ over N nights in SRP) with a limited number of trials. In each trial, the algorithm decides whether to pick a new configuration for exploration or a greedy configuration to exploit the knowledge learned from history. This tradeoff is known as the exploration-exploitation dilemma that multi-armed bandit algorithms aim to tackle. A side effect of exploration in the SRP problem is that it could benefit C_{ACC}.

D. GP-BANDIT *Example*

This section describes a detailed example of how GP-BANDIT selects the test parameters in SRP to maximize the blackbox coverage objective f.

[1]figure source code adopted from https://github.com/fmfn/BayesianOptimization



(a) GP Prior ($\mu = 0$, $\Sigma = 0.5$) (b) Iteration 1 (c) Iteration 2 (d) Iteration 10

Fig. 3: Bayesian optimization (BO) example for a binary parameter x (e.g., no_ebreak, no_wfi in Fig.1). (a) shows the target objective function to be learned (solid blue line) and the initial GP prior with mean of 0 and standard deviation of 0.5 (its 95% confidence interval is shaded in blue). When the prediction $x < 0.5$, the binary parameter is rounded to 0, the coverage return is 20%; when $x \geq 0.5$, the binary parameter is rounded to 1 and the coverage return is 80%. Note that the coverage prediction can go negative since the target function is unknown without any training. (b) shows that BO randomly samples a point $x = 0.4$ and updates the corresponding posterior with its 95% confidence interval shaded in blue. We compute the UCB acquisition function (purple line) and sample the next point $x = 1$ that maximizes the function. In (c), BO updates the posterior and predicts the next sample $x = 0$ to maximize the acquisition function. In (d), BO obtained the predicted GP (dotted black line) that approximates the target objective function (solid blue line) after 10 iterative samples. [1]

Bayesian Optimization: GP-BANDIT is essentially a Bayesian optimization method as it models f as a Gaussian Process (GP). There are two key components in Bayesian optimization: *a statistical model* for approximating the blackbox objective function f and *an acquisition function* for deciding where to sample next. As shown in Fig. 3, the Bayesian optimization works as follows for SRP:

Step 1: A Gaussian Process (GP) will be used as a prior on the black box objective f as shown in Fig. 3a. The GP is a collection of K random variables, e.g., test parameters in SRP), of which have joint Gaussian distributions. Note that the GP is defined over functions. The GP prior does not depend on the training data but specifies some properties of the functions.

Step 2: Run simulation to collect coverage data and use all available data to update the posterior probability distribution on f. The posterior distribution is shown with the dotted line and the blue region in the top coverage plot of Fig. 3b.

Step 3: Select the parameters v that maximize the acquisition function computed using the current posterior distribution. The acquisition function is shown with purple line in the bottom plot of Fig. 3b. The star represents the next point to sample.

Step 4: Repeat Step 2 to 3 for N nights.

Discrete Parameters: In GP-BANDIT, discrete numerical parameters are embedded in $\mathbb{R}$. Categorical parameters with k feasible values are represented via one hot encoding, i.e., $[0,1]^k$. Binary parameters are encoded in $[0,1]^2$ as they are essentially categorical parameters with two classes. These representations present the Gaussian Process a continuous and differentiable space to optimize on. The algorithm then discretizes the points by rounding them to the nearest feasible points once converged. As shown in Fig.3b, the sampled variable value x is 0.4, for a binary test parameter, e.g., no_ebreak, we round it to 0, run the next simulation with $x = 0$, and return the coverage.

Acquisition Function: The GP-BANDIT algorithm in SRP uses upper confidence bound (UCB) as the acquisition function [5] to deal with the exploitation-exploration trade-off. UCB selects the test parameter v_t at night t based on the following equation:

$$v_t = \text{argmax}_v \left[\mu_{t-1}(v) + \beta_t^{1/2} \sigma_{t-1}(v) \right] \tag{1}$$

where $\mu_{t-1}(v)$ is the mean coverage of test parameter v at night $t-1$, $\sigma_{t-1}(v)$ is the standard deviation of test parameter v at night $t-1$, and β is a confidence parameter that controls the level of exploration. As shown in Fig. 3, $\mu(v)$ (dotted line) approximates the mean of the objective function from existing samples. $\sigma(v)$ reflects the uncertainty of the approximation and its value is low around the sampled points.

E. Multi-objective Optimization

In SRP, different parameter configurations can lead to different simulation runtime. To reduce the total cost on verification, our secondary goal is thus to minimize the runtime of each simulation while maximizing its coverage. This turns our SRP optimization problem into multi-objective optimization.

