PAPER NAME

G-20.pdf

WORD COUNT	CHARACTER COUNT
4116 Words	24910 Characters
PAGE COUNT	FILE SIZE
8 Pages	343.1KB
SUBMISSION DATE	REPORT DATE
Apr 23, 2025 4:52 PM UTC	Apr 23, 2025 4:53 PM UTC

12% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 6% Internet database
- 4% Publications database
- Crossref database
- Crossref Posted Content database
- 11% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material

Revolutionizing Software Quality Engineering through LLMs based Testing

Kashif Ali, Abdul Moiz, Saboor Shah, Abdul Rauf Kaka, Khurram Zahid
Software Engineering Department,
NEDUET, Karachi, Pakistan.
{ali4502003, moiz4501944, uddin4501956, rauf4500532, zahid4501017}@cloud.neduet.edu.pk

Abstract—**In the era of rapidly evolving software development practices, ensuring software quality has become increasingly complex. Traditional testing approaches often fall short in terms of scalability, speed, and adaptability. This research focuses on the use of Large Language Models into Software Quality Engineering with a focus on their role in enhancing software testing. Using an empirical study based on a web-based POS application, SalesTree, we compare LLM-generated test cases with human created ones across core modules such as user registration, authentication, and product management. The results demonstrate that LLMs can significantly reduce test generation time while uncovering diverse and insightful test cases often missed by humans. Despite current limitations such as redundancy and contextual inaccuracies, the study concludes that LLMs represent a promising shift toward intelligent, adaptive testing frameworks within modern software engineering practices.**

I. Introduction

In the context of software development, ensuring software quality has become both more critical and more complex. As systems scale progressively, traditional testing approaches often struggle to be relevant due to the increment in demand of effort and time. In this context, the integration of artificial intelligence into Software Quality Engineering has emerged as an innovative solution. AI based automated testing is considered to be superior to traditional testing as it saves time[1]. Artificial Intelligence and its subsets, Large Language Models have outshined in the field of software engineering including but not limited to the understanding and generating of code, documentation, and test cases [2].

Software Quality Engineering is a discipline that encompasses the systematic practices and methodologies aimed at ensuring the quality of software throughout its development. Unlike traditional testing, which often focuses on detecting defects after code is written, quality assurance activities begin from the earliest stages of the software development process. It involves not only functional validation but also non-functional quality attributes such as performance, security, usability, and maintainability. IT adopts a proactive approach emphasizing prevention over detection. As software systems become more complex and delivery cycles shrink, SQE plays a critical role in enabling scalable, reliable, and high-performance software solutions.

LLMs have the capability to revolutionize software testing by automating key quality assurance tasks such as test case generation, code review, defect detection, and documentation validation. Unlike rule-based testing tools, LLMs offer contextual reasoning, enabling them to understand requirements written in natural language and translate them into executable test scripts. This represents a significant shift toward intelligent testing frameworks, potentially reducing testing time and increasing productivity [3]. Modern frameworks like ChatUniTest and TestPilot demonstrate how LLMs can be integrated directly into the development process to achieve high coverage while reducing human effort [19].

Software testing is a phase in the software development life cycle that involves evaluating a software application to identify defects and ensure functionality. The motive of testing is to detect bugs early and prevent costly failures in production [4].The process of testing contains unit, integration, system, and acceptance testing..

Effective testing not only helps maintain the reliability and security of a software system. Without rigorous testing, software systems are prone to vulnerabilities that can compromise their usability [5].

While Software Quality Engineering encompasses a wide range of practices aimed at ensuring overall software excellence, this paper focuses specifically on the testing dimension which is a critical pillar of SQE. From automating test case generation and bug detection to enhancing test maintenance and exploratory testing, our objective is to investigate the practical applications, benefits, and limitations of LLMs in modern software testing workflows. By narrowing the scope to testing within the broader SQE framework, we aim to provide a deep, focused analysis of how LLMs can directly contribute to testing of software faster and more efficiently.

