Leveraging Pre-Trained Large Language Models (LLMs) for On-Premises Comprehensive Automated Test Case Generation: An Empirical Study

Anonymous submission

Abstract

The rapidly evolving field of Artificial Intelligence (AI)-assisted software testing has predominantly focused on automated test code generation, with limited research exploring the realm of automated test case generation from user stories requirements. This paper presents a comprehensive empirical study on harnessing pre-trained Large Language Models (LLMs) for generating concrete test cases from natural language requirements given in user stories. We investigate the efficacy of various prompting and alignment techniques, including prompt chaining, few-shot instructions, and agency-based approaches, to facilitate secure on-premises deployment. By integrating our learnings with an on-premises model setup, wherein we deploy a RoPE scaled 4-bit quantized LLaMA 3 70B Instruct model, optionally augmented with LoRA adapters trained on QA datasets, we demonstrate that this approach yields more accurate and consistent test cases despite VRAM constraints, thereby maintaining the security benefits of an on-premises deployment.

Introduction

Software testing is a vital process in software development that involves evaluating the quality of a software product. The increasing complexity of modern software systems has led to a growing need for efficient and effective testing strategies. According to a recent survey, software testing accounts for approximately 30-50% of the overall software development cost(Kumar and Mishra 2016a)(Myers, Badgett, and Sandler 2012a)(Ramler and Wolfmaier 2006a). Moreover, the rise of agile methodologies and DevOps practices has further emphasized the importance of automated testing in ensuring timely and high-quality software releases. Meanwhile, pre-trained large language models (LLMs) with coding ability have emerged as a promising solution to assist developers in writing feature code and testing code. The advent of transformer-based architectures has enabled LLMs to learn from vast amounts of code data, exhibiting impressive performance in various programming tasks such as code completion, bug fixing, and code generation(Vaswani et al. 2017)(et al. 2021). However, most techniques introduced in these papers focus on helping developers, such as unit test case generation, system input generation, code repair, etc. The application of LLMs in Quality Assurance (QA) daily activities remains underexplored. It is only natural to apply LLMs in the domain of QA to automatically alleviate some of this manual effort. However, despite the significant capabilities of cloud-hosted AI models, data privacy concerns arise when dealing with confidential data, such as that found in user stories and upcoming product features. In such cases, hosting an on-premises LLM service is a necessity to meet data security requirements. Nevertheless, on-premises deployments can be more VRAM constrained, requiring efficient training and deployment of models, and the choice of pre-trained LLMs is limited to open-source models.

Moreover, one of the most critical daily activities for QA members is to write test cases based on feature requirements. Since both feature requirements and test cases are primarily written in natural language, leveraging LLMs to perform this task is a promising direction. However, simple zero-shot prompting may not generate satisfactory test cases meeting QA standards, exposing several issues. Namely,

- **Incompleteness**: Generated test cases do not cover all the requirements and miss important scenarios, leading to inadequate testing.
- **Inconsistency**: Generated test cases lack consistency in quantity, style, and coverage, making it challenging to ensure comprehensive testing.
- **Naivety**: Generated test cases are naive or incorrect due to misunderstandings of domain-knowledge-intensive requirements, potentially resulting in false positives or negatives.
- **Security risks**: Using online LLM services may put corporate confidential data at risk, compromising the integrity of the testing process.

In this paper, we will conduct empirical studies to determine the most effective approach for test case generation in the QA domain using pre-trained LLMs, evaluating various open-source models and techniques such as prompting, alignment, and architecture. Based on those learnings, our research will culminate in an on-premises LLM solution built on top of the LLaMA 70B family of models (LLaMA 3(Dubey et al. 2024) and LLaMA 2(Touvron et al. 2023)), tailored for test case generation in the QA domain.

Background

A typical software testing cycle has the following tasks.

- Product managers define the feature requirement and document them in user stories.

- QA members review the requirement and create test cases.
- Engineers follow the requirement to develop software features/products.
- QA members receive the developed features/products, execute the test cases and report test results.

Test case is the term we use in this work to refer to the documentation of test procedures that QA members follow to perform a test. QA members need to review and understand the requirement, then carefully design test cases to cover the requirement as much as possible. Due to the recent rise of AI and LLM, it's promising to utilize AI and LLM to help reduce the manual effort.