In contrast to single-objective blackbox optimization where $f(v)$ is a scalar objective function, multi-objective optimization aims to maximizes k objectives, $F(v) := f_1(v), ..., f_i(v)$, which are possibly competing objectives. The goal of multi-objective optimization is to converge to the entire Pareto frontier $\mathcal{F}$ of the objective space if it is impossible to maximize all objectives simultaneously. To obtain a uniform metric for comparing different solutions, prior work [9] has introduced the hypervolume indicator that computes the volume of the dominated portion in the Pareto set. This indicator is a function of multiple objective vectors with NP-hard computation complexity. Therefore, various scalarization strategies are developed to approximate the hypervolume indicator. GP-BANDIT used in SRP employs the random hypervolume scalarization technique [10] to efficiently approximate a scalar hypervolume indicator. This technique rewrites the hypervolume as a specific expectation function of maximization of random scalarizations.

F. Transfer Learning

In GP-BANDIT, transfer learning trains on both the current samples and the priors generated from the previous samples. Suppose we have a trained regressor R with posterior mean μ_0 and posterior standard deviation σ_0 on dataset D_0, given a new input dataset $D_1 = (x_t^1, y_t^1)_t$, we want to learn a new mean function μ_1 and standard deviation function σ_1 with transfer learning. The algorithm we used in SRP [11] introduces another regressor R' to train with the residual labels of D_1 on μ_0, which is $(x_t^1, y_t^1 - \mu_0(x_t^1))_t$. The regressor R' outputs the posterior mean function μ' and posterior standard deviation σ'. The new mean function μ_1 for $D0$ and $D1$ is updated to $\mu_1(x) = \mu_0(x) + \mu'(x)$. The new standard deviation function σ_1 is calculated as a weighted geometric mean of $\sigma_0(x)$ and $\sigma'(x)$, where the weights are a function of the amount of data in D_0 and D_1.

II. EXPERIMENTAL RESULTS

In this section, we characterize how SRP compares to the default test parameters set by the verification engineers (which is the BASELINE flow). Our results demonstrate that SRP consistently achieves higher point-in-time and accumulated coverage than the original flow. We also show the impact of applying multi-objective optimization and transfer learning in Bayesian optimization.

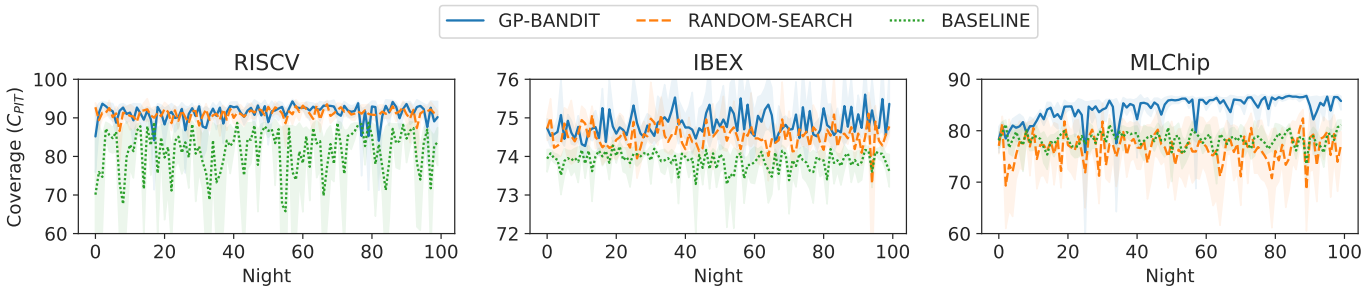
Designs: We evaluate SRP on three designs: RISCV [1], IBEX [6], and MLChip (i.e., TPU [7]). Both functional coverage (RISCV and MLChip) and code coverage (IBEX and MLChip) are targeted in these designs. In RISCV, there are in total 15 ordinal test parameters, each with 10–40 categories. IBEX contains 31 test parameters. Sixteen of them are categorical with two classes, while the rest are the same ordinal parameters from the RISCV testbench. In contrast to general-purpose RISCV/IBEX design, MLChip follows the CISC tradition for its custom instruction set architecture (ISA) design. The test parameters for MLChip include many distribution specifications for the instructions and the test vector values. There are unreachable cover points in the IBEX and MLChip design, so the corresponding aggregated coverage could not reach 100%.

