2025 10th International Conference on Computer Science and Engineering (UBMK) | 979-8-3315-9975-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/UBMK67458.2025.11206764

AI-Driven Synthetic Test Data and Scenario Generation via GAN-LLM Integration: A Modular Approach for Web Application Testing

Miraç Emektar
Virgosol
İstanbul, Türkiye
mirac.emektar@virgosol.com

Fatih M. Harmancı
Customer Operations
Virgosol
İstanbul, Türkiye
fatih.harmanci@virgosol.com

Aysun Güran
Department of Computer Engineering
Doğuş University
İstanbul, Türkiye
adogrusoz@dogus.edu.tr

Berat Karadavut
Department of Computer Engineering
Doğuş University
İstanbul, Türkiye
202103001042@dogus.edu.tr

Günay Kirmizi
Department of Computer Engineering
Doğuş University
İstanbul, Türkiye
202103001059@dogus.edu.tr

Berkay Öztürk
Department of Computer Engineering
Doğuş University
İstanbul, Türkiye
202103001046@dogus.edu.tr

Ata Emir Karamuk
Department of Computer Engineering
Doğuş University
İstanbul, Türkiye
202003001068@dogus.edu.tr

Mert Güven
Department of Computer Engineering
Doğuş University
İstanbul, Türkiye
202003001044@dogus.edu.tr

***Abstract*—The growing complexity of web applications has heightened the need for automated and adaptive testing solutions. Manual test case authoring and static data generation often fall short in handling dynamic user interfaces and form-driven workflows. In this paper, we present a modular test automation architecture that integrates Generative Adversarial Networks (GANs) and Large Language Models (LLMs) to enable end-to-end generation and execution of test scenarios. The system utilizes a LLaMA4-based LLM to semantically extract input fields from web forms and guide both data and scenario generation. CTGAN (Conditional Tabular Generative Adversarial Network) is used to synthesize realistic, context-aware tabular test data, while SeqGAN (Sequence Generative Adversarial Network) produces logically ordered user interaction steps. These outputs are executed in a browser using Selenium WebDriver to simulate real-time user behavior. The architecture supports both contextual (URL-based) and non-contextual (prompt-based) modes and allows individual modules to be used independently or as a unified pipeline. The system also includes a graphical user interface to facilitate test configuration and execution. Experimental results across five public web applications show high format compliance in synthetic data (95.05 percent) and strong scenario validity, with an average automation success rate of 84.6 percent. The proposed approach enhances test coverage while significantly reducing manual effort, offering a scalable and privacy-aware solution for modern QA workflows.**

***Keywords*—Synthetic Data Generation, Test Automation, Large Language Models, Scenario-based Testing.**

I. Introduction

Software testing remains one of the most critical and resource-intensive phases in the software development lifecycle. As systems become increasingly complex and data-driven, the need for diverse, high-quality test data and realistic test scenarios has grown significantly. In addition to this, achieving broader test coverage requires increased test data diversity—particularly for complex and data-driven systems where edge cases and variability play a critical role. Traditional testing approaches, which rely heavily on manually written test cases and static test data, often fall short in terms of coverage, efficiency, and reproducibility [1], [2]. Particularly for web applications, where form inputs, navigation logic, and user interaction patterns vary significantly, test automation faces critical limitations when scenario and data generation are decoupled from system context.

To address these challenges, artificial intelligence (AI)-driven techniques have emerged as promising solutions in the field of automated software testing. In addition, synthetic data generation has become foundational for developing tools that comply with privacy regulations—such as the Turkish Personal Data Protection Law (KVKK)—by ensuring the separation of production and test environments. Among these, GANs have demonstrated strong potential in producing synthetic data distributions that closely resemble real-world data [3]. In particular, CTGANs are designed to generate high-fidelity tabular datasets by conditioning on discrete features—a key advantage when simulating user records such as names,

emails, and passwords in test environments. Similarly, SeqGANs have been adopted for generating sequential outputs, such as ordered test steps or user action flows, using reinforcement learning to manage temporal dependencies [4].