II. Related work

Several researchers have explored the integration of AI techniques in software engineering in the recent years. Wang et al. conducted a survey on the application of machine learning and deep learning in the software engineering domain [6]. Watson et al. reviewed the role of deep learning within software development practices [7] . Bajammal et al. examined how computer vision techniques are being utilized to enhance various software engineering procedures, while Zhang et al. focused their survey on testing strategies specifically tailored for machine learning systems.
Zhou et al. [11] further extended this line of research by proposing an evaluation framework for assessing the effectiveness of LLMs across a variety of software testing tasks. Their benchmark includes criteria for test case quality, fault detection, and alignment with human testers—highlighting key strengths and current limitations of existing models [20].
Authors of [8] used 210 Python programs from LeetCode to assess code coverage.
Hou et al. presented a systematic review on the use of LLMs to improve software engineering processes and outcomes [9]. Wang et al. 2024 presented the use of LLMs in differential and mutation testing. They studied 102 research papers focused on the use of LLMs in software testing [1].
While these studies provide useful knowledge into the broader application of LLMs in software engineering, they often pay limited attention to software testing specifically. In contrast, this paper addresses that gap by focusing solely on the use of LLMs for software testing.

III. Methodology

To explore the potential of Large Language Models in the domain of software testing, this study adopts a comparative approach involving both human-authored and LLM-generated test cases. AI can streamline the test suite by eliminating redundant or low-value tests, thereby shortening execution time while maintaining thorough test coverage. For instance, machine learning models can evaluate the performance of specific test cases and recommend which ones to discard or merge [10]. The motive is to tell how far LLMs can enhance traditional software testing practices, particularly within the scope of functional testing. By examining test case generation through both manual and AI-assisted means, this methodology seeks to uncover differences in coverage, clarity, efficiency, and practical applicability. The selected case study for this evaluation is [SalesTree](), a web-based Point of Sale (POS) system developed to facilitate business operations such as inventory control, sales tracking, and user management.

A. Module Selection

For this empirical study, three modules from the SalesTree POS system were selected:

- User registration
- User Login & authentication
- Product management

These modules were chosen due to their high user interaction, business-critical functionality, and presence of diverse logic flows such as validation, conditional responses, and authentication checks.

User Registration Module

This module is responsible for creating new user accounts within the SalesTree platform. It performs strict validation on inputs such as name, email, password, role, and address, ensuring required fields are not missing. The module checks the existence of the specified company and ensures that the user's email is unique. Passwords are validated and securely hashed using bcrypt before database insertion. Upon successful registration, the system sends a welcome email to the user and fetches enriched profile information via SQL joins across related tables.

```
const registerUser = async (body,res)=>{

  if(!body.name || !body.email || !body.company || !body.password){
    throw new Error("required fields are missing!");
  }
  const oldCompany = await db.query(`SELECT * FROM company WHERE name = '${body.company}'`);
  if(oldCompany.rowCount){
    throw new Error("Company Already Exist.");
  }
  const userAvailable = await db.query(`SELECT * FROM users where email = '${body.email}' LIMIT 1`);
  if(userAvailable.rowCount){
    throw new Error("Email Already Exist.");
  }
  const passwordNotValid = validatePassword(body.password);
  if(passwordNotValid){
    throw new Error(passwordNotValid);
  }
  const password = await bcrypt.hash(body.password,10);
  // Insert Company
```

```
 const newCompany = await db.query(`INSERT INTO company
(name) VALUES ('${body.company}') RETURNING ID`);

const newUser = await db.query(`INSERT INTO users (name, email,
company_id,currency, timezone, password)
               VALUES (
                 '${body.name}',
                 '${body.email}',
                  ${newCompany.rows[0].id},
                  ${body.currency ? `'${body.currency}'` : null},
                  ${body.timezone ? `'${body.timezone}'` : null},
                 '${password}'
                  )
               RETURNING ID`);

const user = await db.query(`SELECT users.id ,users.name,
users.email, users.currency, users.timezone,
                  company.name AS company_name FROM users JOIN
company ON users.company_id = company.id
                  WHERE users.id = ${newUser.rows[0].id}`);
const accessToken = jwt.sign({
      user: {
        id: user.rows[0].id,
        name: user.rows[0].name,
        email: user.rows[0].email,
        company:user.rows[0].company_name
      }
   }, process.env.ACCESS_TOKEN_SECRET, { expiresIn: '24h' });

   mailOptions['to'] = user.rows[0].email;
   mailOptions['subject'] = "Welcome To SalesTree"
   mailOptions['html'] = welcomeEmail({name:user.rows[0].name})

   emailClient.sendMail(mailOptions,(error, info) => {
      if (error) {
        console.log(error);
        throw new Error('Failed to send Welcome email.');
      }
   });

   return {
      jwt: accessToken,
      user: user.rows[0]
   }
}
```