Public online AI services are powerful tools that can help with this task. However, the data will be submitted to the service provider and subject to training and other purposes. In such cases, hosting an on-premises LLM service is a must due to data security requirement.

This article presents an in-house LLM solution for test case generation. We evaluate current open source models, conduct experiment for both prompting and finetuning and analyze the result thoroughly.

There are not many automated ways to create test cases. Test case creation and documentation is historically a manual process. Since there are open source models available, it is natural to try them for this task. Here we categorize the task into two types and evaluate vanilla model (Section 3.2) performance.

Basics Test Case Creation

Different projects may have different style of user story requirement and test cases. The way to write test cases can also vary from person to person. Here we start from the most natural and common test case creation process. We use user story requirement as input to the LLM and prompt the LLM to generate desired test cases.

Example of user story requirement:

As a user who purchased license, I want to use the premium feature without limitation.
Acceptance Criteria:
1. User should have access to premium feature after purchasing license.
2. User should be blocked to access premium feature when license is expired or deactivated.

QA member will determine all the scenarios to test and document each scenario with steps as a test case. A typical test case consists of title, precondition, test steps and expected result. See example below.

Title:
Verify that a user who purchased license should have access to premium feature.
Precondition:
App is installed.
Test Steps:
1. Launch the app.
2. Try to access premium feature.
3. Go to settings, license, input license key, activate license.
4. Try to access premium feature.
Expected Result:
At Step 1: App is launched successfully.
At Step 2: Pop up showing "Please activate license"
At Step 3: License is activated successfully.
At Step 4: Should be allowed to user premium feature without limitation.
We use simple one-liner prompt to instruct the vanilla model. We found that although it can generate some meaningful test cases, a lot of rework needs to done due to the inconsistency, missing coverage and incorrect styling.

Advanced Test Case Creation

Above is the most basic requirement for test case creation. However, this is still not sufficient in real life testing. To further improve the test cases, the following enhancement are needed.

- Project background information. The user story is incremental and describes the new requirement on top of existing product. For testing, QA members usually build new verification steps on top of existing product behaviors. This will make the test cases more concrete and actionable.
- Edge cases. In real life, testing the basic scenarios from user stories is usually insufficient. We need to consider edge cases to make the produce more robust. For example, to test app related feature on the phone, we need to consider how the feature behaviors when the device is rotated, rebooted, in low-battery mode etc. Usually we prepare a checklist of scenarios as refence when creating additional edge test cases.

Unless specifically handled, Vanilla model will perform poorly for advanced test case generation. Since Vanilla model does not meet the need, in this article, we improve both basic test case creation and advanced test case creation utilizing a combination of techniques.

Experiments

Benchmark

Evaluation for basic test case creation.
It is crucial to know how to evaluate the result so we can iteratively improve it. Due to the characteristic of this use case, not many straight-forward evaluation methods are available. Here we use human evaluation.

For test case generation, basic test coverage is the most important metrics. Basic test coverage means the generated test cases should cover all the requirement described in the input. We calculate basic test coverage by the following two steps.

Step 1, we sequentially evaluate each generated test case and filter out low quality ones.

- Inaccuracy: This evaluates whether the test case correctly verifies one scenario. If inaccurate, this test case does not count towards total coverage and should be ignored.

- Irrelevance: This evaluates whether the test case is related to the requirement or not. If irrelevant, this test case does not count towards total coverage and should be ignored.
- Redundancy: This evaluates whether the test case verifies same scenarios covered by other test cases. If redundant, this test case does not count towards total coverage and should be ignored.

Step 2, we check the remaining good quality test cases and compare them with manual generated test cases. Here is one example. In this case, the coverage will be 8 / 10 = 80%.

	Total generated	Inaccurate	Irrelevant	Redundant	Basic Test Coverage
Human	10	0	0	0	10
AI	12	1	2	1	8

Table 1: Calculate Basic Test Coverage

Evaluation for advanced test case creation.
Since project background information and edge case testing are additional information that does not directly come from original user story, we can manual examine the output and quickly determine if the two enhancement exists.

Experiment Setup

Language Models

There are many open source LLM models available with different size and variant. We choose the best one that fits our needs. We use llama models(Touvron et al. 2023) with varying scales to do the experiment.