Despite these strengths, GANs inherently lack domain context awareness [17]. Without understanding the semantics of the input fields or the logic of the user interaction, synthetic outputs can be unrealistic or misaligned with the system under test. This limitation has motivated the integration of Large Language Models (LLMs)—such as OpenAI's GPT-4, Meta's LLaMA, and Google's PaLM—which offer advanced natural language understanding and context modeling capabilities [5]–[7]. When paired with GANs, LLMs can serve as intelligent controllers: extracting form fields from web pages, inferring test intentions, and guiding data and scenario generation based on semantic understanding.

This paper proposes a modular and AI-driven approach to synthetic test data and scenario generation by integrating GAN and LLM architectures into a cohesive automated testing pipeline. Specifically, the system leverages LLaMA4-based LLMs to extract and understand form structures from a given website. CTGAN is then used to generate context-aligned tabular test data, while SeqGAN creates logical test step sequences. These outputs are automatically executed using Selenium WebDriver to simulate real browser interactions.

In contrast to prior work that focuses separately on either synthetic data generation or scenario modeling, our framework bridges the two within a unified pipeline. The result is a highly flexible and scalable system that enables end-to-end test automation, even in the absence of real-world data. Furthermore, the solution is packaged with a user-friendly graphical interface and supports both contextual (website-based) and non-contextual (prompt-based) modes of operation.

The remainder of this paper is organized as follows. Section II discusses related research in GAN-based data generation, LLM-assisted testing, and integrated AI testing tools. Section III describes the proposed system architecture in detail. Section IV presents experimental results and evaluations conducted on real websites. Finally, Section V concludes the paper with key findings and future research directions.

II. Related Work

Recent advancements in generative models and language understanding have greatly influenced the software testing domain, particularly in the context of synthetic data generation and automated scenario design [18]. This section reviews prior research related to (i) GAN-based data generation, (ii) sequence generation via GANs, (iii) LLM-driven test automation, and (iv) integrated AI testing systems.

A. Tabular Data Generation with GANs

GANs, first introduced by Goodfellow et al. [8], have gained significant traction in areas such as image synthesis, anomaly detection, and data augmentation. However, the generation of structured tabular data has proven more challenging due to the presence of mixed-type columns (categorical and continuous), complex interdependencies, and sparsity in real-world datasets.

CTGAN, proposed by Xu et al. [3], addresses these issues by using mode-specific normalization and conditional vectors to generate high-quality samples from tabular datasets. Empirical evaluations have shown CTGAN to outperform traditional oversampling techniques like SMOTE or random perturbation in maintaining statistical properties of the original data distribution [9][19].

In the context of software testing, tabular data generation is crucial for simulating user profiles, login credentials, payment records, and other structured inputs required for form-based systems. Despite its effectiveness, CTGAN alone lacks semantic awareness of field context, which limits its usability in dynamic UI scenarios.

B. Sequence Generation with SeqGAN

SeqGANs have been widely applied in natural language processing tasks, such as poetry generation, dialogue modeling, and music composition. SeqGAN, introduced by Yu et al. [4], integrates reinforcement learning with adversarial training to handle the discrete nature of textual outputs. Instead of backpropagation through discrete tokens, SeqGAN uses a Monte Carlo rollout mechanism to estimate intermediate rewards and guide training.

In software testing, especially in GUI-driven or workflow-based applications, test scenarios are essentially sequences of actions. These include user steps such as logging in, navigating menus, or submitting forms. Prior research has demonstrated that sequence-based generative models can effectively learn the underlying patterns of such interactions [10].

C. Language Models in Test Case Generation

Large Language Models (LLMs) such as GPT-3 [5], PaLM [6], and LLaMA [7] have significantly advanced the state of the art in machine understanding and generation. Their applications in software engineering include code completion, bug fixing, and recently, test case generation.