User Login & authentication

This module verifies user identity and handles access management for the SalesTree system. It includes input validation, password hashing and verification, account status checking, and token generation. The login process involves multiple decision points, such as checking for missing credentials, invalid passwords, and deactivated user accounts. JWT-based authentication ensures secure session management. Additionally, the module supports password recovery via tokenized reset links. With its layered logic and exception handling mechanisms, this module presents an ideal candidate for detailed white-box testing.

```
const loginUser = async (body, res) => {
   const { email, password } = body;
   if (!email || !password) {
      throw new Error("Required fields are missing!");
   }
 const user = await db.query(`SELECT users.id, users.name,
users.email, users.currency, users.timezone, company.name AS
company_name, users.password , users.profile_image
FROM users JOIN company ON users.company_id = company.id
WHERE users.email = $1 AND users.is_delete = false`, [email]);

   if (!user.rows.length) {
      throw new Error("Invalid email or password.");
   }
   const isPasswordValid = await bycrpt.compare(password,
user.rows[0].password);
   if (!isPasswordValid) {
      throw new Error("Invalid email or password.");
   }
   const employee =  await db.query(`SELECT * from employees
where user_id = ${user.rows[0].id} and is_delete = false`)

   if(employee.rowCount && employee.rows[0].status ===
"inactive"){
      throw new Error("Your account is deactivated. Please contact the
company admin.");
   }
```

Product Management

The Product Management module in SalesTree handles the creation, editing, deletion, and retrieval of product records, along with advanced features such as search, pagination, and quantity validation. Each controller interacts with service-layer logic and enforces business rules like user authentication, input validation, and company-based data scoping. The module supports multiple conditional flows and query-based filtering, making it rich in logic branches and edge-case scenarios.

```
const createService = async (body, createdBy) => {
  if (!body.name || !body.description || !body.category ||
!body.sub_category || !body.barcode || !body.manufacture || !body.price
|| !body.unit || !body.sku || !body.photo) {
    throw nasew Error("required fields are missing!");
  }
  const { name , description , category , sub_category , barcode ,
manufacture , price:selling_price , unit , sku , photo } = body;
  const lowerCaseCategory = category.toLowerCase().trim();
  const lowerCaseSubCategory = sub_category.toLowerCase().trim();
  const lowerCaseManufacture = manufacture.toLowerCase().trim();
  const lowerCaseName = name.toLowerCase().trim();
  const company = await db.query(
    `SELECT ID FROM company where name =
'${createdBy.company}' AND is_delete = false`
  );
  const categoryExist = await db.query(
    `SELECT * FROM category WHERE company_id =
${company.rows[0].id} AND name= '${lowerCaseCategory}' AND
is_delete = false`
  );
  if (!categoryExist.rowCount) {
    throw new Error("Category does not exist.");
  }
    const sub_categoryExist = await db.query(
```

```
  `SELECT * FROM sub_category WHERE company_id =
${company.rows[0].id} AND category = ${categoryExist.rows[0].id}
AND name= '${lowerCaseSubCategory}' AND is_delete = false`
  );
  if (!sub_categoryExist.rowCount) {
    throw new Error("SubCategory does not exist.");
  }
  const manufactureExist = await db.query(
    `SELECT * FROM manufacture WHERE company_id =
${company.rows[0].id} AND name='${lowerCaseManufacture}' AND
is_delete = false`
    );
    if (!manufactureExist.rowCount) {
      throw new Error("Manufacture does not exist.");
    }
  const productExist = await db.query(
  `SELECT * FROM product WHERE company_id =
${company.rows[0].id} AND manufacture_id=
${manufactureExist.rows[0].id} AND sub_category_id =
${sub_categoryExist.rows[0].id} AND category_id
=${categoryExist.rows[0].id} AND name='${lowerCaseName}' AND
is_delete = false`
  );
  if (productExist.rowCount) {
    throw new Error("Product already exist.");
  }
  const product = await db.query(
    `INSERT INTO product
(name,description,company_id,category_id,sub_category_id,manufactu
re_id,sku,selling_price, unit, barcode, image)
    VALUES (
      '${lowerCaseName}',
      '${description}',
      '${company.rows[0].id}',
      '${categoryExist.rows[0].id}',
      '${sub_categoryExist.rows[0].id}',
      '${manufactureExist.rows[0].id}',
      '${sku}',
      '${selling_price}',
      '${unit}',
      '${barcode}',
      '${photo}'

    )`
  );
  return {
    // user: user.rows[0],
    message: "Product successfully created.",
    success: true,
  };
};
```