- Llama-2-70b (7b, 13b and 70b)

This is the base model for text completion. Since our use case requires well-designed instructions, base model may not be best option. This also applies to fine-tuning. Training the base model to follow test case generation instructions requires large amount of data. It is more efficient to reuse of instruction-following ability of chat model instead of training base model from scratch.

- Llama-2-chat (7b, 13b and 70b)

We evaluated models of three sizes. Smaller models require much less computation resource but may not necessarily meet our use case requirement.

- Meta-Llama-3-70B-Instruct

This is the latest llama model with enhanced performance. We evaluate this model for our use case as well.

Prompts

Test case creation is not a simple task that can be described in a simple instruction. To improve the performance of test case creation, we explore the following prompting techniques.

- Zero-Shot

In zero-shot prompting, we just use a simple prompt without any demonstrations to guide the model to generate test cases. We specify the test case template so the output test cases formatting looks consistent.

- Few-Shot(Brown et al. 2020)

This is to provide one, two or more examples in the prompt so that the model can follow. In our use case, output can be very long if many test cases are generated. In such case, inserting even one example will make the prompt much longer and more complex. Adding more than one example may exceed the conext window limit of our set up and degrade the output quality. Therefore, we only experiment one shot.

- Prompt-Chaining(Li et al. 2023)(Wu, Terry, and Cai 2021)

Test case creation is a complex task with one prompt. It makes sense to break it down into smaller tasks. Here we define two subtasks(Wei et al. 2022).

- Step 1. Generate one-liner test scenarios which summurize the test case.
- Step 2. Generate test cases using test scenarios as guidance.

In the first step we prompt the model to generate test scenarios instead of complete test cases with steps. Here we define test scenario as a one-liner statement covering one funcitonality of the software under test. Then in the second prompt, we use the intermediate test scenarios as guidance to prompt the model to generate complete test cases.

- Prompt-Chaining + Few-Shot

In prompt chaining, the first prompt generates a list of one-liners. This provides more room for improvement since the output are shorter compared to a list of full test cases. Inserting examples becomes more doable. Here we use one-shot, two-shot and three-shot in the first prompt, keeping second prompt the same, and evaluate prompt chaining result.

- Prompt-Chaining + Few-Shot sequentially

During experiment, we find that sometimes the output of second prompt is not able to follow the test scenarios guidance accurately. The chance is high when there are a lot of intermediate test scenarios. In this case, we design a variant strategy for second prompt. When we have a list of intermediate one-liner test scenarios, we prompt the model to generate one test case following one test scenario at a time.

Fine-tuning setup

QA teams usually maintain test cases for existing projects. If the model can learn from existing data, output test cases should be improved greatly.

We collect human-generated test cases for existing projects and convert them to training samples. Each sample contains one user story as input and matching test cases as output.

We use Llama-2-70B-chat as base model for fine-tuning. Due to GPU VRAM limit, we use QLoRA(Dettmers et al. 2023), in which we quantize the base model in 4bit and only train the adapter. Our goal is to maximize the training quality while making sure training run always fits in the memory.

Experiment Result

Basic test case creation.

