A GPT-2 Tabular Transformer Model for Automated DRAM Verification

Lorenzo Ferretti
Micron Technology
San Jose, California, USA
lferretti@micron.com

Surya Teja Bandlamudi
Micron Technology
San Jose, California, USA
suryatejaban@micron.com

Shailesh Sharma
Micron Technology
San Jose, California, USA
shaileshshar@micron.com

Chinmaya Behera
Micron Technology
San Jose, California, USA
cbehera@micron.com

Nihar Athreyas
Micron Technology
San Jose, California, USA
nathreyas@micron.com

Vikram Narayan
Micron Technology
San Jose, California, USA
vnarayan@micron.com

Samir Mittal
Micron Technology
San Jose, California, USA
samir@micron.com

***Abstract*—Design verification for complex designs like DRAM is a time-consuming and challenging task. Bus Functional Models (BFMs) play a crucial role in this process by emulating the behavior of various components and providing means to test the design functionality comprehensively. In this work, we propose a novel approach to accelerate the verification of DRAM hardware designs by leveraging a GPT-2 based Relational and Tabular Transformer (RTF) model. Unlike existing Deep Learning (DL) approaches that rely on reinforcement learning or adversarial training, our RTF model uses a conditioned tabular transformer architecture to learn correlations among plusargs and their values, which are then used to generate new unseen tests that target regions of the input space expected to maximize functional coverage. We employ a novel rarity score that quantifies the gap between the current and desired coverage, and use it to guide the test generation process. Experimental results on a legacy industrial DRAM design demonstrate that our approach achieves up to 7.89% coverage improvement compared to the currently employed DV flow, with a higher reliability compared to existing DL approaches. Our results highlight the potential of our RTF-based methodology to accelerate the verification of complex hardware designs.**

***Index Terms*—Design Verification, DRAM, Generative AI, Transformer Models**

I. Introduction

The verification of hardware designs, particularly for complex systems like DRAM (Dynamic Random-Access Memory), is a time-consuming and resource-intensive process. Bus Functional Models (BFMs) play a crucial role in this process by emulating the behavior of various components and providing a means to test the design functionality comprehensively. BFMs allow the evaluation of the impact of specific input sequences on a target Design Under Test (DUT), enabling the DUT to be exercised under different scenarios. Design Verification (DV) engineers control the generation of specific input sequences using plusargs to configure a Universal Verification Methodology (UVM) testbench. UVM testbenches offer a flexible way to configure and control the simulation environment through the plusargs passed during simulation runtime. Plusargs allow control on various simulation aspects such as choosing specific test sequences, enabling/disabling functionalities, controlling random seeds, or, more generally, affecting the DUT behavior.

DV engineers can generate many different test scenarios by exploring the plusargs' input space, with the goal of maximizing the functional coverage of the BFM. By maximizing the coverage goal, designers and DV engineers can ensure that the Design Under Test (DUT) is exercised extensively. However, the plusargs' input space is often too vast to be explored exhaustively. Therefore, human-designed combinations of plusargs and/or smart methodologies need to be employed to produce meaningful combinations of plusargs and valid tests.

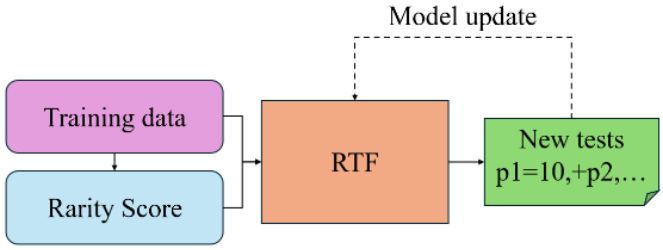

Fig. 1. RTF-based flow overview. Training data are leveraged to derive a rarity score. Both are then used to train a model and generate new tests.

This process is of fundamental importance when dealing with the verification of complex designs such as DRAM or NAND. In fact, by identifying the minimum set of plusargs that maximizes the functional coverage of the BFM, engineers can identify bugs and unintended behaviors early on, both at design and testbench level. Moreover, identifying the minimum set of plusargs that can maximize coverage within a given simulation budget has direct implications for downstream processes, particularly when physical simulations are performed at the gate level. These simulations are more computationally expensive and time-consuming compared to those performed with a BFM, and simulating the most effective combination of plusargs allows saving significant resources, time, and enables more extensive testing.

Machine Learning (ML) algorithms, and in particular Reinforcement Learning (RL), have been shown to be particularly effective in this task. State-of-the-art industry AI tools are currently employed in industrial settings [1] to navigate the plusarg space and produce a minimum set of tests to simulate at hardware level.

In this work, we propose a different approach, where we leverage a GPT-2 architecture to train a Relational and Tabular Transformers (RTF) model [2] to generate synthetic tabular data. By representing the input plusarg space in a tabular format, we can exploit the ability of tabular models to learn correlations among plusargs and their values in order to produce new unseen data following the training data distribution. Then, by leveraging contributing tests' information, obtained from past simulations, we control the test generation to target regions of the input space expected to maximize the coverage

based on the frequency of the design functionality being tested and a derived score (rarity score) from [3]. These synthetically generated tests can then be simulated, and their resulting coverage can be used to fine-tune the model over time.