B. Human Generated Test Cases

Test Case ID	Module	Description	Expected Output
TC-REG-01	User Registration	Register user with all valid fields	User registered, JWT returned, welcome email sent
TC-REG-02	User Registration	Attempt registration with missing name and email	Error: “required fields are missing!”
TC-REG-03	User Registration	Attempt registration with a duplicate email	Error: “Email Already Exist.”
TC-REG-04	User Registration	Password fails validation	Error: "Password too weak" (via validatePassword())
TC-LOGIN-01	User Login & Authentication	Login with correct credentials	JWT token issued, user object returned
TC-LOGIN-02	User Login & Authentication	Login with wrong password	Error: “Invalid email or password.”
TC-LOGIN-03	User Login & Authentication	Login with inactive employee	Error: “Your account is deactivated.”
TC-LOGIN-04	User Login & Authentication	Login with empty email field	Error: “Required fields are missing!”
TC-PROD-01	Product Management	Add product with all valid fields	Product created successfully
TC-PROD-02	Product Manage	Create product without price field	Error: “Product name and price are

	ment		required!”
TC-PROD-03	Product Management	Attempt to delete a product that doesn’t belong to user	Error: "Unauthorized action"
TC-PROD-04	Product Management	Search for product using name query	List of matching products returned

C. LLM Generated Test Cases

To explore the potential of Large Language Models (LLMs) in software testing, this section presents test cases generated using OpenAI’s GPT-4o for the same three modules: User Registration, User Login & Authentication, and Product Management.

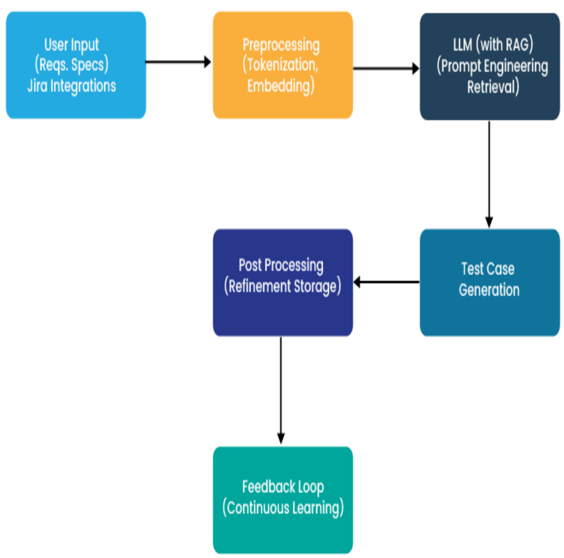

Fig. 1. Flow chart of approach of construction of test cases through LLMs

The approach we employed in generating LLM-based test cases is shown as above in Fig. 1. It begins with user input, which may include requirement specifications, user stories, or integration from platforms such as Jira. We followed the technique of Role Prompting because it gives more elaborated results [7]. This allowed us to influence the model's tone and the type of information it provides .Thus the input undergoes a preprocessing stage, where it is tokenized and transformed into vector embeddings to ensure compatibility with language model inputs. The processed data is then passed to an LLM module, potentially enhanced with Retrieval-Augmented Generation (RAG), which combines prompt engineering with context-aware retrieval from relevant documentation. The model generates preliminary test cases based on the internal logic, expected behavior, and edge scenarios inferred from the input. These outputs move into a post-processing phase, where they are refined for clarity and correctness before being stored. Finally, the system incorporates a feedback loop, enabling continuous learning by capturing tester corrections and execution results