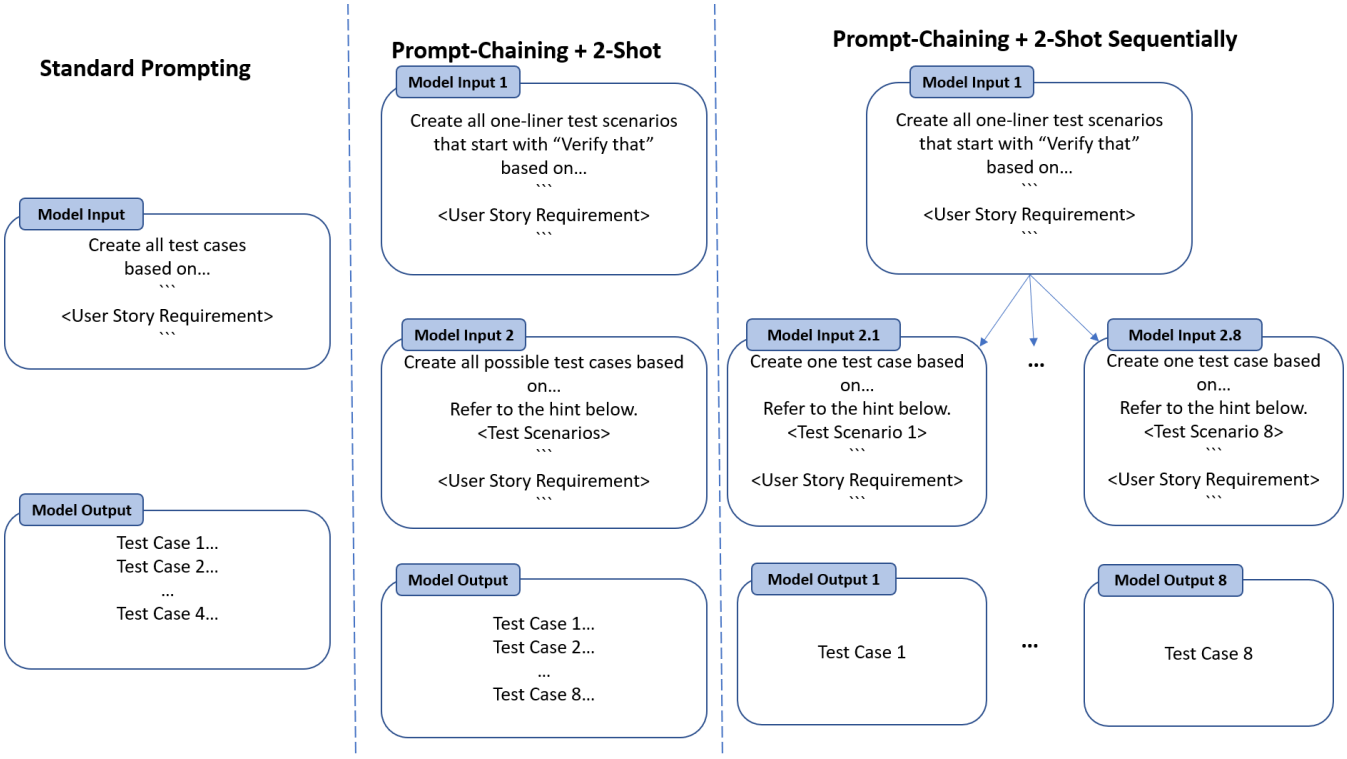

Figure 1: Prompt-Chaining + Few-Shot

Model Size	Coverage
Llama-2-7b-chat	46.7%
Llama-2-13b-chat	49.8%
Llama-2-70b-chat	80.6%
Llama-3-70B-Instruct	79.6%

Table 2: Performance of different model sizes

In conclusion, we can see from Table II that 70B model can generate test cases with higher coverage than 7B and 13B models. Llama3 offers same performance for this specific task.

Technique	Coverage
Zero-Shot	67.7%
One-Shot	73.9%
Prompt-Chaining + zero-shot	76.9%
Prompt-Chaining + one-shot	85.2%
Prompt-Chaining + two-shot	89.7%
Prompt-Chaining + three-shot	80.6%
Prompt-Chaining + two-shot sequentially	90.3%

Table 3: Performance of different prompting techniques

According to Table III, comparing zero-shot with one-shot, we can see that one-shot will degrade the performance. This is probably because the example is too complex for the model to learn.

Prompt chaining with zero-shot achieves same level of performance as zero-shot, but makes injecting example feasible. We evaluate prompt chaining with one-shot, two-shot and three-shot. Among them, two-shot performs the best. With three-shot or more, the prompt may become long and complex again, which is hard for the model to recognize a pattern. Therefore, adding more examples does not increase, or even decrease, the output quality.

Comparing prompt chaining with two-shot sequentially and non-sequentially, they achieve similar performance when not many intermidiate test scenarios are generated. However, when a lot of intermidiate test scenarios are generated, the model is not able to follow the guidance effectively in a single prompt. In this case, sequentially prompting the model has higher chance to align the generated test cases with the test scenarios.

Advanced test case creation.

Adding project background information Background information is stored in different format and at different places. It will be hard to automatically inject relevant project-specific background information to the prompt. Implementing retrieval-augmented generation with background information as references is one option. But consolidating background information from varous formating is a challenge. We observe that if the project is not in very early stage, there should be existing test cases created. These test cases can act as intial source of background infromation and be used training dataset. Fine-tuning the model with existing test cases is a more practical and effective way improve the

test cases generation. We observe that after finetuning, generate test cases learns project background information and includes them in the test cases. The preconditions and test steps contain steps specific to the project.