The RTF methodology presented in this work leverages the flexibility of transformer models to generate synthetic tabular data without imposing any specific constraints on the input data. As long as the preprocessed data can be represented in a tabular format, the RTF model can transform the data into tokens and handle the generation accordingly, allowing for the creation of synthetic data tailored to the specific needs of design verification. Figure 1 provides an overview of the flow.

Our key contributions are:

1) A *rarity score* that quantifies coverage gaps based on coverbin hit frequency, enabling targeted test generation toward under-explored coverage regions.
2) A *conditional sampling mechanism* extending RTF with relational operators and thresholds to guide test generation toward desired rarity distributions.
3) Comprehensive *industrial evaluation* on a 497K DRAM design across 4 weeks, demonstrating up to 7.89% coverage improvement with higher reliability compared to the industry AI tool.
4) *Exploratory analysis* showing RTF discovers 1.7-2x more unique bins than the industry AI tool, enabling effective ensemble approaches.

II. Related Work

Different methods have been proposed in the past years leveraging AI to affect test generation. Most of these works rely on ML models, such as Bayesian Networks [4], [5], Genetic Algorithms [6], and Neural Networks [7], to estimate the impact of existing tests and plusargs to identify the most promising ones and avoid redundant simulations by influencing the Constrained Random Verification (CRV) space controlled by the specific plusargs.

Recently, Huang et al. [8] proposed Multi-Armed Bandit (MAB) regression to automatically explore the test plusargs with the goal of maximizing the coverage. This approach leverages Bayesian MAB to identify and select the best test configuration to simulate without any knowledge of the DUT. Similarly, Jayasree et al. [9] proposed employing decision trees to estimate test coverage and simulation time and to schedule new regressions. Ferretti et al. [3] recently proposed a similar approach that uses model predictions to create new regression batches. Unlike earlier methods, their work incorporates DUT properties and hierarchical information to guide test selection, and it automatically generates optimal batches to reduce overall simulation time. In this work, we leverage the notion of rarity score, proposed by Ferretti et al. [3] in their predictor model, to define target classes for our model. These classes are used as labels for the synthetic generation of new tests.

Recent advancements in deep learning, including generative adversarial networks [10], autoencoders [11], language models [12], and diffusion models [13], have been shown to be particularly effective when dealing with tabular data. Studies have shown that these deep learning models can produce more realistic data compared to traditional methods like Bayesian networks. Existing models, based on Hierarchical Modeling Algorithms and Gaussian Copulas, can synthesize data but often fail to capture the nuanced conditions within and across tables accurately. To address these limitations, Gong et al. [14] introduced TabGPT, which uses autoregressive transformers, specifically GPT, to generate synthetic transactional data of arbitrary length. Despite its effectiveness, TabGPT requires training independent models for each user, making it impractical for real-world applications. To overcome this, Solatorio et al. [2] propose a sequence-to-sequence framework that generalizes the use of GPT for generating arbitrary-length synthetic data conditioned on an input, offering a more scalable solution for synthetic data generation.

In this work, we leverage the REaLTabFormer (RTF) framework [2] by representing the test input space in a tabular format and generating new tests. Further, we have enhanced the model to use the rarity score from [3] to condition the test generation to chase uncovered regions and uncovered functionalities of our design under test.

III. Problem Formulation

The goal of our methodology is to identify an optimal set of tests that maximizes a target coverage metric, namely functional coverage, for a given budget of simulation. Given the vast scope of the input space and the complexity of RTL designs, achieving 100% coverage is often impractical, requiring many time-consuming iterations. Additionally, due to project deadlines, the number of simulations that can be performed is often constrained by a predefined budget or timeframe. In this study, we define the maximum attainable coverage, or coverage closure, as the coverage obtained after incorporating all user-specified exclusions. Similarly, our simulation budget is a user-defined number of simulations decided by the DV engineer, performed using a BFM. This practice is commonly adopted by DV engineers, who typically run BFM simulations to identify a minimum set of contributing tests, which are then used for gate-level simulations in further downstream verification phases.

Verification engineers create a set of different tests to exercise specific design functionalities. Moreover, they define plusargs (P) to parametrize the test execution and affect a specific test functionality. The simulation of a test and its associated set of plusargs generates a coverage report. The coverage report can be represented as a vector cov of integers that describe the coverage obtained for each functional coverbin or group. Coverbins and covergroups are usually manually created by DV engineers to represent specific portions of the design being executed or functionality being exercised. From the coverage vectors, the cumulative coverage can be derived. This refers to the aggregated measurement of the simulated tests over time or across multiple simulation runs, representing the extent to which the design's functionality and scenarios have been tested. By post-processing the cumulative coverage, it is also possible to derive the set of contributing tests, denoted as TC. Contributing tests are defined as the minimum number of tests required to be simulated to replicate the maximum attained coverage.