Test case ID	Module	Description	Data	Expected Output
LLM-REG-01	User Registration	Register with a valid email but excessively long name (300 chars)	name = "a...a" (300 chars), valid other fields	Error: "Name exceeds max length" or stored if allowed
LLM-REG-02	User Registration	Register with unsupported characters in password	Password = "<script>alert(1)</script>"	Error: Invalid password format (via validatePassword())
LLM-REG-04	User Registration	Register with unsupported timezone format	timezone = "Mars/Phobos", valid other fields	Error: "Invalid timezone"
LLM-REG-05	User Registration	Register with password missing numeric characters	password = "NoNumbers!"	Error: “Password must contain at least one number”

LLM-REG-06	User Registration	Register with null company_id (simulate company lookup failure)	Valid fields, but company_id = null internally	Error: “Company does not exist.”
LLM-LOGIN-01	User Login & authentication	Attempt login using SQL injection in email	Email = ' OR 1=1--, any password	Error: “Invalid email or password.” (SQL Injection prevented)
LLM-LOGIN-02	User Login & authentication	Login with correct email but expired JWT token (simulated)	Valid credentials but expired token	Error: “Token expired” (if token validation is enforced)
LLM-LOGIN-03	User Login & authentication	Login using correct credentials but employee table missing entry	Valid email/password; no employee record linked	Login succeeds with role = "owner"
LLM-LOGIN-04	User Login & authentication	Login with trailing spaces in email	Email = " user@example.com "	Trimmed input, successful login or validation error
LLM-LOGIN-05	User Login & authentication	Login with emoji in email field	Email = "😊@example.com"	Error: Invalid email format
LLM-PROD-01	Product Management	Attempt to create a product with negative price	productName = "Apple", price = -5	Error: "Invalid price value"
LLM-PROD-02	Product Management	Search with special characters in query	query = "@#$%"	Empty list returned, no crash
LLM-PROD-03	Product Management	Check quantity with non-integer input	quantity = "five"	Error: “Invalid quantity input”
LLM-PROD-04	Product Management	Create product with HTML tags in name field	productName = "Milk"	Stored or sanitized output (based on implementation)
LLM-PROD-05	Product Management	Edit a non-existent product	Valid update body; productId doesn’t exist	Error: "Product not found"
LLM-PROD-06	Product Management	Check quantity with zero as input	quantity = 0	Success or validation error depending on rules
LLM-PROD-07	Product Management	Paginate with negative page number	page = -1, limit = 10	Error or auto-correct to first page

IV. Results

To finalise the results of test cases constructed by LLMs in comparison to manually authored ones, a set of objective metrics was used. These include functional coverage, fault detection potential, redundancy, test case clarity, and time to generate. Each test case set was analyzed for its ability to handle valid and invalid scenarios, identify edge cases, and maintain alignment with business logic. Additionally, the clarity of documentation and the practicality of test case execution were taken into account. This evaluation aims to determine not only the technical soundness of the generated tests but also their real-world usability within a standard QA workflow.

The analysis reveals that both manual and LLM-generated test cases effectively cover the primary functional paths of the selected modules. Manual test cases tended to be more focused on common scenarios and clear requirements derived directly from the specification, while LLM-generated cases were more exploratory and diverse in nature. For instance, the LLM identified boundary inputs, format violations, and injection-like test scenarios not explicitly covered by manual efforts. Moreover, LLMs were able to produce a broad set of test cases in a fraction of the time it took to write them manually. However, some LLM test cases required refinement to eliminate minor redundancy and improve alignment with exact business rules. Overall, the LLM demonstrated strong potential as a complementary tool in white-box testing workflows.

AI systems are capable of adapting and evolving alongside software changes. By continuously learning from new data, they ensure that testing approaches stay aligned with ongoing application updates [12]. They enhance test coverage by creating a broader variety of test cases that may be missed by human testers. Leveraging historical data, AI models can uncover edge cases and uncommon scenarios often overlooked by conventional testing approaches. This results in more comprehensive testing, helping to identify a wider array of potential issues before the software is deployed [13]. These benefits are especially valuable in the early stages of testing when rapid test coverage is essential.