Adding edge test cases There are multiple ways to add additional edge test cases. Usually, it's more efficient to specify the edge cases to add in the prompt. Here we alter the first prompt in prompt chaining to add desired edge scenarios.

Limitation and mitigation

Output test cases quality has dependencies on input user stories.

Input Accuracy The user story has to be written in a professional and accurate manner. If some part of the user story is not written clearly, LLM may get confused and generate vague test cases.

Input Detailedness Since user story is the sole information that LLM can rely on to create test cases, the more self-contained the user story, the more detailed and accurate the generated test cases. Any missing information could lead to test cases with decreased coverage.

To mitigate these issues, it is best to enforce accuracy and detailedness standard when creating user stories.

Conclusion and Discussion

In this paper, we defined the scope for automated test case creation and introduced novel techniques and experiments to address this use case effectively. For basic test case creation, we employed a combination of prompt chaining and two-shot prompting to achieve optimal coverage, ensuring that the generated test cases were comprehensive and aligned with the software requirements. For advanced test case creation, we utilized supervised fine-tuning (SFT) to incorporate specific project information, thereby enhancing the relevance and accuracy of the generated test cases. We further applied prompt chaining to inject additional checklist-based test cases, ensuring that edge cases and project-specific nuances were adequately covered.

Additionally, we discussed the practical challenges encountered during the implementation, including the limitations inherent in using large language models (LLMs) and the strategies to mitigate these challenges. These included managing the complexity of user stories, ensuring the accuracy and consistency of the generated test cases, and addressing the computational constraints of on-premises deployment.

The results of our experiments demonstrate that the integration of LLMs into the test case generation process not only reduces manual effort but also enhances the efficiency and scalability of the QA process. Our approach significantly decreases the time required for test case creation, particularly for complex user stories, thereby contributing to faster software delivery cycles and improved software quality.

Future work could explore further optimization of the models used, potentially integrating more sophisticated fine-tuning techniques and expanding the scope to include additional types of test cases. Moreover, as LLM technology continues to evolve, there is potential for even greater efficiency and coverage in automated test case generation, further reducing the reliance on manual processes and enhancing overall software reliability.

Practical Implications and Economic Impact of Automated Test Case Generation

The creation and documentation of high-coverage test cases are crucial steps in the QA process. Leveraging large language models (LLMs) can significantly reduce the manual effort required for this task. According to a survey conducted by interal QA team, generating around 15 test cases for a user story typically takes 1-3 hours for a QA member to complete, depending on the complexity of the user story and the member's familiarity with the project. In contrast, our on-premises LLM, fine-tuned specifically for the project, can produce corresponding test cases in under 5 minutes.

For more complex user stories requiring 30 or more test cases, the manual process may extend over a day, whereas our LLM accomplishes the task in under 40 minutes. This stark reduction in time not only enhances productivity but also significantly lowers the costs associated with manual test case generation. Furthermore, with additional computational resources, such as batching across a multi-GPU setup or reducing quantization, the generation time can be further decreased, making LLMs even more efficient and effective in automating test case creation.

The economic implications of these advancements are substantial. Defect leakage into production, often due to inadequate test coverage, can be costly. According to the National Institute of Standards and Technology (NIST), the cost to fix defects post-release can be up to 30 times higher than during the early development stages (National Institute of Standards and Technology (NIST) 2002). Poorly designed test cases lead to higher maintenance costs, as frequent updates are required to ensure adequate coverage, diverting resources from critical tasks such as feature development. This is especially problematic in agile environments where rapid iteration is key (Kumar and Mishra 2016b).

Moreover, suboptimal test cases can reduce software quality, leading to customer dissatisfaction and brand damage, particularly in industries where reliability is paramount, such as finance or healthcare (Myers, Badgett, and Sandler 2012b). Delayed time-to-market is another consequence, as extended testing phases caused by inefficient test case creation can push back release schedules, resulting in lost market opportunities (Ramler and Wolfmaier 2006b). Lastly, the opportunity cost of redirecting resources to address issues stemming from poor test cases can stifle innovation and reduce competitiveness (Gaffney and Durek 1989).