IV. Methodology

The proposed methodology is divided into four main phases: Data Preparation, Model Training, Conditional Sampling, and Simulation. Data from past simulations are leveraged to collect plusargs information and coverbins hit frequency information. This information is used to derive a score, which is then used to identify target bins by defining rarity groups. Then, once input tests and scores are extracted from the original data, a tabular representation of tests and scores is used to train the RTF model. Following model training, synthetic test samples are produced. A conditional layer is subsequently employed to steer the generative process, ensuring that the output conforms to a predefined distribution characterized by the target bins. Lastly, for each new batch of tests generated, a new regression is triggered, and the process is repeated until a user-defined budget of simulations has been reached.

Figure 2 provides an overview of the flow and its different phases. Each phase is described in detail in the following subsections.

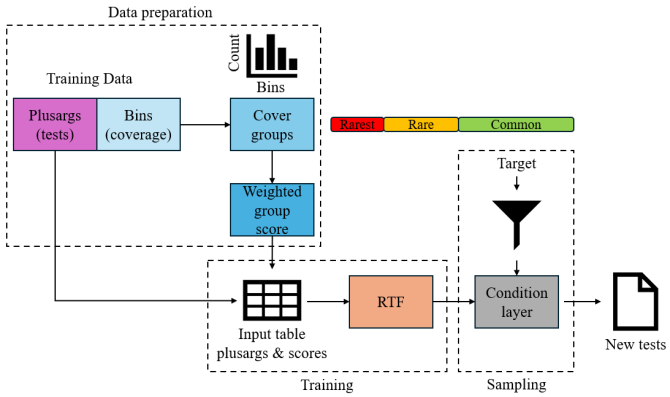

Fig. 2. RTF-based flow and its different components. Data preparation processes the data for the Training phase. New tests are produced after the training phase, leveraging a conditional layer.

A. Data Preparation

The process begins by selecting contributing tests derived from simulation results obtained over the past weeks. This data is generated by combining the simulations from different weekly runs. Each prior week's simulation data is merged, and the contributing tests from the past weeks are extracted consecutively. Each test is associated with a set of plusargs that uniquely define the test. In practice, the tests may contain missing values or outliers, which can lead to suboptimal model performance. To address these issues, we employ various data preprocessing techniques, including imputation, smoothing, and data cleaning methods, to ensure that the input data is accurate and reliable. For each such test, coverage information at a coverbin level is obtained from its simulation. For each coverbin, we store the coverage information with respect to each test. This provides us with the frequency of coverage for a given coverbin b across a set of tests t. The coverage data is stored as a vector. For each test, the coverage vector indicates the bins hit during the test simulation. From the hit frequency, we derive different groups of bins:

- Rarest: bins hit by 150 tests or fewer, including bins never hit ($n \leq 150$),
- Rare: hit by more than 150 tests but fewer than 400 tests ($150 < n \leq 400$),
- Common: hit by more than 400 tests ($400 < n \leq 1000$).

The definition of these groups is currently based on empirical observation. Based on this grouping, we can determine if each coverbin belongs to the rarest, rare, or common group, and calculate the rarity score as follows:

$$\text{RarityScore} = \frac{\#\text{ of bins hit by the test in group } i}{\text{Total number of bins}} \quad (1)$$

where $i \in G$, i identifies the group, and G is the set of all groups $G = \{Rarest, Rare, Common\}$. The rarity score within the plusargs are then used to define the input data for the RTF Training phase.

B. Model Training

We formulate our problem for the RTF model by representing tests' plusargs in a tabular format. Once the rarity score for each test is calculated, this information is concatenated with the plusargs defining the test, such that each test would have three columns representing the groups with their corresponding rarity scores. Our input data is a matrix $M_{n\times p}$ where n is the number of contributing tests, and p is the number of plusargs. After concatenating the rarity score to such a matrix, we obtain a new matrix $M'_{n\times(p+1)}$. The modified plusargs table, now containing rarity scores, is used to train the RTF model.

RTF is a transformer-based framework designed to generate non-relational tabular data using an autoregressive model, and relational tabular data through a sequence-to-sequence (Seq2Seq) architecture. Tabular-specific strategies are used to encode columns and row data within statistical methods for detecting overfitting and a constrained sampling strategy for data generation. An autoregressive GPT-2 model is employed to generate synthetic observations (tests) for non-relational tabular data. Each observation, or test, is treated as a sequence of tokens, where each cell in the row is represented as a set of unique tokens corresponding to the respective plusarg and its values, allowing the model to learn the conditional distribution of plusarg values. Numerical data are transformed into a text-based format to optimize training and sampling efficiency. This involves rounding, string conversion, padding, and tokenization. Categorical data, such as unique values, are treated as unique tokens. If missing values are present (i.e., a plusarg is not used in a test), no transformation is applied, and the model learns the distribution of missing values.

The RTF methodology relies on a unique way of representing numbers and label-encoded values. For categorical data, each number is split into n-parts based on the amount of precision required to represent each part in the latent space. Then, each representation is combined with the respective plusarg. For example, the same plusarg value associated with two different plusargs would be represented by different tokens. This enables the model to learn the sequence of a particular set of tokens while learning the respective distributions using attention.

During training, regularization is performed to minimize the impact of outlier observations and overfitting. Similarly, distance to the closest record and quantile difference statistics are used to detect overfitting, and bootstrapping is employed to guide early stopping during training.