Methods: We evaluate the performance of SRP with respect to both RANDOM-SEARCH and GP-BANDIT vs the BASELINE flow which is the fixed human-generated parameters. When running RANDOM-SEARCH and GP-BANDIT in SRP, a new test parameter configuration is generated for the simulation. The configuration stays unchanged during simulation for the same test.

Setup: For the blackbox optimizer in SRP, we repurpose the open-source implementations of these algorithms with default parameters from Google's hyperparameter tuning platform Vizier [11]. To mitigate the impact of randomness on the results, we run each experiment five times and report the average coverage across the five runs.

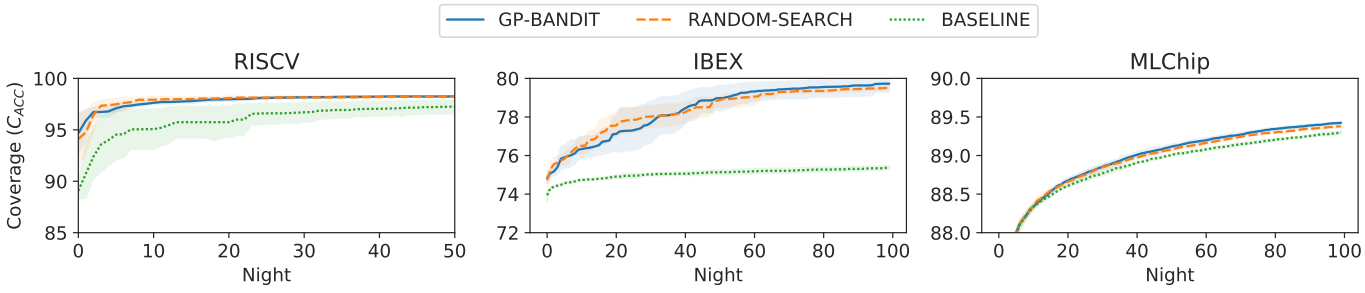
A. Proposed Verification Flow

To benefit from both exploration introduced by SRP and exploitation from low-variance baseline setup with human-specified parameters, we propose a new use case for regression testing by running SRP in addition to the original BASELINE flow. To keep the comparison fair when comparing with the original baseline, we run two copies of the baseline (with different seeds) thus ensuring the same amount of simulations in both cases. In this study, we merge the coverage for an experimental algorithm run with a baseline run and report it as the point-in-time coverage (C_{PIT}) for every iteration.



	RISCV					IBEX					MLChip				
	iter(95%)	iter(99.5%)	max	mean	std	iter(95%)	iter(99.5%)	max	mean	std	iter(95%)	iter(99.5%)	max	mean	std
GP-BANDIT	2	**58**	**94.28**	**91.27**	2.15	**1**	**17**	**75.60**	**74.84**	0.30	**13**	**64**	**86.78**	**84.31**	2.31
RANDOM-SEARCH	**1**	-	93.53	91.14	1.26	**1**	-	75.11	74.52	0.29	-	-	82.40	76.57	2.70
BASELINE	-	-	88.83	81.43	5.81	**1**	-	74.28	73.89	0.22	-	-	81.20	78.35	1.39

Fig. 4: **Point-in-time Coverage** C_{PIT}. In the figure, each line represents the mean coverage across five random seeds and the shaded region shows the standard deviation across the five runs. In the table, $iter(95\%)$ and $iter(99.5\%)$ show the nights taken to reach 95% and 99.5% of the maximal attainable coverage. "-" indicates the target coverage is not reached within 100 nights. max, $mean$, std represent the maximum, average, and the standard deviation of coverage across 100 nights respectively. **GP-BANDIT** achieves the highest maximum and mean C_{PIT} and consistently outperforms BASELINE over 100 nights on all designs.



	RISCV					IBEX					MLChip				
	iter(95%)	iter(99.5%)	max	mean	std	iter(95%)	iter(99.5%)	max	mean	std	iter(95%)	iter(99.5%)	max	mean	std
GP_BANDIT	**1**	17	98.29	98.04	0.53	**5**	**62**	**79.72**	**78.37**	1.36	**2**	**39**	**89.42**	**88.85**	1.18
RANDOM_SEARCH	**1**	**9**	**98.30**	**98.08**	0.59	**5**	73	79.50	78.36	1.17	**2**	42	89.38	88.81	1.30
BASELINE	4	80	97.95	96.80	1.48	-	-	75.38	75.06	0.26	**2**	49	89.30	88.75	1.19