Mozannar et al. proposed a framework where language models could be trained to reason about software logic and generate test assertions [11]. Similarly, Nasir and Wang developed a system that utilized LSTM-based networks for extracting and labeling web form elements to assist in automated testing [12]. More recently, BEAST [13] introduced semantic-aware test generation for deep learning models, leveraging LLMs to identify meaningful test boundaries.

However, most of these approaches either focus solely on generating test descriptions or require extensive prompt engineering. The lack of integration between LLMs and execution frameworks limits their direct applicability in automated test environments.

D. Hybrid and AI-Augmented Testing Systems

Several academic and industrial tools aim to automate various components of the testing pipeline. For example, CrashScope [14] enables automatic GUI testing of Android applications through static and dynamic analysis. TESTAR

[15] applies model-based testing techniques to generate and execute random UI actions on desktop applications.

Commercial platforms such as TestRigor and Mabl use AI to interpret natural language test scripts and automate web testing. However, these systems often rely on either pre-defined templates or human-authored scenarios and are limited in their ability to generate synthetic data from scratch.

Notably, these tools lack a combined approach that uses both generative modeling for data and deep language understanding for scenario composition in a single modular framework. Our work aims to fill this gap by integrating CTGAN and SeqGAN with LLMs in an end-to-end automated testing pipeline, thus bridging data generation, scenario design, and execution within a unified architecture.

III. System Architecture

The proposed system introduces a modular architecture for the automatic generation and execution of test cases by combining three AI components—LLMs, GANs (CTGAN and SeqGAN), and an automation engine (Selenium). It is designed to operate in both contextual (website-based) and non-contextual (prompt-based) modes, enabling testers to simulate realistic user interactions and validate form-based web applications with minimal human intervention.

An overview of the system architecture and data flow is presented in Figure 1.

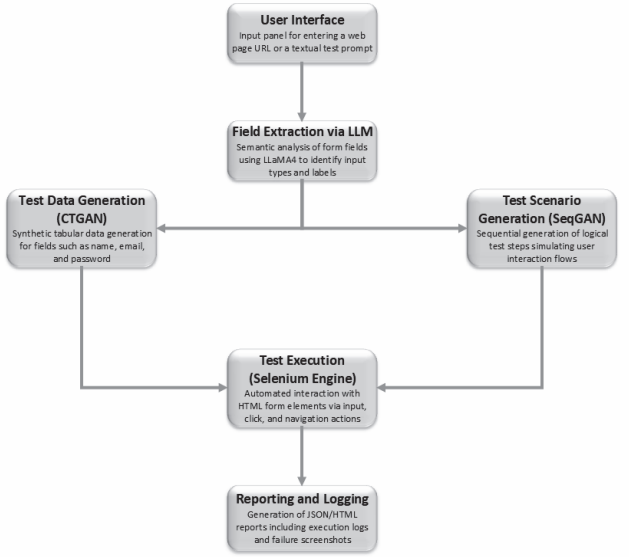

Fig. 1. End-to-End Test Automation Pipeline with GAN and LLM Integration

A. User Interface Layer

The workflow begins with a user-friendly graphical interface built using .NET WinForms. The interface allows testers to:

- Input a target website URL or textual test prompt,
- Select between contextual or non-contextual mode,
- Trigger test data or scenario generation independently,
- Execute automated tests and view the logs.

This design abstracts the underlying complexity, enabling non-technical users to engage with the system effectively [16].

B. Context Extraction Module (LLM-Powered)

Once a URL is provided, the system uses Selenium to load the HTML DOM of the page. A pre-trained LLM (LLaMA4 via Groq API) processes the DOM and extracts semantic descriptions of form fields such as:

```
[
{"field": "username", "type": "text"},
{"field": "password", "type": "password"},
{"field": "email", "type": "email"}
{"field": "country", "type": "text"}
]
```

This contextual information guides both the test data and scenario generation processes, ensuring semantic alignment between generated content and application logic [12], [13].