Upon completion of training, the model can be leveraged to produce new synthetic tests reflecting the training data distribution. The synthetic tests are a combination of plusargs following the contributing test data distribution, but at the same time different from the original training set. This allows us to generate new unseen tests while preserving important information, such as correlation among plusargs, valid and invalid combinations, and acceptable input ranges, etc.

C. Conditional Sampling

We sample desired plusarg combinations based on input constraints according to the notion of Rarity Score. The new samples created are intended to explore all possible plusargs combination within the valid plusargs' value range and according to the learned plusarg distribution. However, to achieve coverage improvement, we skew the sampling distribution, targeting unreached portions of the functional coverage landscape. To achieve this, we have implemented a conditional sampling layer that enhances the training data, represented as plusargs, rarity scores, and additional attributes: a relation column and a threshold column. The relation column represents logical operators (such as: equality, greater/lower than, etc.) and is used in combination with the threshold column. The aim of these two columns is to represent the distribution of relational operators and thresholds with respect to plusargs combinations and rarity scores. The two additional columns reshape our input matrix into a $M'_{n\times(p+3)}$. Therefore, during the RTF training phase, the model can learn the associated distribution of plusargs combinations, thresholds, and rarity scores for

a given relational operator. When sampling, the relational condition and the threshold are passed to the model in order to generate only plusargs combinations expected to have a rarity score greater than a desired threshold. Different conditional operators can be passed to the conditioning layer according to the user's needs, allowing them to sample from different rarity score groups. Conditional sampling relies on the existence of rarity scores; without this score, it is not applicable, and hence, the model's performance cannot be evaluated separately. Therefore, in this work, we adopt a hierarchical rarity score definition where bins are assigned to three categories: rarest, rare, and common, based on their frequency.

In our scenario, we sample new tests by conditioning the model to select tests that are more likely to belong to the rarest group rather than to other groups. Thus, the model can be directed to produce new tests that are more likely to reach rarely hit bins rather than focus on the more common ones. The actual distribution of the sampling from the different groups can be directly controlled by the user providing group-specific thresholds.

D. Simulation

Once the new batch of tests is sampled, the tests are simulated, and the coverage information is extracted. After each batch of simulation is completed, the coverage outcome of the simulated data is added to the training corpus; the model is then re-trained using the same process described above. This cycle is repeated until a user-defined number of simulations is reached.

V. Experiments

A. Experimental Setup

We have implemented and tested our methodology targeting a legacy industrial DRAM design. The design has roughly 497K functional bins. After applying user-defined exclusions for unreachable states and infeasible scenarios, these bins represent the coverage space we aim to maximize within our simulation budget. The compared DV flow used the industry AI tool to generate weekly simulations leveraging training data from the past week. The industry AI tool flow produces an average of six different runs, each consisting of one-thousand simulations, per week. Then, the best run among the industry AI tool runs and the training batch is selected for hardware simulation. The selected run is also used for the following week's training. We tested our methodology following a similar approach. The same prior week coverage information used by the industry AI tool has been provided to the RTF model for training, and a different batch of simulations has been produced leveraging our flow. The industry AI tool flow used for comparison in this work has been recently published in [1], where it was used to test designs in a similar setting. We have replicated the same flow and adopted it as a comparison in our evaluation.

Initially, contributing tests are collected from historical data. Based on the frequency of bins hit for a given set of tests, rarity scores for each of the groups (Rarest, Rare, and Common) are calculated. These scores, along with plusargs, are used to train the RTF model. Upon training, a batch of new tests x_1 are generated using conditional sampling, which generates tests with rarity scores greater than the threshold for each given rarity group G, where the threshold is defined by the user according to the exploration needs. In the current implementation, 70% of the samples are expected to belong to the Rarest group, 10% to the Rare group, and 20% to the Common one. These numbers were chosen based on empirical observation, as observed in Figure 3. Lastly, the sampled tests produced from

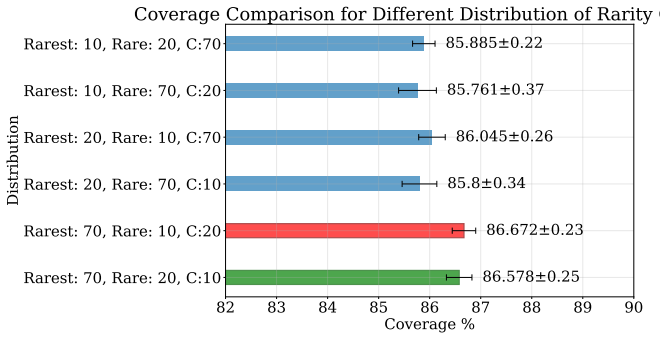

Fig. 3. Empirical exploration of sampling percentage of each rarity group. The red and green bars are the best performing ones, averaged across 5 repetitions. Sampling distributions are indicated as Rarest, Rare, and Common (C).

RTF are adjusted to factor out randomness to avoid getting stuck in local optima. Currently, 10% of tests are randomly generated.