However, LLMs are not without limitations. They can occasionally hallucinate scenarios that do not align with actual system behavior, or generate redundant test cases without understanding project context. Additionally, they lack access to runtime data, code dependencies, or implicit business logic that human testers often consider. Based on the results of this study, LLMs can be further integrated into the software quality engineering protocol through tools and plugins that suggest tests in real-time as developers code. With further refinement, LLMs may evolve from test generators into collaborative quality engineering agents, capable of learning from project feedback and adapting test cases accordingly.

V. Challenges

Despite the promising benefits of Large Language Models software engineering such as automated test case and unit test generation [17,18] their usage introduces many challenges. One major concern is the dependency of LLM-generated test cases on the quality and structure of input data. Poorly written requirements, outdated documentation, or inconsistent specifications can misguide the model, resulting in inaccurate or irrelevant test cases. Making sure the input data is structured and high-quality to achieve the best results [14].

Another key challenge is the cost associated with deploying these models. While companies concerned about data privacy might opt for open-source LLMs to maintain control, doing so comes at a high computational cost. Running large-scale LLMs demands substantial infrastructure and specialized hardware, which can make adoption financially prohibitive for many organizations [15].

Moreover, integrating LLM outputs into existing testing workflows is a complex task. Traditional testing frameworks often require significant adaptation to accommodate AI-generated artifacts. Ensuring smooth interoperability and maintaining consistency across manual, automated, and AI-generated test cases is essential for achieving operational efficiency and sustaining long-term adoption of LLM-based testing approaches [16].

V. Conclusion

This research shows the use of Large Language Models in the software testing procedure. Through the empirical study on the SalesTree POS system, we show that LLMs can generate comprehensive, edge-aware, and time-efficient test cases that complement traditional methods. Their ability to process natural language and infer logic from specifications enables broader test coverage and faster execution cycles, capabilities particularly valuable in Agile and fast-paced development environments. Challenges are always there however despite these hurdles, the findings suggest that LLMs, with fine tuning, have the potential to evolve from being test generators to collaborative agents in quality engineering.

References

[1] Wang, Junjie & Huang, Yuchao & Chen, Chunyang & Liu, Zhe & Wang, Song & Wang, Qing. (2024). Software Testing With Large Language Models: Survey, Landscape, and Vision. IEEE Transactions on Software Engineering. PP. 1-27. 10.1109/TSE.2024.3368208.
[2] Singh, V., & Sood, R. (2021). Cost-Effective Software Testing with AI Techniques. Journal of Software Engineering, 26(7), 101-112.
[3] Khan, N., Aslam, A., & Bukhari, S. (2022). Automating Test Case Generation Using Natural Language Processing. Journal of Software Engineering, 31(2), 78-92.
[4] Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., and Wang, Q. (2024). Software testing with large language models: Survey,