C. Synthetic Test Data Generation (CTGAN)

CTGAN is responsible for generating tabular data based on the identified input fields. For example, given extracted field names like “First Name”, “Email”, and “Password”, CTGAN creates synthetic records that mimic real-world values while maintaining statistical integrity [3], [9]. A sample output is as follows (Table I):

Table I. Sample Output

Example ID	**User Info**			
Sample	***First Name***	***Email***	***Password***	***Country***
1	Elif	elif@mockmail.com	Abc#1234	Turkiye
2	Alex	alex@testmail.org	9uYT!xZ8	Germany

CTGAN was trained on 500 anonymized records and configured to produce 250 samples per run. The output is saved in CSV format and directly used by the test execution module.

D. Test Scenario Generation (SeqGAN)

In parallel, the system uses SeqGAN to generate structured sequences of test actions. These steps simulate how a user might interact with the system. The model receives initial prompts from the LLM and expands them using reinforcement learning. Example test steps:

1. Enter a valid username
2. Enter a valid password
3. Click the login button
4. Assert redirection to dashboard

SeqGAN’s ability to learn sequential dependencies ensures logical test flows and provides scenario diversity across executions [4], [10].

E. Test Execution Layer (Selenium Engine)

The automation engine, built using Python and Selenium WebDriver, interprets the test scenario and binds the generated data to corresponding HTML elements. It performs the following actions:

- Identifies input fields via name, label, or placeholder attributes,
- Inputs the generated data into the fields,
- Performs button clicks and form submissions,
- Captures screenshots and logs outcomes at each step.

If the system fails to identify a UI element or validate a state, the test case is marked as failed and reported accordingly. Selenium logs and browser snapshots are stored for post-run analysis.

F. Logging and Reporting

After each test execution, the system generates structured reports containing:

- Timestamps and scenario IDs,
- Action-by-action success/failure logs,
- Screenshots at failure points,
- JSON/HTML summaries of test outcomes.

These reports support test reproducibility and help developers trace regressions or UI mismatches efficiently.

G. Modularity and Extensibility

Each module—context extraction, data generation, scenario generation, and test execution—can operate independently or as part of the full pipeline. This modularity allows flexible adoption:

- Developers may skip GAN-based data and plug in their datasets,
- Researchers can test alternative LLMs or GAN variants,
- QA teams can integrate the automation scripts into CI/CD environments.

The system's architecture enables test automation that is not only scalable but also adaptable to a wide range of application contexts, including login flows, registration forms, and survey systems.

IV. Results and Discussion

To assess the effectiveness of the proposed test automation system, a series of experiments were conducted on five publicly available web applications featuring login and registration forms. The evaluation was designed to measure the system's performance in three key aspects:

1. Realism and accuracy of the generated test data (CTGAN)
2. Logical consistency and diversity of test scenarios (SeqGAN)
3. Success rate of automated execution on live websites (Selenium)

All tests were run on a local environment (Intel Core i7, 16 GB RAM, Python 3.12) using Google Chrome controlled via Selenium WebDriver.

A. CTGAN - Based Test Data Evaluation

The test data generated by CTGAN was evaluated against two criteria:

- **Field compatibility**: Does the generated data match the expected format (e.g., email pattern, password length)?
- **Data realism**: Does the data distribution resemble that of real users?

A reference dataset of 500 anonymized user records was used to train the CTGAN model. The CTGAN model was trained for 300 epochs with a batch size of 500, employing a generator embedding dimension of 128, generator and disciminator learning rates both set to 0.0002, two hidden layers of 256 units each for both networks, Adam optimizer betas configured as (0.5-0.9). The model generated 250 synthetic entries per run. Each record was programmatically validated using rule-based format checks (e.g., regular expressions for email format and minimum length for passwords). The validation outcomes were then manually confirmed by a QA engineer to ensure semantic plausibility. Table II summarizes the field-wise format compliance rates based on this dual evaluation process.