Then, the newly generated batch of plusargs is simulated using the BFM, and a new set of coverage vectors are obtained. The new batch, including the coverage information, is added to the initial train data, and rarity scores are re-calculated. The RTF model is re-trained on the new data to generate a new batch x_2.

The process is repeated r times until the x_i batch is produced, or a given budget of simulation B is reached. For the following weeks, training data are prepared by combining the initial contributing tests from historical data with contributing tests from the newly generated batches. The RTF training and simulation process is repeated to generate a budget for a given number of repetitions. Following this pattern, we generate a set of new plusargs across different weeks, aiming at increasing the overall set of contributing tests, leveraged to train the model, and consequently, increase the overall coverbin reachability and coverage.

We have generated batches of 200 tests each, and we have repeated the process for a total budget of 1,000, 2,000, and 3,000 tests. For the different RTF flow's budgets, we have produced 6, 3, and 2 different runs, respectively, in order to produce the same number of simulations adopted by the industry AI tool-for a total of 6,000 simulations.

The RTF model was trained using a GPT-2 architecture with standard hyperparameters optimized for tabular data generation. Training was performed on GPU hardware with early stopping based on overfitting detection metrics. The model processes plusargs as tokenized sequences, learning conditional distributions through self-attention mechanisms. Regularization techniques including bootstrapping and distance-to-closest-record statistics were employed to ensure robust generalization.

B. RTF Model Performance

We have measured the ability of the RTF model to learn plusargs and their legal value distributions using statistical similarity between the real and synthetic data generated for each single column of data. This metric, also referred to as the marginal distribution of each column, captures aspects such as skewness, modality, and central tendency of a column value distribution. A high score indicates that the synthetic data effectively replicates the distribution features of each column. Likewise, similarity between real and synthetic data for pairs of columns is measured, often referred to as correlation or bi-variate distributions of the columns. The observed marginal distribution score across columns resulted in a score of 96.15%, while the column correlation score resulted in 97.11%, highlighting the effectiveness of our trained model in learning from the original data.

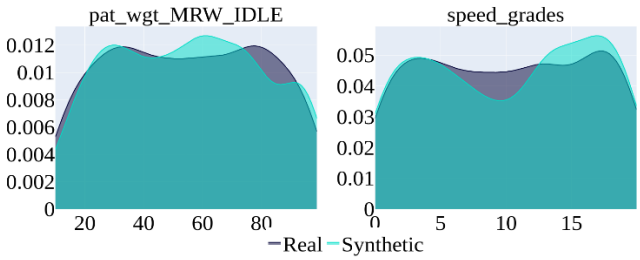

Fig. 4. Example of real and synthetic data distribution learned from the RTF model for plusargs as an example `pat_wgt_MRW_IDLE`, and `speed_grades`.

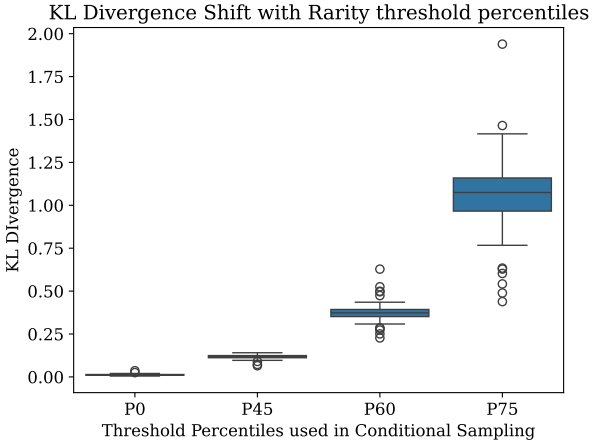

Fig. 5. Impact of threshold values on Conditional Sampling, for training data (P0) and different percentiles (45th, 60th, and 75th).

Figure 4 shows the real and synthetic data distribution learned for two plusargs: a numerical plusarg `pat_wgt_MRW_IDLE`, and a categorical plusarg `speed_grades`. The two resulted in marginal distribution scores of 97.4% and 93.4% respectively, as shown by the resembling shapes of the real and synthetic distributions in the plots.

In addition, we measured the Kullback-Leibler (KL) divergence with respect to the training data and different sampling thresholds. Figure 5 shows this analysis. In the plot, we considered P0, representing the original training data, P45, representing the 45th percentile, P60, representing the 60th percentile, and P75, representing the 75th percentile. These thresholds were chosen based on empirical observations, which revealed an exponential increase in KL divergence values as the sample size for higher percentiles decreases, leading to a distribution that skews further away from the original training data. The results, averaged over five different samples from the model, show a low KL divergence score for the P0 data, indicating that the model can accurately reproduce the original training data distribution. As the sampling threshold increases, the model begins sampling from specific regions in the high-dimensional plusargs space, resulting in higher KL divergence values. This behavior suggests that the model effectively identifies and reproduces these specific regions, enabling it to target groups with a higher chance of reaching hard-to-hit bins, thus enhancing the coverage of the test configurations.