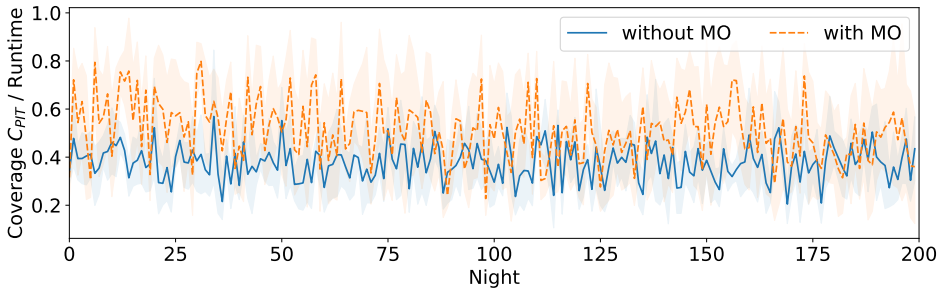
Fig. 5: **Accumulated Coverage** C_{ACC}. SRP algorithms, augmenting GP-BANDIT or RANDOM-SEARCH, achieve the highest maximum and mean C_{ACC} and take fewer nights to reach 95% and 99.5% attainable coverage on all designs.

In Fig. 4, we see that this mode ensures that C_{PIT} driven by GP-BANDIT in almost every night is higher than the BASELINE on all designs. Note that RANDOM_SEARCH does not provide such assurance. Its exploration is quite expansive and sometimes falls below the baseline. GP-BANDIT, contrarily, has learned the parameter values to optimize C_{PIT} through both exploration and exploitation. As a result, employing GP-BANDIT in SRP produces highest maximum and mean C_{PIT} coverage in 100 nights on all designs as shown in the Fig. 4 table. Fig. 5 shows that accumulated coverage C_{ACC} is consistently higher than BASELINE for every design with SRP algorithms running GP-BANDIT or RANDOM-SEARCH. With more exploration, SRP algorithms converge to the highest accumulated coverage C_{ACC} much faster than baseline, resulting in huge time savings during coverage closure.

The GP-BANDIT mode is a highly attractive proposition for practical settings as it simultaneously achieves higher point-in-time coverage and accumulated coverage than human baseline. The prediction time of GP-BANDIT is within seconds, which is negligible compared to the simulation time of our testbenches which ranges from tens of minutes to hours.

B. Multi-Objective Optimization

Optimizing for high coverage can sometimes lead to unacceptably high simulation runtimes. An engineer might want to trade off one for the other at different points in the verification phase. We explore multi-objective optimization (MO) in Bayesian optimization [10] to simultaneously minimize simulation runtime and maximize coverage. Fig. 6 shows that adding MO leads to $1.18\times$ **speedup in the mean runtime while achieving higher mean coverage over 200 nights.** Shown in Fig. 6, the mean C_{PIT} over runtime ratio is 34% higher with MO, demonstrating the effective optimization of both objectives.



	Coverage C_{PIT}				
	max_20	max_50	max_100	mean	std
without MO	73.60	74.42	74.49	70.84	1.48
with MO	74.64	74.64	75.12	**71.81**	1.62

	Runtime (s)	
	mean	std
without MO	240.62	66.93
with MO	**203.45**	67.28

	Ratio: C_{PIT} / Runtime				
	max_20	max_50	max_100	mean	std
without MO	0.48	0.57	0.57	0.38	0.07
with MO	0.79	0.80	0.80	**0.51**	0.12

Fig. 6: **Multi-objective optimization (MO) to improve runtime and coverage.** The figure shows the ratio of coverage C_{PIT} over runtime for IBEX. Applying MO improves this ratio. In the table, max_20, max_50, and max represent the maximum value achieved in 20, 50, and 100 nights respectively. With MO, GP-BANDIT leads to $1.18\times$ speedup in test runtime while achieving higher mean C_{PIT}.