Table II. CTGAN Field Compatibility Accuracy

ld Category	**Validation Outcome**
ld Type	***ormat Compliance Rate***
ail	98.4%
sword	91.2%
t Name	96.5%
ntry	94.1%

The CTGAN model was able to replicate realistic field values with a mean format compliance rate of 95.05%. Minor inconsistencies in password generation were attributed to the limited training data variety in that field.

B. SeqGAN-Generated Scenario Evaluation

To validate the scenario generator, a sample of 50 test sequences was produced using LLM prompts followed by SeqGAN-based expansion. Two QA engineers independently reviewed each of the 50 generated test sequences using a structured evaluation rubric. Each scenario was scored on a 5-point Likert scale (1: very poor, 5: excellent) across three evaluation dimensions: step validity, semantic relevance, and variation. The definition of the evaluation dimensions are given below:

- **Step Validity**: Whether each step was actionable and correctly ordered.
- **Semantic Relevance**: Alignment with the intended user action (e.g., login flow).
- **Variation Score**: Degree of step diversification across generated sequences.

To assess the consistency between reviewers, inter-rater reliability was calculated using Cohen's kappa coefficient. For the purposes of agreement measurement, scores were binarized by grouping ratings ≥4 as "acceptable" and <4 as "unacceptable." The resulting Cohen's kappa coefficients were 0.81 for step validity, 0.74 for semantic relevance, and 0.69 for variation, indicating substantial agreement

according to the Landis and Koch scale [20]. These values are reported alongside the average rating scores in Table III.

TABLE III. SCENARIO QUALITY METRICS (MANUAL REVIEW)

Scenario Evaluation	**Scoring Details**
Metric	***Score (Mean / Max = 5)***
Step Validity	4.7
Semantic Relevance	4.6
Variation Score	4.2

The generated scenarios demonstrated high logical consistency and acceptable variation. While most sequences followed standard authentication flows, some included exploratory paths like "Click Forgot Password" or "Toggle password visibility," enhancing test coverage.

To assess the contribution of LLM-guided context extraction to scenario quality, a baseline set of 50 test scenarios was generated without the use of LLM prompts. Independent QA reviewers evaluated both sets using the same rubric. Compared to the baseline, scenarios enhanced with LLM-generated context exhibited a 12% improvement in step validity and a 19% improvement in semantic relevance. This comparison is summarized in Table IV.

TABLE IV. IMPACT OF LLM-GUIDED CONTEXT EXTRACTION ON SCENARIO QUALITY

Scenario Quality Metrics	**LLM Usage**	
Metric	***Without LLM***	***With LLM***
Step Validity	4.2	4.7
Semantic Relevance	3.9	4.6
Variation Score	4.0	4.2

C. Selenium Execution Success Rate

End-to-end automated tests were executed using the generated test data and scenarios on five target web applications. The names of these websites have been anonymized in accordance with data privacy and protection requirements. Execution success was defined as the successful completion of all test steps without runtime exceptions or element identification failures (Table V).

TABLE V. WEB PAGE FORM STRUCTURE SUMMARY

Website Info	**Form Structure**	**Success Metrics**
Website (Anonymized)	***Form Type***	***Success Rate (%)***
Website1	Login	88.0
Website2	Registration	82.0
Website3	Login	78.0
Website4	Registration	91.0
Website5	Login	84.0

The tested web pages consisted of typical form-driven workflows, including login and registration screens.

Across all test cases, the average execution success rate was 84.6%. Failures were primarily due to dynamic element rendering delays and DOM layout changes. To address these challenges, future enhancements will include smarter synchronization mechanisms and AI-based locator healing strategies.