Lastly, we have evaluated the effectiveness of the conditioning layer to skew the test generation in order to produce the desired rarity score distribution. Figure 7 shows the results of the model training after conditional sampling. We sampled new tests with the goal of targeting `rarest`, `rare`, and `common` bins. For each of the rarity scores, the figure shows the RTF model-learned distribution from the training data (blue), the predicted data distribution for the

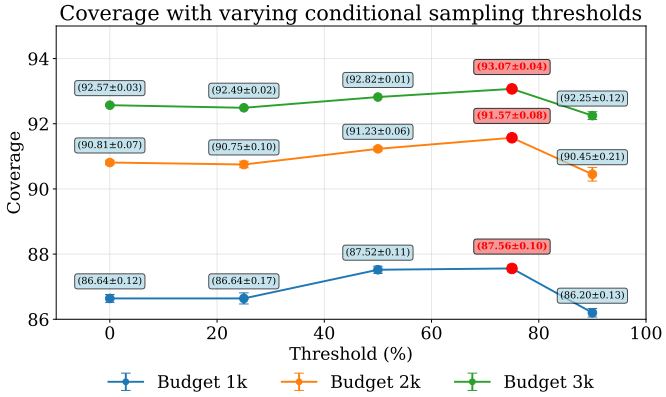

Fig. 6. The model was sampled using varying thresholds as conditions, then simulated to collect coverages. Across 1K, 2K and 3K budgets, 75% of the datasets showed better performance.

TABLE I
RESULTS COMPARISON BETWEEN CURRENT INDUSTRY-TOOL FLOW AND THE PROPOSED RTF-BASED METHODOLOGY.

Methodology	# Contributing Tests	Coverage %	# Bins Hit	New Bins Hit
Industry-Tool	990	85.34	424183	1024
RTF-1K	997	87.68	435817	1722
RTF-2K	1844	91.66	455601	3264
RTF-3K	2501	93.23	463403	4174
Industry-Tool+RTF-1K	1828	90.92	451903	2503
Industry-Tool+RTF-2K	2443	92.90	461751	3791
Industry-Tool+RTF-3K	2906	93.90	466707	4758

new samples (green), and the actual ground truth distribution (red) obtained after the simulations. However, to confirm the effectiveness of the conditioning layer, a more detailed analysis of the distribution and its correlation with the rarity scores is necessary. The plots show a clear learning trend for conditional sampling across repetitions. For each rarity group—rarest, rare, and common—the plots demonstrate that the skewness of the predicted data distribution closely mirrors that of the ground truth distribution. This indicates that the conditioning layer effectively steers the test generation process towards the desired rarity score distribution. While the RTF model does not look into the numeric values due to the special embedding required by the RTF architecture [2], it learns the joint distributions across tokens effectively. This is particularly evident for the rarest and rare groups, where the uniqueness of the plusarg combinations with respect to the rarity groups is more pronounced. The right condition threshold was selected using another empirical study. The model was sampled using different threshold conditions, which were then simulated to fetch the coverage values. Figure 6 clearly shows that at each budget, the threshold value of 75% performed better. The model's ability to capture these nuanced distributions highlights its capability to generate meaningful and targeted test cases.

C. Coverage Improvement

Table I shows a detailed summary of the results obtained by the industry AI tool, and by the RTF model using different batch budgets, for the same number of simulations. The results reported in the table were collected over the design and testbench for the same week, and the same initial data were used to train the models. We can observe how the proposed RTF flow is able to achieve higher coverage than the industry AI tool baseline, obtaining a 2.34%, 6.32% and 7.89% coverage improvement for the RTF-1K, RTF-2K, and RTF-3K respectively; this is reflected in the number of bins hit. The coverage reported is the maximum coverage obtained across each of the different runs.

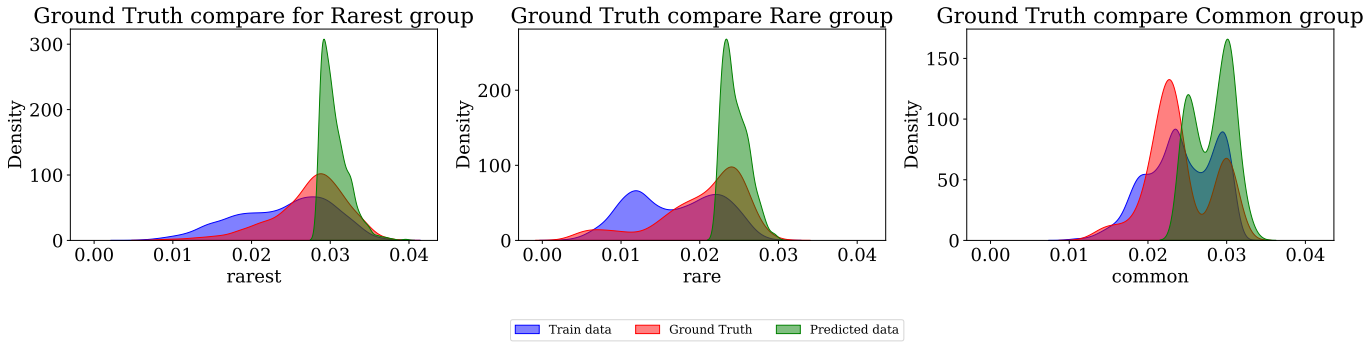

Fig. 7. Ground truth comparison of Groups with Prediction and Training. Distributions of rarity scores for `Rarest`, `Rare`,and `Common` respectively for the training and predicted rarity scores compared with the output rarity scores post simulation. The closer the ground truth is to the predicted data, the better is the model at predicting coverage.