[5] Anwar, Nahid & Kar, Susmita. (2019). Review Paper on Various Software Testing Techniques & Strategies. Global Journal of Computer Science and Technology. 43-49. 10.34257/GJCSTCVOL19IS2PG43.
[6] S. Wang, L. Huang, A. Gao, J. Ge, T. Zhang, H. Feng, I. Satyarth, M. Li, H. Zhang, and V. Ng, “Machine/deep learning for software engineering: A systematic literature review,” IEEE Trans. Software Eng., vol. 49, no. 3, pp. 1188–1231, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2022.3173346
[7] C. Watson, N. Cooper, D. Nader-Palacio, K. Moran, and D. Poshyvanyk, “A systematic literature review on the use of deep learning in software engineering research,” ACM Trans. Softw. Eng. Methodol., vol. 31, no. 2, pp. 32:1–32:58, 2022. [Online]. Available: https://doi.org/10.1145/3485275
[8] Wang, W.; Yang, C.; Wang, Z.; Huang, Y.; Chu, Z.; Song, D.; Zhang, L.; Chen, A.R.; Ma, L. TESTEVAL: Benchmarking Large Language Models for Test Case Generation. arXiv 2024, arXiv:2406.04531.
[9] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. C. Grundy, and H. Wang, “Large language models for software engineering: A systematic literature review,” CoRR, vol. abs/2308.10620, 2023.
[10] Kumar, R., & Gupta, S. (2021). Optimizing Test Suites Using Artificial Intelligence Techniques. Software Engineering Journal, 29(4), 135-144.
[11] Rehan, Shaheer & Al-Bander, Baidaa & Al-Said Ahmad, Amro. (2025). Harnessing Large Language Models for Automated Software Testing: A Leap Towards Scalable Test Case Generation. Electronics. 14. 1463. 10.3390/electronics14071463.
[12] Kumar, P., Singh, A., & Sood, R. (2022). AI and Test Automation: The Future of Continuous Testing in Agile Environments. Agile Development Journal, 18(2), 55-67.
[13] Gupta, S., & Jain, A. (2021). Enhancing Test Coverage through AI-driven Test Case Generation. Journal of Software Testing, 14(3), 112-123.
[14] Patel, K., Sood, R., & Singh, D. (2020). Efficiency Gains in Software Testing through Artificial Intelligence. Journal of Software Engineering and AI, 19(6), 142-156.
[15] Singh, V., & Sood, R. (2021). Cost-Effective Software Testing with AI Techniques. Journal of Software Engineering, 26(7), 101-112.
[16] Hernandez, L., Zhang, T., & Lee, S. (2023). Predictive Analytics in Software Testing: Leveraging Deep Learning for Defect Detection. International Journal of AI Research, 22(1), 98-109
[17] Yuan, Z.; Lou, Y.; Liu, M.; Ding, S.; Wang, K.; Chen, Y.; Peng, X. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. arXiv 2024, arXiv:2305.04207.
[18] Schäfer, M.; Nadi, S.; Eghbali, A.; Tip, F. An empirical evaluation of using large language models for automated unit test generation. IEEE Trans. Softw. Eng. 2023, 50, 85–105.
[19] Sherifi et al. (2024): “Towards AI-Powered Test Generation: An Agent-Oriented Framework for Software Testing.” arXiv.
[20] Zhou et al. (2024): “Evaluating Large Language Models for Software Testing: Benchmarks, Metrics, and Open Challenges.” Computer Standards & Interfaces.

12% Overall Similarity

Top sources found in the following databases:

- 6% Internet database
- 4% Publications database
- Crossref database
- Crossref Posted Content database
- 11% Submitted Works database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	University of Alabama at Birmingham on 2023-08-07	Submitted works	<1%
2	doc.multimag.tndproject.org	Internet	<1%
3	Iftekhar Ahmed, Aldeida Aleti, Haipeng Cai, Alexander Chatzigeorgiou ...	Crossref	<1%
4	Meridian Sp. z o.o. on 2025-04-09	Submitted works	<1%
5	arxiv.org	Internet	<1%
6	Middlesex University on 2024-01-12	Submitted works	<1%
7	University of Warwick on 2024-03-27	Submitted works	<1%
8	bh on 2021-04-18	Submitted works	<1%

9	jettystudy.com Internet	<1%
10	ela.kpi.ua Internet	<1%
11	Liverpool John Moores University on 2024-06-16 Submitted works	<1%
12	University of Central Lancashire on 2024-12-13 Submitted works	<1%
13	University of Pretoria on 2008-11-04 Submitted works	<1%
14	National School of Business Management NSBM, Sri Lanka on 2023-1... Submitted works	<1%
15	Xiangping Chen, Xing Hu, Yuan Huang, He Jiang et al. "Deep learning-b... Crossref	<1%
16	multimag.tndproject.org Internet	<1%
17	Coventry University on 2021-05-14 Submitted works	<1%
18	dev.to Internet	<1%
19	Manipal University on 2024-01-12 Submitted works	<1%
20	ar5iv.labs.arxiv.org Internet	<1%

21	sol.sbc.org.br Internet	<1%
22	Kaplan College on 2025-03-02 Submitted works	<1%
23	Liverpool John Moores University on 2024-08-27 Submitted works	<1%
24	Middle East Technical University on 2024-06-09 Submitted works	<1%
25	Sreenidhi International School on 2023-10-31 Submitted works	<1%
26	University of Keele on 2024-09-13 Submitted works	<1%
27	git.suyu.dev Internet	<1%
28	mikro-orm.io Internet	<1%
29	Ketai Qiu, Niccolò Puccinelli, Matteo Ciniselli, Luca Di Grazia. "From To... Crossref	<1%