In summary, automating test case generation using LLMs not only mitigates these risks but also enhances efficiency, allowing QA teams to focus on higher-level, value-adding tasks. This results in faster time-to-market, lower costs, and ultimately, higher software quality.

User Story Input (Plain Text)

Title: User Registration
As a new user
I want to register for an account
So that I can access members-only content on the website.

Acceptance Criteria:

1. The registration form should include fields for the user's name, email, and password.
2. The email field should validate for proper email format.
3. The password field should require at least 8 characters, including one uppercase letter, one number, and one special character.
4. The system should send a confirmation email with a link to verify the user's email address.
5. The user should be redirected to the login page after successful registration.

Notes:

- The confirmation email should be sent using the existing email service.
- The registration form should be accessible on both desktop and mobile devices.

Prompt Chaining + Two-shot Output

Test Case 1: Valid Registration Information
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password.

Test Steps:

1. Open the registration form on the website.
2. Enter the user's name, email, and password in the appropriate fields.
3. Click the "Submit" button.

Expected Results:

1. The system should send a confirmation email to the user's registered email address.
2. The user should receive a message indicating that they have successfully registered.
3. The user should be able to log in with their registered email and password.

Test Case 2: Invalid Email Address
Preconditions:

1. The user has not registered previously.
2. The user has an invalid email address (e.g., `username@example.com`).
3. The user knows their desired password.

Test Steps:

1. Open the registration form on the website.
2. Enter the user's name, invalid email address, and password in the appropriate fields.
3. Click the "Submit" button.

Expected Results:

1. The system should display an error message indicating that the email address is invalid.
2. The user should not receive a confirmation email.
3. The user should not be able to log in with their registered email and password.

Test Case 3: Insufficient Password Length
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password, but it is less than 8 characters long.

Test Steps:

1. Open the registration form on the website.
2. Enter the user's name, email, and short password in the appropriate fields.
3. Click the "Submit" button.

Expected Results:

1. The system should display an error message indicating that the password is too short.
2. The user should not receive a confirmation email.
3. The user should not be able to log in with their registered email and password.

Test Case 4: Confirmation Email Not Sent
Preconditions:

1. The user has registered successfully.
2. The user has not received a confirmation email.

Test Steps:

1. Check the user's email inbox for the confirmation email.
2. If the email is not found, contact the system administrator to investigate.

Expected Results:

1. The system should send a confirmation email to the user's registered email address.
2. The user should receive a message indicating that they have successfully registered.
3. The user should be able to log in with their registered email and password.

Test Case 5: Registration Form Accessibility
Preconditions:

1. The user has a desktop or mobile device with internet access.
2. The user has not registered previously.

Test Steps:

1. Open the registration form on the website using a desktop or mobile device.
2. Attempt to enter the user's name, email, and password in the appropriate fields.
3. Click the "Submit" button.

Expected Results:

1. The registration form should be accessible and usable on both desktop and mobile devices.
2. The user should be able to register successfully using either device.

Test Case 6: Existing Email Service Integration
Preconditions:

1. The user has registered successfully.
2. The system uses an existing email service (e.g., SendGrid) to send confirmation emails.

Test Steps:

1. Check the system's configuration to ensure that it is using the existing email service.
2. Send a test email using the existing email service to verify its functionality.

Expected Results:

1. The system should use the existing email service to send confirmation emails to newly registered users.
2. The test email should be delivered successfully using the existing email service.

Prompt Chaining + One-shot Output

Test Case 1: Valid Registration
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password.

Test Steps:

1. Open the registration form on the website.
2. Enter the user's name, email, and password in the appropriate fields.
3. Click the "Submit" button.
4. Verify that the system sends a confirmation email to the user's email address.
5. Follow the link in the confirmation email to verify the user's email address.
6. Log in with the newly created account.

Expected Result:

1. The user receives a confirmation email with a link to verify their email address.
2. The user's email address is verified successfully.
3. The user is able to log in with their newly created account.

Test Case 2: Invalid Email Address
Preconditions:

1. The user has not registered previously.
2. The user has an invalid email address (e.g., `username@example.com`).
3. The user knows their desired password.

Test Steps:

1. Open the registration form on the website.
2. Enter the user's name, invalid email address, and password in the appropriate fields.
3. Click the "Submit" button.
4. Verify that the system does not send a confirmation email.
5. Verify that the user is not able to log in with their newly created account.