C. SRP Bug Detection Study

We deployed SRP into real production for the MLChip design (under active development) and ran it for 30+ days with the GP-BANDIT algorithm. Then we collected the unique failure signatures found in the last three days. These signatures represent runtime failures and tests that are failing due to infrastructure issues or compile and build errors are not counted. Our bug detection study yielded the following results. 29 signatures were found from 4230 tests (6.86 failures per 1000 tests) driven by GP-BANDIT while BASELINE found 26 signatures over 4378 tests (5.94 failures per 1000 tests). Given that each unique signature manifests a distinct bug in design or testbench, **this experiment demonstrates not just higher coverage, but better bug detection capability with fewer tests.**

D. Learning in SRP

1) GP-BANDIT *based learning:* In Fig. 7, each curve represents specific parameter configurations of every test selected by GP-BANDIT. Based on the density of lines across each Y axis representing the parameter value, we see the following parameter setup is preferred by GP-BANDIT: test instructions count $> 10k$, $4-20$ number of subprograms, $15\%-35\%$ of illegal instructions and $10\%-50\%$ of test hint instructions. Fig. 8 shows an example test generated by SRP and the corresponding human-generated baseline test. Interestingly, some of the parameters like stream frequency of instructions are heuristically decided by humans, but the optimizer settles at a very different value (like stream_freq_2). The SRP test uses fewer instructions, counter-intuitive to our understanding, yet achieves a higher coverage than the baseline.

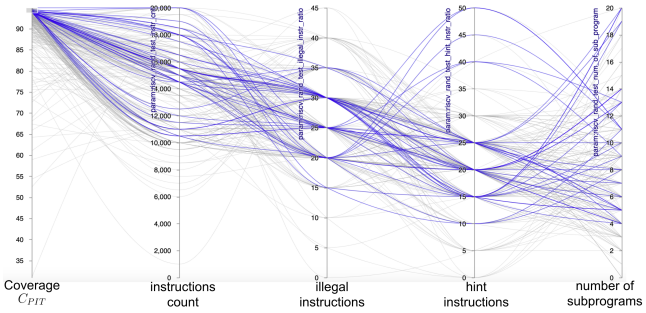


Fig. 7: Test parameters suggested by GP-BANDIT for 100 nights. Each blue and grey curve represents test parameters for one night. Blue lines show the configurations that lead to high coverage.

Flow	Test Parameters
SRP C_{PIT} = 75.9	hint_instr_ratio=15, illegal_instr_ratio=50, instr_cnt=7000, num_of_sub_program=8,stream_freq_0=5, stream_freq_1=40, stream_freq_2=0, stream_freq_3=20, stream_freq_4=10, stream_freq_5=50, stream_freq_6=0, stream_freq_7=10, stream_freq_8=20, stream_freq_9=25, stream_freq_10=35
Baseline C_{PIT} = 74.2	hint_instr_ratio=10, illegal_instr_ratio=30, instr_cnt=14000, num_of_sub_program=4, stream_freq_0=45, stream_freq_1=30, stream_freq_2=30, stream_freq_3=30, stream_freq_4=35, stream_freq_5=45, stream_freq_6=25, stream_freq_7=45, stream_freq_8=35, stream_freq_9=25, stream_freq_10=35

Fig. 8: A comparison between the test parameters originally set by the verification engineers and the optimized parameters learned by the SRP framework for RISCV.

2) Transfer Learning: As the design evolves, new tests and parameters may be added; the existing ones may get deleted. The deletion of parameters can be easily addressed by removing the corresponding input dimensions in GP-BANDIT. The addition of parameters, on the other hand, can be more challenging to handle. Instead of re-training the blackbox algorithms in this case, we investigate the ability to transfer learned heuristics and improve sample efficiency. In this experiment, we held out 5 of the 11 parameters during initial optimization, then added them back to simulate new parameters being added to the test. We repeated this experiment for 100 random subsets of the parameters to account for the possibility that some parameters may have an outsized influence on coverage. Results of transfer learning in Fig. 9 show that the prior learned from the initial training applies well to new runs with any set of 5 additional test parameters added. The C_{PIT} **coverage with transfer learning starts higher and converges around 20 nights earlier** than the runs without transfer learning. However, we observe no improvement in the accumulated coverage C_{ACC} with transfer learning enabled, potentially due to limited exploration.