D. Comparison with Manual Testing Baseline

To understand productivity gains, we compared our system against a manual testing baseline where QA engineers created test data and wrote scenarios manually. For a batch of 50 test cases:

- Manual time spent: ~4 hours
- Automated time spent (including generation and execution): ~50 minutes
- Time savings: ~ 79.1%

E. Observations and Limitations

- The modular design of the system allowed independent validation of each component (LLM, CTGAN, SeqGAN, Selenium).
- LLM-based context extraction improved test data relevance and scenario logic
- Execution failures highlighted the need for adaptive selector strategies and responsive wait handling.
- The current implementation focuses on form-based workflows like login and registration, but the architecture supports future extension to multi-step scenarios such as onboarding or dashboards.
- Performance evaluation still involves manual review, limiting full automation. Incorporating quantitative, automated metrics is planned for future improvements.

F. Comparison with Related Work

The proposed system stands out through its integration of LLM-guided context extraction, GAN-based test generation, and browser-level execution within a modular architecture.

Academic tools like CrashScope [14] and TESTAR [15] contribute to automated GUI testing but lack synthetic data generation and semantic understanding. CrashScope focuses on Android crash inputs via static/dynamic analysis, while TESTAR uses random UI actions on desktop apps without context awareness.

In contrast, our system leverages LLMs (e.g., LLaMA4 [7]) to extract and label input fields with contextual meaning (e.g., identifying a password vs. an email field). This enhances the relevance and realism of both the generated test data and scenario steps. Unlike TESTAR and CrashScope, our pipeline covers both the "what to test" and "how to test" questions by synthesizing the data, generating valid action flows, and executing them in-browser with feedback.

Platforms such as TestRigor and Mabl rely on templates and rule-based selectors, which are brittle against DOM changes and do not support generative models like CTGAN [3] or SeqGAN [4]. Our system generates structure-aware test data with improved flexibility and robustness, achieving 95.05% format compliance and 84.6% test success.

LLM-assisted academic tools (e.g., BEAST [13], Mozannar et al. [11]) typically focus on abstract test logic and omit execution or data generation. Our system closes this gap by combining CTGAN-based data synthesis and scenario execution in one pipeline.

Compared to CTGAN's original application [3]—which was mostly evaluated on benchmark datasets for machine learning—the current work demonstrates its novel use in live form-based testing. The generated data is not only statistically coherent but also practically actionable, as confirmed by the Selenium execution success rates.

Additionally, while tools like TextGAN [10] and SeqGAN [4] were previously applied in domains like poetry or dialogue generation, our work adapts SeqGAN for test step synthesis. This repurposing validates its applicability in sequential logic modeling for web workflows, particularly when initialized with context-rich prompts from LLMs.

In summary, our system delivers a fully integrated, executable test pipeline—bridging data and scenario generation—unlike isolated efforts in both academia and commercial tools.

V. Conclusion and Future Work

In this paper, we presented a modular, AI-driven test automation framework that integrates CTGAN, SeqGAN, and LLaMA4-based LLMs for generating and executing context-aware, form-based test cases. The full pipeline operates via a user-friendly interface and automates both data generation and browser-level testing with Selenium.

Experiments on five real-world websites showed strong performance: ~95% data validity, ~4.5/5 scenario quality, and ~85% execution success. Compared to manual efforts, the system reduced testing time by nearly 80%, while improving reproducibility and coverage.

Its modular design enables independent or combined use of components, making it adaptable for QA engineers and DevOps workflows. Moreover, generating synthetic yet semantically aligned data supports privacy-compliant testing environments (e.g., GDPR, KVKK).

However, some limitations remain. The system currently relies on static LLM prompting and rule-based element matching, which may struggle with dynamically generated or JavaScript-heavy UI components. Moreover, test execution failures caused by layout shifts and element timing inconsistencies suggest the need for smarter locator strategies and adaptive synchronization mechanisms.