Similarly, Figure 8 shows a detailed summary of the results obtained by the industry AI tool, and by the RTF model over four different weeks during which both the design and the testbench may have changed due to the live dynamics of the hardware design process. Across weeks, apart from the first week, where both the flows have access to the same information, the industry AI tool flow leverages the contributing tests discovered by the industry AI tool flow over the previous weeks, while the RTF flow leverages the contributing tests obtained by the RTF explorations. From the results, it is possible to observe how overall the RTF flow consistently outperforms the industry AI tool across all the weeks. At the end of the four weeks, the RTF flow achieves 4.2%, 7.8%, and 9.2% average coverage increments using RTF-1K, RTF-2K, and RTF-3K respectively, w.r.t. the industry AI tool. Moreover, it is also possible to observe how, compared to the industry AI tool flow, the RTF versions present a smaller standard deviation, resulting in higher reliability compared to the industry AI tool.

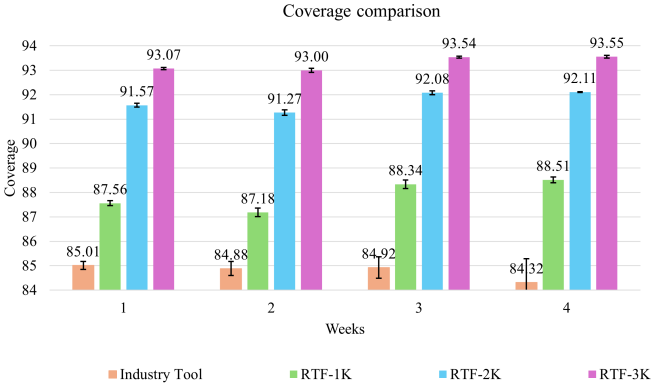

Fig. 8. Functional coverage comparison among Industry-Tool and our RTF-flow variations for the same total budget of simulations (six thousand). The results highlight the ability of our RTF-flow to attain both higher coverage and lower standard deviation across the different weeks.

D. Test Effectiveness

In addition to measuring the coverage at the end of each regression across different weeks, we have also evaluated the ability of the different flows to discover new bins with respect to the initial training data. The test effectiveness allows us to evaluate the different flows based on their effectiveness in exploring the search space rather than focusing on already identified regions by reusing the learned distribution. Note that, apart from the first week where the same training data are used, starting from week two the flows will rely on the training data derived from the prior week's exploration.

Figure 9 illustrates the cumulative discovery of new bins over the four-week evaluation period. The stacked bar representation shows how each week's exploration contributes additional coverage, with the RTF variations consistently discovering more unique bins compared to the industry AI tool across all budget configurations.

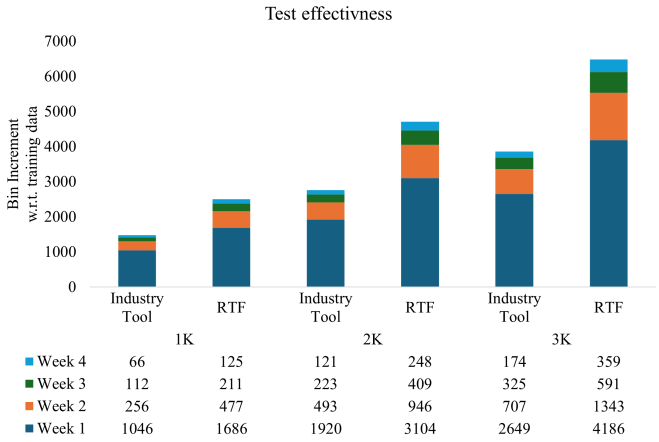

Fig. 9. Test effectiveness across the four weeks among the Industry-Tool and RTF flow for different budgets. The stacked bar plot highlights the flows' contributions across weeks. After the first week, our RTF flow is already able to outperform the coverage obtained by the industry AI tool at the end of the four weeks.

E. Model Ensemble

Lastly, we have evaluated the contribution of our RTF-based methodology when combined with the industry AI tool. In this setup, we have merged the results obtained by both the industry AI tool and RTF flow, while maintaining the same simulation budget. Moreover, in this scenario, multiple industry AI tool runs have been merged to enable comparison among the same number of simulations.