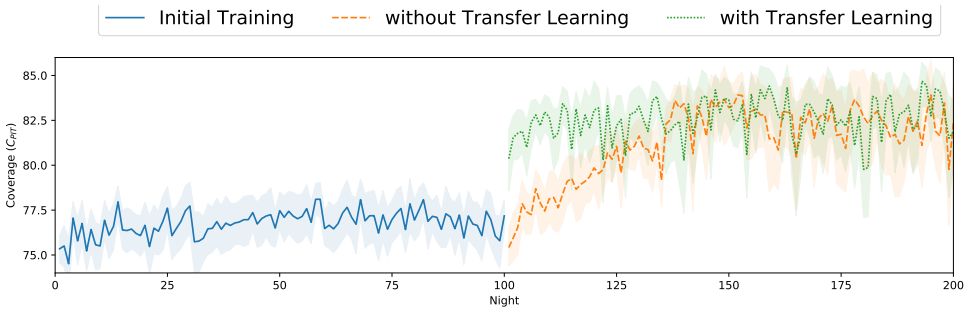


Fig. 9: **Transfer learning to more efficiently handle evolving designs.** Point-in-time coverage C_{PIT} over 200 nights. Blue line shows the initial GP-BANDIT runs in the first 100 nights on 6 parameters. Green and orange lines show the subsequent 100 runs when 5 more parameters are added to the design. Green line shows GP-BANDIT performance on 11 parameters with transfer learning.

E. Ablation Studies

In this section, we show the impact of random seeds, followed by another ablation study showing the impact of blackbox optimization methods solely on the point-in-time and accumulated coverage compared to the baseline,

1) Optimization with Fixed Random Seed: Normally in CRV, the random seed is varied from run to run to get greater coverage through randomness, but this leads to a stochastic learning problem for the blackbox optimizer. We ran a control experiment with a fixed seed to examine if the learning in GP-BANDIT proceeds better when the randomness in feedback is eliminated. We ran 100 nights with the same fixed seed for all algorithms, repeated the experiment 5 times with 5 different seeds and reported results in Table I. The standard deviation of C_{PIT} across 100 nights of GP-BANDIT is consistently lower than RANDOM-SEARCH on all designs, showing the feedback based learning in the absence of randomness is more consistent.

	RISCV					**IBEX**					**MLChip**				
	iter(95%)	iter(99.5%)	max	mean	std	iter(95%)	iter(99.5%)	max	mean	std	iter(95%)	iter(99.5%)	max	mean	std
GP-BANDIT	**1**	**60**	**93.71**	**88.79**	4.55	**1**	37	74.42	71.05	1.36	**22**	**63**	**86.02**	**82.89**	3.51
RANDOM-SEARCH	2	-	92.23	86.17	5.23	**1**	**1**	**74.67**	69.98	1.58	-	-	79.95	68.53	6.02
BASELINE	-	-	82.89	82.89	0.00	**1**	-	74.01	**74.01**	0.00	-	-	70.49	70.49	0.00

TABLE I: **Ablation Study with Fixed Seed**: statistics on point-in-time coverage C_{PIT}. In the absence of randomness, the coverage for BASELINE remains constant. Lower variance is observed with GP-BANDIT compared to RANDOM.

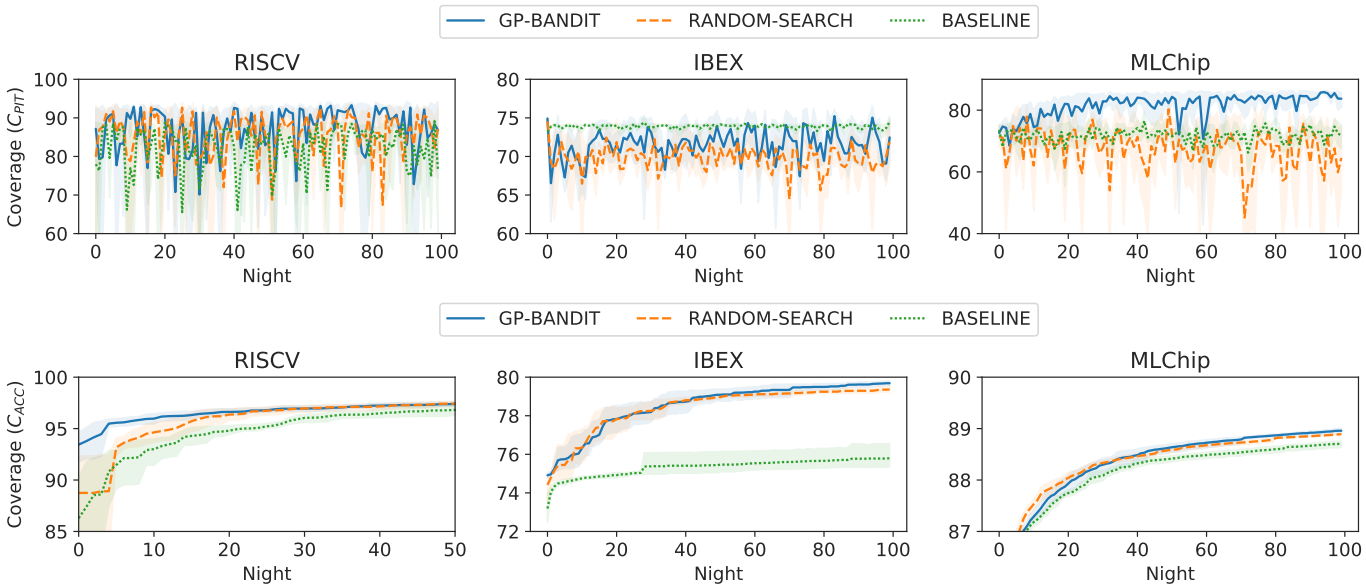


Fig. 10: **Ablation Study with Blackbox Optimization Only**. More exploration in blackbox optimization leads to both higher maximum point-in-time coverage (C_{PIT}) and accumulated coverage (C_{ACC}). However, in spite of getting better maximum C_{PIT}, as seen on IBEX, average C_{PIT} may be worse due to increased variance from exploration, particularly when the baseline parameter values are already well-tuned. This study motivates our recommended flow of using GP-BANDIT in conjuction with BASELINE.

2) Blackbox Optimization Only: In this study, we compare coverage reported by each method when using a random seed as is part of the state-of-practice standard CRV. From Fig. 10, we observe that, on IBEX with well optimized human-defined parameters, more exploration with blackbox optimization (BO) approaches leads to higher accumulated coverage C_{ACC}, but introduces variance to the point-in-time coverage C_{PIT}. When applying GP-BANDIT on MLChip, we see an upward coverage trend increasing over nights, leading us to conclude that the GP-BANDIT can learn complex parameter spaces well enough to maximize coverage. This justifies our recommendation to leverage the advantages of both approaches by running GP-BANDIT in addition to the BASELINE as discussed in Section II-A. For accumulated coverage, BO methods achieve a) higher maximum and b) faster convergence coverage over the BASELINE on all designs. These numbers show the value of the BO flow for accumulated coverage during coverage closure. Although C_{ACC} is not a metric that is directly optimized by GP-BANDIT, more exploration on the parameter space enabled in the these algorithms leads to significant improvement on C_{ACC}, especially on IBEX.

III. RELATED WORK

Input stimulus generation techniques with coverage as feedback have been explored by many previous techniques [12], [13], [14], [15], [2], [16], some of which are based on Bayesian networks [3], [4] and ML-based stimulus generation [17], [18]. Fuzzing techniques for inputs and tests have also been studied for hardware verification [19], [20], [21]. In comparison, SRP uniquely tackles the end-to-end problem at a different level of abstraction by leveraging the existing tests specified with human expertise and automatically configuring the test parameters to maximize coverage. This is a more tractable search space than the space of stimuli which allows blackbox optimizers to be effective. Despite the difference in abstraction levels, we discuss the state-of-the-art approaches with respect to our approach from an algorithmic standpoint. Static-analysis-based approaches for test generation [22], [23] have inherent scalability limitations. Approaches that combine static and dynamic analysis like HYBRO [24] and concolic testing [25], [26], [27] in RTL rely on SMT/SAT solvers, which are also limited by scale. Other approaches combine random forests, and decision trees with static analysis and formal verification [28], [29] require manual feature engineering and hand-crafted algorithms. Although interesting techniques have been put forth [30], [31], [32], none of them are publicly available to be evaluated on industrial designs. Our proposal, on the other hand, allows us to use commercial simulators to provide industry-standard coverage metrics and publicly available blackbox optimizers [11] to improve coverage closure for large designs.

IV. CONCLUSION

In this work, we have formulated a verification problem capable of significant practical impact, at an abstraction level where scalability is a natural byproduct. We have shown that employing Bayesian optimization to adjust parameters in a testbench allows us to improve coverage in real industrial settings. The improvement comes at a very low engineering overhead: the system can be assembled from standard commercial RTL simulators and off-the-shelf blackbox optimizers, and involves no effort from the verification engineers once set up. Furthermore, we show that through multi-objective optimization, it is possible to improve both test run-time and coverage (leading to more efficient use of simulator licenses and compute), and through transfer learning, it can effectively handle continuously evolving designs and testbenches (as is a requirement in practice).

REFERENCES

[1] *SV/UVM based instruction generator for RISC-V processor verification*, 2020. [Online]. Available: https://github.com/google/riscv-dv

[2] M. imková and Z. Kotásek, "Automation and optimization of coverage-driven verification," in *Euromicro Conference on Digital System Design*, 2015.

[3] S. Fine and A. Ziv, "Coverage directed test generation for functional verification using bayesian networks," in *Proceedings of the Design Automation Conference (DAC)*, 2003.

[4] M. Braun, S. Fine, and A. Ziv, "Enhancing the efficiency of bayesian network based coverage directed test generation," in *Proceedings. Ninth IEEE International High-Level Design Validation and Test Workshop*, 2004.

[5] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[6] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in *International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017.

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[8] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, vol. 58, no. 5, 1952.

[9] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, 1999.

[10] R. Zhang and D. Golovin, "Random hypervolume scalarizations for provable multi-objective black box optimization," in *International Conference on Machine Learning*, 2020.

[11] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

[12] L. Liu and S. Vasudevan, "Scaling input stimulus generation through hybrid static and dynamic analysis of rtl," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2014.

[13] L. Piccolboni and G. Pravadelli, "Simplified stimuli generation for scenario and assertion based verification," in *Latin American Test Workshop-LATW*, 2014.

[14] S. Yang, R. Wille, D. Große, and R. Drechsler, "Minimal stimuli generation in simulation-based verification," in *Euromicro Conference on Digital System Design*, 2013.

[15] Y. Zhou, T. Wang, T. Lv, H. Li, and X. Li, "Path constraint solving based test generation for hard-to-reach states," in *Asian Test Symposium*, 2013.

[16] S. M. Ambalakkat, "Optimization of constrained random verification using machine learning," Master's thesis, 2018.

[17] F. Wang, H. Zhu, P. Popli, Y. Xiao, P. Bodgan, and S. Nazarian, "Accelerating coverage directed test generation for functional verification: A neural network-based framework," in *Proceedings of the on Great Lakes Symposium on VLSI*, 2018.

[18] A. S. Jagadeesh, "Accelerating coverage closure for hardware verification using machine learning," Master's thesis, 2019.

[19] K. Laeufer, J. Koenig, D. Kim, J. Bachrach, and K. Sen, "Rfuzz: Coverage-directed fuzz testing of rtl on fpgas," in *International Conference on Computer-Aided Design (ICCAD)*, 2018.

[20] S. Canakci, L. Delshadtehrani, F. Eris, M. B. Taylor, M. Egele, and A. Joshi, "Directfuzz: Automated test generation for rtl designs using directed graybox fuzzing," 2021.

[21] S. Sokorac, "Optimizing random test constraints using machine learning algorithms," in *Proceedings of the design and verification conference and exhibition US (DVCon)*, 2017.

[22] J.-C. Ou, D. G. Saab, Q. Qiang, and J. A. Abraham, "Reducing verification overhead with rtl slicing," in *Proceedings of the ACM Great Lakes Symposium on VLSI*, 2007.

[23] Y. Guo, W. Qu, T. Li, and S. Li, "Coverage driven test generation framework for rtl functional verification," in *2007 10th IEEE International Conference on Computer-Aided Design and Computer Graphics*, 2007, pp. 321–326.

[24] L. Liu and S. Vasudevan, "Efficient validation input generation in rtl by hybridized source code analysis," in *Design, Automation Test in Europe (DATE)*, 2011.

[25] H. Witharana, Y. Lyu, and P. Mishra, "Directed test generation for activation of security assertions in rtl models," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, no. 4, 2021.

[26] Y. Lyu and P. Mishra, "Automated test generation for activation of assertions in RTL models," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[27] Y. Lyu, X. Qin, M. Chen, and P. Mishra, "Directed test generation for validation of cache coherence protocols," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, 2019.

[28] L. Liu, D. Sheridan, W. Tuohy, and S. Vasudevan, "A technique for test coverage closure using goldmine," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 5, 2012.

[29] M. Farkash, B. Hickerson, and M. Behm, "Coverage learned targeted validation for incremental hw changes," in *Proceedings of the Design Automation Conference (DAC)*, 2014.

[30] M. Benjamin, D. Geist, A. Hartman, Y. Wolfsthal, G. Mas, and R. Smeets, "A study in coverage-driven test generation," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 1999.

[31] M. Farkash, B. Hickerson, and B. Samynathan, "Mining coverage data for test set coverage efficiency," in *Design and Verification Conference (DVCON)*, 2015.

[32] D. Araiza-Illan, D. Western, A. Pipe, and K. Eder, "Coverage-driven verification—," in *Haifa Verification Conference*. Springer, 2015, pp. 69–84.