As part of future work, we plan to address these limitations by exploring the following directions:

- Integration of AI-powered **self-healing selectors** to improve element robustness during runtime.
- Support for **captcha bypassing** or test mocking where necessary.
- Extension to **mobile web or native app testing** using tools like Appium.
- Incorporation of **CI/CD pipeline support** and reporting dashboards.
- Use of active learning to iteratively fine-tune CTGAN/SeqGAN based on real test feedback.
- Future iterations of the framework should aim to automate the performance evaluation process by replacing manual validation with rule-based or learning-based quality metrics.

By bridging the gap between data generation, scenario design, and automated execution, this research lays the groundwork for next-generation AI-powered testing platforms that are scalable, adaptive, and highly productive.

In addition to improving scenario quality and test coverage, the system offers practical benefits such as integration with GUI-based test management, modular extension, and CI/CD pipelines. Future work will focus on expanding these capabilities toward production-grade deployments.

Acknowledgments

This work was supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK TEYDEB) under Project No. 1501-2024-1.

References

[1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in 2007 Future of Software Engineering, IEEE Computer Society, 2007, pp. 85–103.

[2] M. Fewster and D. Graham, Software Test Automation: Effective Use of Test Execution Tools, Addison-Wesley, 1999.

[3] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling Tabular Data Using Conditional GAN," in Advances in Neural Information Processing Systems, vol. 32, 2019.

[4] [L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," in Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017, pp. 2852–2858.

[5] T. Brown et al., "Language Models are Few-Shot Learners," in Advances in Neural Information Processing Systems, vol. 33, 2020.

[6] J. Chowdhery et al., "PaLM: Scaling Language Modeling with Pathways," arXiv preprint arXiv:2204.02311, 2022.

[7] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv preprint arXiv:2302.13971, 2023.

[8] I. Goodfellow et al., "Generative Adversarial Nets," in Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.

[9] A. Choi et al., "Learning Generative Models for Tabular Data with Noisy Mixed Types," in ICML Workshop on Uncertainty and Robustness in Deep Learning, 2020.

[10] X. Zhang, L. Yu, and Y. Yu, "TextGAN: Improving Text Generation via Adversarial Training and Policy Gradient Optimization," in COLING, 2018.

[11] S. Mozannar et al., "Teaching Language Models to Reason About Code," in NeurIPS, 2022.

[12] M. Nasir and Z. Wang, "A Deep Learning Based Approach for Extracting Form Fields from Web Applications," in IEEE Trans. on Software Engineering, vol. 47, no. 6, 2021, pp. 1192–1205.

[13] B. Liang et al., "BEAST: A Semantic-aware Testing Framework for Deep Learning Systems," in ICSE, 2021.

[14] K. Moran et al., "CrashScope: A Practical Tool for Automated Testing of Android Applications," in Proc. of ICSE, 2017.

[15] A. Alégroth and R. Feldt, "TESTAR: Tool Support for Automated Black-box GUI Testing," in ISSTA, 2014.

[16] D. Böhmer, T. Lutz, and B. Brügge, "TestOps: Bringing Continuous Testing to Modern DevOps," in IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2020.

[17] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," arXiv preprint, arXiv:1701.07875, 2017.

[18] A. Aleti, "Software testing of generative AI systems: Challenges and opportunities," in Proc. of the 2023 IEEE/ACM Int. Conf. on Software Engineering: Future of Software Engineering (ICSE-FoSE), Melbourne, VIC, Australia, May 2023, pp. 4–14.

[19] B. Karadavut, G. Kırmızı, A. E. Karamük, B. Öztürk, M. Güven, and A. Guran, "Data augmentation with conditional GANs and SMOTE for balancing imbalanced breast cancer dataset," in Proc. of the 6th Int. Trakya Scientific Research Congress, Edirne, Türkiye, 2025, pp. 229–238.

[20] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," Biometrics, vol. 33, no. 1, pp. 159–174, 1977.