Figure 10 shows the Venn Diagram for the industry AI tool flow and the three RTF variations. RTF is able to discover a significantly larger number of unique scenarios, with 1.7x, 1.9x, and 2x unique bins discovered, respectively, with respect to the industry AI tool for the 1-K, 2-K, and 3-K budgets. Nonetheless, the industry AI tool is also able to contribute to coverage improvement, as shown by the unique bins discovered. In fact, merging the two flows enables the achievement of 90.92%, 93.62%, and 94.65% coverage, respectively. The goal of this analysis is to observe the ability of the different flows to effectively target different regions of the search space. Due

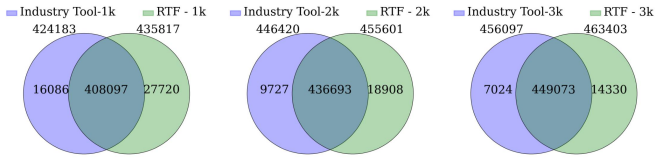

Fig. 10. Unique tests contribution analysis among Industry-Tool and our RTF-based flow. Our methodology consistently discovers a larger number of bins.

to the different nature of the exploration strategies, the flows target different regions of the search space. Thus, by adopting an ensemble of the two models, we can gain the benefit of the different flows.

VI. Conclusion

The proposed methodology presents a comprehensive approach to accelerate the verification of DRAM hardware designs by generating new tests using a conditioned transformer model architecture. By leveraging the strengths of transformer models and incorporating a novel rarity score feature, this technique provides a targeted and efficient way to ensure thorough testing and validation of complex hardware systems, achieving up to 7.89% coverage improvement with respect to the currently employed design verification flow.

A. Limitations and Future Work

While our results demonstrate clear improvements, several limitations should be acknowledged. Our evaluation is limited to a single legacy DRAM design; future work should validate the approach on other DRAM generations and memory types to assess broader applicability. The rarity group thresholds ($\leq$150, 150-400, $\geq$400) were empirically chosen for this design, and future work should explore dynamic threshold adaptation based on current coverage state or percentile-based definitions for better generalization across designs. Additionally, comprehensive comparisons to random sampling baselines and other ML approaches (TabGAN, genetic algorithms) would provide more complete performance characterization.

Promising future directions include integration with formal verification, online learning to adapt thresholds during test generation, multi-objective optimization balancing coverage and simulation time, and extension to other verification domains beyond memory designs.

References

[1] G. Pallela, P. Shi, S. Deshmukh, V. Venkatesh, B. Hughes, R. Suvarna, and S. Srinivasan, "Stimulus diversification and coverage closure of 3d nand flash with artificial intelligence," in *Proceedings of the design and verification conference and exhibition US (DVCon)*, 2025.

[2] A. V. Solatorio and O. Dupriez, "Realtabformer: Generating realistic relational and tabular data using transformers," *arXiv preprint arXiv:2302.02041*, 2023.

[3] L. Ferretti, C. Behera, S. T. Bandlamudi, N. Athreyas, V. Narayan, and S. Mittal, "A scalable gray-box instance-based reachability predictor for automated dv regression scheduling," in *Proceedings of the design and verification conference and exhibition US (DVCon)*, 2025.

[4] S. Fine and A. Ziv, "Coverage directed test generation for functional verification using bayesian networks," in *Proceedings of the 40th Annual Design Automation Conference*, DAC '03, p. 286–291, 2003.

[5] 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 (IEEE Cat. No.04EX940)*, pp. 75–80, 2004.

[6] M. Imková and Z. Kotásek, "Automation and optimization of coverage-driven verification," in *2015 Euromicro Conference on Digital System Design*, pp. 87–94, 2015.

[7] 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 2018 Great Lakes Symposium on VLSI*, GLSVLSI '18, (New York, NY, USA), p. 207–212, Association for Computing Machinery, 2018.

[8] Q. Huang, H. Shojaei, F. Zyda, A. Nazi, S. Vasudevan, S. Chatterjee, and R. Ho, "Test parameter tuning with blackbox optimization: A simple yet effective way to improve coverage," in *Proceedings of the design and verification conference and exhibition US (DVCon)*, 2022.

[9] V. Jayasree, "Machine learning for coverage analysis in design verification," in *Proceedings of the design and verification conference and exhibition Europe (DVCon)*, 2021.

[10] Z. Zhao, A. Kunar, R. Birke, and L. Y. Chen, "Ctab-gan: Effective table data synthesizing," in *Proceedings of The 13th Asian Conference on Machine Learning* (V. N. Balasubramanian and I. Tsang, eds.), vol. 157 of *Proceedings of Machine Learning Research*, pp. 97–112, PMLR, 17–19 Nov 2021.

[11] S. Darabi and Y. Elor, "Synthesising multi-modal minority samples for tabular data," *CoRR*, vol. abs/2105.08204, 2021.

[12] V. Borisov, K. Seßler, T. Leemann, M. Pawelczyk, and G. Kasneci, "Language models are realistic tabular data generators," 2023.

[13] A. Kotelnikov, D. Baranchuk, I. Rubachev, and A. Babenko, "Tabddpm: modelling tabular data with diffusion models," in *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, JMLR.org, 2023.

[14] H. Gong, Y. Sun, X. Feng, B. Qin, W. Bi, X. Liu, and T. Liu, "TableGPT: Few-shot table-to-text generation with table structure reconstruction and content matching," in *Proceedings of the 28th International Conference on Computational Linguistics* (D. Scott, N. Bel, and C. Zong, eds.), (Barcelona, Spain (Online)), pp. 1978–1988, International Committee on Computational Linguistics, Dec. 2020.