Expected Result:

1. The system does not send a confirmation email.
2. The user is not able to log in with their newly created account.

Test Case 3: Password Length
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password, which is less than 8 characters long.

Test Steps:

1. Open the registration form on the website.
2. Enter the user's name, email, and short password in the appropriate fields.
3. Click the "Submit" button.
4. Verify that the system does not allow the user to register.
5. Verify that the user is not able to log in with their newly created account.

Expected Result:

1. The system does not allow the user to register.
2. The user is not able to log in with their newly created account.

Test Case 4: Confirmation Email Link
Preconditions:

1. The user has registered successfully.
2. The user has not yet verified their email address.

Test Steps:

1. Open the confirmation email sent by the system.
2. Click the link to verify the user's email address.
3. Verify that the system marks the user's email address as verified.
4. Log in with the newly created account.

Expected Result:

1. The system marks the user's email address as verified.
2. The user is able to log in with their newly created account.

Test Case 5: Mobile Device
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password.

Test Steps:

1. Access the registration form on a mobile device.
2. Enter the user's name, email, and password in the appropriate fields.
3. Click the "Submit" button.
4. Verify that the system sends a confirmation email to the user's email address.
5. Follow the link in the confirmation email to verify the user's email address.
6. Log in with the newly created account.

Expected Result:

1. The user receives a confirmation email with a link to verify their email address.
2. The user's email address is verified successfully.
3. The user is able to log in with their newly created account.

Test Case 6: Desktop Device
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password.

Test Steps:

1. Access the registration form on a desktop device.
2. Enter the user's name, email, and password in the appropriate fields.
3. Click the "Submit" button.
4. Verify that the system sends a confirmation email to the user's email address.
5. Follow the link in the confirmation email to verify the user's email address.
6. Log in with the newly created account.

Expected Result:

1. The user receives a confirmation email with a link to verify their email address.
2. The user's email address is verified successfully.
3. The user is able to log in with their newly created account.

Test Case 7: Registration Form Accessibility
Preconditions:

1. The user has not registered previously.
2. The user has a valid email address.
3. The user knows their desired password.

Test Steps:

1. Access the registration form on a desktop device with a screen reader enabled.
2. Enter the user's name, email, and password in the appropriate fields.
3. Click the "Submit" button.
4. Verify that the system sends a confirmation email to the user's email address.
5. Follow the link in the confirmation email to verify the user's email address.
6. Log in with the newly created account.

Expected Result:

1. The user is able to complete the registration form using a screen reader.
2. The system sends a confirmation email to the user's email address.
3. The user's email address is verified successfully.
4. The user is able to log in with their newly created account.

Prompt Chaining + Zero-shot Output

Test Case 1: Valid Registration
Preconditions:

- The user has not registered before.
- The user has a valid email address.
- The user knows their password.

Test Steps:

1. Access the registration form on the website.
2. Enter a valid name, email address, and password in the appropriate fields.
3. Click the "Submit" button.

Expected Results:

1. The system should send a confirmation email to the user's email address.
2. The user should receive a message indicating that they have been registered successfully.
3. The user should be able to log in to their account using their registered email and password.

Test Case 2: Incorrect Email Format
Preconditions:

- The user has not registered before.
- The user has an invalid email address (e.g., `username@localhost`).
- The user knows their password.

Test Steps:

1. Access the registration form on the website.
2. Enter a valid name and password in the appropriate fields.
3. Enter an invalid email address in the email field.
4. Click the "Submit" button.

Expected Results:

1. The system should display an error message indicating that the email address is in an invalid format.
2. The user should not be registered.

Test Case 3: Short Password
Preconditions:

- The user has not registered before.
- The user has a valid email address.

- The user knows their password but it is less than 8 characters long.

Test Steps:

1. Access the registration form on the website.
2. Enter a valid name and email address in the appropriate fields.
3. Enter a short password (less than 8 characters) in the password field.
4. Click the "Submit" button.

Expected Results:

1. The system should display an error message indicating that the password is too short.
2. The user should not be registered.

Test Case 4: Confirmation Email Not Sent
Preconditions:

- The user has registered successfully.
- The user has not received a confirmation email.

Test Steps:

1. Check the user's email inbox for a confirmation email from the system.
2. If no confirmation email is found, contact the system administrator to verify that the email was not sent.

Expected Results:

1. The system should send a confirmation email to the user's email address.
2. The user should receive a confirmation email within a reasonable timeframe (e.g., 5 minutes).

Test Case 5: Registration Form Accessibility
Preconditions:

- The user has a valid email address.
- The user knows their password.

Test Steps:

1. Access the registration form on a desktop device.
2. Enter a valid name, email address, and password in the appropriate fields.
3. Click the "Submit" button.
4. Repeat steps 1-3 on a mobile device.

Expected Results:

1. The registration form should be accessible and functional on both desktop and mobile devices.
2. The user should be able to register successfully on both desktop and mobile devices.

Test Case 6: Database Account Creation
Preconditions:

- The user has registered successfully.

Test Steps:

1. Verify that the user's account has been created successfully in the database.
2. Check the user's account details (e.g., email, password, username) in the database.

Expected Results:

1. The user's account should be created successfully in the database.
2. The user's account details should match the information entered during registration.

Test Case 7: Logging In With Registered Email and Password
Preconditions:

- The user has registered successfully.
- The user knows their registered email and password.

Test Steps:

1. Access the login page on the website.
2. Enter the registered email and password in the appropriate fields.
3. Click the "Login" button.

Expected Results:

1. The system should authenticate the user successfully.
2. The user should be able to access members-only content on the website.

Test Case 8: Verification Email Link
Preconditions:

- The user has registered successfully.
- The user has not verified their email address.

Test Steps:

1. Locate the verification email sent to the user's email address.
2. Click the verification link in the email.

Expected Results:

1. The system should verify the user's email address successfully.
2. The user should receive a message indicating that their email address has been verified.

Test Case 9: Email Address Verification
Preconditions:

- The user has registered successfully.
- The user has not verified their email address.

Test Steps:

1. Access the user profile page on the website.
2. Click the "Verify Email" button.

Expected Results:

1. The system should send a verification email to the user's email address.
2. The user should receive a message indicating that a verification email has been sent.
3. The user should be able to click the verification link in the email to verify their email address.

Test Case 10: Login After Email Verification
Preconditions:

- The user has registered successfully.

- The user has verified their email address.

Test Steps:

1. Access the login page on the website.
2. Enter the registered email and password in the appropriate fields.
3. Click the "Login" button.

Expected Results:

1. The system should authenticate the user successfully.
2. The user should be able to access members-only content on the website.

References

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; et al. 2020. Language Models are Few-Shot Learners.

Dettmers, T.; Pagnoni, A.; Holtzman, A.; and Zettlemoyer, L. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

et al., C. 2021. A Survey on Neural Code Generation.

Gaffney, J. E.; and Durek, T. A. 1989. *Software Cost Estimation Techniques*. Academic Press.

Kumar, D.; and Mishra, K. 2016a. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia Computer Science*, 79: 8–15. Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

Kumar, D.; and Mishra, K. 2016b. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia Computer Science*, 79: 8–15.

Li, Z.; Peng, B.; He, P.; Galley, M.; Gao, J.; and Yan, X. 2023. Guiding Large Language Models via Directional Stimulus Prompting. arXiv:2302.11520.

Myers, G. J.; Badgett, T.; and Sandler, C. 2012a. The Art of Software Testing.

Myers, G. J.; Badgett, T.; and Sandler, C. 2012b. *The Art of Software Testing*. Wiley, 3rd edition.

National Institute of Standards and Technology (NIST). 2002. The Economic Impacts of Inadequate Infrastructure for Software Testing. Accessed: 2024-08-26.

Ramler, R.; and Wolfmaier, K. 2006a. Economic perspectives in test automation: balancing automated and manual testing with opportunity cost.

Ramler, R.; and Wolfmaier, K. 2006b. Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost. In *Proceedings of the 2006 ACM International Workshop on Automation of Software Test*, 85–91. ACM.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Vaswani, A.; Shazeer, N.; Parmar, N.; Jones, L.; Gomez, A.; Łukasz Kaiser; and Polosukhin, I. 2017. Attention Is All You Need.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Chi, E. H.; Le, Q.; and Zhou, D. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models.

Wu, T.; Terry, M.; and Cai, C. J. 2021. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts.