UMEÅ UNIVERSITY

AI driven Test Case Generation

An In-Depth study on the Utilization of Large Language Models for Test Case Generation

Nicole Johnsson

Nicole Johnsson
Spring 2023
Degree Project in Interaction Technology and Design, 30 credits
Supervisor: Gustav Grund Pihlgren
External Supervisor: Bharath Karthikeyan
Examiner: Ola Ringdahl
Master of Science Programme in Interaction Technology and Design, 300 credits

Abstract

This study investigates the utilization of Large Language Models for Test Case Generation. The study uses the Large Language model and Embedding model provided by Llama, specifically Llama2 of size 7B, to generate test cases given a defined input. The study involves an implementation that uses customization techniques called Retrieval Augmented Generation (RAG) and Prompt Engineering. RAG is a method that in this study, stores organisation information locally, which is used to create test cases. This stored data is used as complementary data apart from the pre-trained data that the large language model has already trained on. By using this method, the implementation can gather specific organisation data and therefore have a greater understanding of the required domains. The objective of the study is to investigate how AI-driven test case generation impacts the overall software quality and development efficiency. This is evaluated by comparing the output of the AI-based system, to manually created test cases, as this is the company standard at the time of the study. The AI-driven test cases are analyzed mainly in the form of coverage and time, meaning that we compare to which degree the AI system can generate test cases compared to the manually created test case. Likewise, time is taken into consideration to understand how the development efficiency is affected.

The results reveal that by using Retrieval Augmented Generation in combination with Prompt Engineering, the system is able to identify test cases to a certain degree. The results show that 66.67% of a specific project was identified using the AI, however, minor noise could appear and results might differ depending on the project's complexity. Overall the results revealed how the system can positively impact the development efficiency and could also be argued to have a positive effect on the software quality. However, it is important to understand that the implementation as its current stage, is not sufficient enough to be used independently, but should rather be used as a tool to more efficiently create test cases.

Acknowledgements

I would like to thank the people around me, who believed in my abilities, in moments when I had a drought. To my parents that kept encouraging me, and reminding me that no step, easy or hard, is impossible to make. That a star is only one step from the moon. To my brother that every day inspired me to be better and have a hunger for learning. And lastly, to my family around the world, who celebrated all of my successes like their own.

I would also like to express my gratitude to my supervisor from Umeå University, Gustav Grund Pihlgren, who continuously provided guidance, calmness, and advice that was needed. To my external supervisor, Bharath Karthikeyan, who with a direction and a smile helped me through the entire process. To my teams, Onboarding Experience and Activation Experience, and to the organization PayPal as a whole. Thank you for listening, cheering, and lifting me up.

Last but definitely not least, I want to thank my friends. That even on the rainiest days brought me sunlight.

Contents

1 Introduction

Artificial Intelligence, or AI is a type of technology that mimics intelligent behavior through the use of computer models. The term was officially introduced in 1955, by John McCarthy but is today fully incorporated into society through numerous technologies such as the Voice Assistant Siri or Fraud Detection in banks [34].

One of the most acknowledged breakthroughs in the field of AI is ChatGTP, a chatbot that uses Natural Language Processing (NLP) to enable human-like conversation dialogues. NLP can be described as a field of Computer Science and Artificial Intelligence that focuses on enabling computers to understand, interpret, and generate human language. The field aims to enable machines to perform tasks that traditionally require a human-level understanding of languages. These types of tasks are for example text summarizing and speech recognition, among others [38] [43].

Even if Artificial Intelligence is a commonly used term, the term itself encompasses various underlying layers such as Machine Learning and Deep Learning. Machine learning or ML, in comparison to Artificial intelligence, does not only mimic the human way of thinking but instead uses data-trained models to perform different tasks [9]. This discipline of AI focuses on building computers that improve automatically through experience. Machine Learning is a heavily researched topic that rapidly has been evolving and is today used for evidence-based decision-making [18]. Figure 1 illustrates the different layers included under the field of AI.

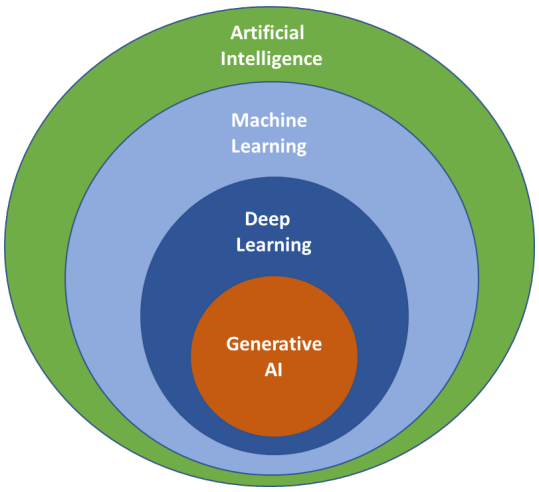

Figure 1: Illustration of the layers of AI [56]

Continuing on Figure 1, the next layer after Machine learning is Deep learning. Deep learning is a subset that uses Artificial Neural Networks (ANNs) with multiple layers. These Neural Networks aim to mimic the behavior of the human brain [16]. Initially, Machine Learning faced difficulties in limitations with processing natural raw data which caused the

development of Deep Learning. The underlying Artificial Neural Network (ANN) is a deep learning method that was inspired by the concept of the human brain [10]. The similarities between the Artificial and Biological Neural Network, are clear since the ANN also includes nodes that behave similarly as neurons in our brain. These nodes are organized into layers, which is where the information flows through. The connection strength between the nodes is adjusted during training to learn from data, which is essential to recognize patterns and make predictions [25][44].

Lastly from the illustration, the final layer is Generative AI. Artificial Intelligence Generated Content or AIGC is a type of technology that creates different types of content by using Generative AI, which for example is seen with ChatGTP which creates text in different forms, but also DALL-E which creates images based on prompts [6]. From a technical point of view, AIGC extracts human instructions to generate content satisfying the instructions. The AIGC technology has advanced tremendously by training more sophisticated generative models on larger datasets[6].

One of the specific large models in deep learning is the Large Language Model (LLM). These models are pre-trained on colossal amounts of data. Underlying the model is a set of Neural Networks, which act as transformers in the system. The networks as a whole consist of an encoder and a decoder with self-attention capabilities. Why Large Language Models have an important role in today's technological evolution, can partly be because of the models' flexibility [2]. As previously described with Natural Language Processing, LLMs are a subset of NLP and can therefore perform such tasks. However, since LLMs have trained on significantly bigger datasets, these models also tend to perform significantly better. LLMs can be described as an advancement of the NLP capabilities, that not only can produce a summary like NLP, but also can produce, manipulate, and comprehend text in a human-like way, due to the underlying neural network of LLMs [46].

Because of this, LLMs have shown great potential in solving problems and potential to disrupt content creation, as described with Artificial Intelligence Generated Content. Some other areas where LLMs have had a great impact is in the way people use search engines and virtual assistants [2]. ChatGTP as mentioned previously is one of the products that utilizes LLMs to solve tasks such as summarizing text, answering questions, creating content such as presentations, and so on.

Other companies that have utilized Large Language Models to solve other specific problems are for example Google with the underlying LLM called PaLM or Meta with LLM called Llama. Google Duet, launched by Google, is among other features, a Code Assistant, designed to facilitate different tasks such as generating code blocks based on comments [11]. This has been widely applied to for example create Unit Tests, with a large research block investigating this specific use case. One part of Software Development that is considered to be resource-intensive, and could therefore benefit from automation with LLMs is Quality Assurance.

1.1 Quality Assurance in Software Development

A significant part of developing software is the ability to test the software quality. The assigned role for this is called Quality Assurance Engineer and is a role with many responsibilities in the development process. As a Quality Assurance Engineer, also called a QA, the role often acts as a bridge between stakeholders and developers, interacting with various

development team members [1]. Commonly the following phases are used.

- **Designing the tests**: The QA checks in each stage that the product meets all the requirements in both expected and unexpected scenarios. To check this, QA engineers design tests to identify issues that could appear.
- **Executing Tests**: Running the designed tests and documenting the results.
- **Analyzing Tests**: Analyze the results of the unexpected outcomes and ascertain if any areas have not been assessed.
- **Enhancing Tests**: As the testing cycle continues, QA engineers modify and redesign tests to improve quality depending on new findings.

There are numerous scenarios that a product needs to be tested against to ensure that the product being delivered to the users holds the standard that the organization has defined. These scenarios are tested through many different methodologies that address the testing phases that were previously described. The first testing phase meaning the test design is a process that defines how testing has to be done. This process involves the steps of identifying the testing techniques, test scenarios, test cases, test data, and expected test results [49].

Because of the amount of steps included in this first testing phase, the phase can therefore be time-consuming for the Quality Assurance Engineers. The specific part of creating Test Cases requires the QA engineer to think of all of the specific scenarios that the user can face. If the product is less complex, these steps might not require as much time to identify. But as soon as the product complexity is higher, this also means that the identification of scenarios might be more difficult to map. For this reason, the exploration of using Large Language Models in order to map user scenarios in the form of Test Cases is interesting to increase QA productivity.

Even if the exploration of using Generative AI in the context of software testing has been gaining more traction, the specific use case of using Generative AI for facilitating the Test Planning phase is still very limited. As QA needs to test the software against many threats and scenarios, it can be considered a complex problem the AI algorithm needs to face. In this study, Generative AI in the context of Software Testing is investigated to solve the specific task of generating Test Cases with Large Language Models.

1.2 Objective

This study is an early phase exploratory study, that investigates the potential of using Large Language Models to automatically create Test Cases. The study aims to implement and evaluate:

- A system that simplifies the Test Design phase by creating automatic test cases by using Large Language Models.

The objective of the study is to use Generative AI to enhance the test case design process by creating or complementing the existing test cases designed by Quality Assurance Engineers. The following research question will be answered through the project:

- How does the integration of AI-driven test case generation in software development pipelines impact the overall software quality and development efficiency?

2 Background

This background chapter is divided into three main parts. The first section introduces a breakdown of Large Language Models and important terminology. The concept of LLMs is broken down into sections discussing Embedding, Transformer Architecture, and Scaling Laws. The second part of the chapter addresses the commonly used customization techniques for Large Language Models. The relevant customization techniques are discussed, namely Retrieval Augmented Generation (RAG) and Prompt Engineering. The third and final part of this chapter addresses the concept of Software Testing. The chapter aims to introduce the reader to the methodologies used when testing and specifically break down the concept of White Box Testing.

2.1 Large Language Models

Large Language Models, as described in the previous chapter, are a type of Artificial Intelligence that uses Deep Learning techniques in combination with large data sets, in order to solve diverse tasks. As a concept, LLMs are not a new concept but rather have evolved with Artificial Intelligence. From an evolutionary point of view, earlier language models had a greater focus on generating text data, while newer models like Google's PaLM focus on complex task-solving [64] [7].

2.1.1 Embedding

One commonly used concept when discussing Large Language Models (LLMs) and Natural Language Processing (NLP) is Embedding. Humans unlike computers can interpret different nuances in a language. From the human brain, we make connections such as understanding that a king is also a man, without it needing to be clearly stated in the text. But from a computer's point of view, the language or data, needed to be understood must be represented as numbers. These numbers are illustrated as a point in a vector space. The process of converting such data into numbers is a process called Embedding, and is a heavily researched topic due to its essential role in data analysis [58] [15].

The following Figure 2, illustrates four points representing the words man, woman, king, and queen. Depending on the word's numerical illustration we can make the connection that a king and a man are related, similarly to a woman and a queen.

Following the previous example, the human brain is able to understand that a King also is a Man, or that Rome is a city in Italy, which is essential in order to obtain the desired information. If the word would directly be encoded, this would also mean that the computer would not be able to connect the words in the same way as the human brain does. In order to practically solve this issue *Word Embedding* does not only identify the word but also successfully identifies the semantics and syntax of the word to build a vector representation of this information. The main objective is to capture the characteristics of the text rather

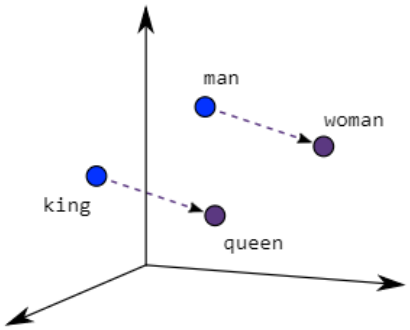

Figure 2: Illustration word embedding [15]

than obtaining just the features as numbers [24] [26].

A different approach to embedding, in comparison to Word embedding, is *Contextual embedding*. This technique shifts focus from word representation to giving a contextual representation that can capture many syntactic and semantic properties of words under diverse linguistic contexts. This type of embedding technique has through different studies shown outstanding results in a range of commonly applied NLP tasks, such as text classification and text summarization [29].

As previously described, contextual embedding aims to capture the overall sentiment and syntax of the entire sentence. For example can the word 'Mouse' have different meanings depending on the context, such as a computer mouse or an animal called mouse [53]. If we would consider a text corpus being represented as an input sequence S of tokens $(t_1, t_2, ..., t_N)$, then contextual embedding could be represented as t_i associating each token that is a function of S, meaning $h_{t_i} = f(e_{t_1}, e_{t_2}, ..., e_{t_N})$. Each of the input tokens is represented as t_j which is firstly mapped in a non-contextual representation e_{t_j}, before applying an aggregation function f [30] [39].

2.1.2 Transformer Architecture

Technically, when referring to the term LLMs, this is typically a referral to Transformer language models, with hundreds of billions of parameters. These Transformers are trained on huge amounts of text data, which is used for example in in PaLM, LLama, and GTP-4. In the following section, the Transformer architecture is described to better understand the building block of all LLMs [54] [12] [35]. Figure 3 gives an overview of the architecture with its seven important components.

The first component of the system is *Input and input embedding*. This component is where the user provides all input tokens. In other words, this means the query or instructions provided to the LLM. But even if the user writes instructions to the model in text format, the model itself can't understand text the way humans do. Because of this, the input needs to be embedded, or in other words converted into numbers, so that the computer can understand it. To understand more on how embedding works, read section 2.1.1.

The second component of the system is *Positional Encoding*. This component focuses on understanding the order of words that build up a sentence. The order of the words will play a crucial role in understanding the sentence's meaning and can be obtained through positional encoding. When each word has been encoded, these numbers can be fed into the

Transformer model, along with the input embeddings. If the transformer architecture is able to obtain both, the positional encoded and input embedding, the LLM can more effectively understand a sentence and generate grammatically correct and semantically meaningful output.

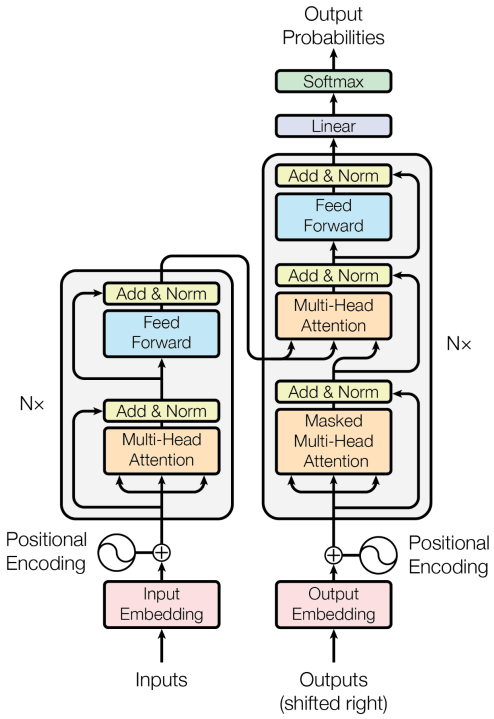

Figure 3: Overview of a basic transformer architecture [54]

The third component of the system is *Encoder*. This component is a part of the Neural Network that processes the input text and generates a series of hidden states that capture the meaning and context of the text. The component works as a tokenizer, where the input text is split into a sequence of tokens. From this point, a series of self-attention layers is applied, meaning that the component generates a series of hidden states that represent the input text at different levels of abstraction [54].

The fourth component of the system is *Outputs (shifted right)*: This component predicts the next word in a sequence by looking at the words before it, which is possible from the huge amount of training data. Technically the component works by moving the output sequence over one spot to the right, so the decoder can only use the previous words to gather an understanding on what the next word is.

The fifth component of the system is *Output Embeddings*: Similarly as described in the first component, the output needs to be converted in order to be understood by the computer. Positional encoding is also applied at this stage. Lastly, a loss function is used in machine learning, which measures the difference between a model's predictions and the actual target values. This function adjusts some parts of the model to improve accuracy by reducing the difference between predictions and targets to improve the overall performance. This is used during two scenarios, training and inference. When training, the loss function is computed in order to update the model parameters. During during interference, the output text is produced by correlating the model's predicted probabilities for each token with the

corresponding token in the vocabulary.

The sixth component of the system is *Decoder*: The input representation that has been positionally encoded, and the positionally encoded output embeddings need to at this stage go through the decoder. In transformers, multiple layers of decoders are used to generate the output sequence based on the encoded input sequence. In the stages of training, the decoder component learns how to predict the next word by looking at the words before it, similar to the fourth component. In for example ChatGTP, the decoder uses natural language text based on the input sequence and the context learned by the encoder to generate the output.

The seventh and final component of the system is *Linear Layer and Softmax*: Finally after all other components have been applied, the last step is to transform the output embeddings into the original input space. The linear layer works by mapping the output embeddings to a higher-dimensional space. Lastly, the softmax function is used to generate a probability distribution for each output token in the vocabulary, enabling us to generate output tokens with probabilities.

2.1.3 Scaling Laws

As previously expressed, LLMs can be described as an advancement of the NLP capabilities, which is due to LLMs training on increasingly higher amounts of data. However Large Language Models does not only extend the data size, but also the model size and total compute. From different research papers, it has been demonstrated that scaling can directly, and positively improve the model capacity of LLMs [64] [60]. For Transformer language models, the scaling law can be represented through the *KM scaling law*, which firstly was proposed by a team at OpenAI (2020) [19]. The team proposed that the model performance will be dependent on Model size (N), Data size (D) and Training compute (C). Given the compute budget `c`, the scaling formulas are presented underneath, where `L(·)` symbolises the cross entropy loss in nats [64].

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \alpha_N \sim 0.076, N_c \sim 8.8x10^{13} \tag{2.1}$$

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \alpha_D \sim 0.095, D_c \sim 5.4x10^{13} \tag{2.2}$$

$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}, \alpha_C \sim 0.050, C_c \sim 3.1x10^{8} \tag{2.3}$$

In the same study by OpenAI, the three laws were derived by fitting the model performance with different values on the parameters Model size (N), Data size (D) and Training compute (C), under some assumptions. The results showed that the model performance has a strong dependence relation on the three factors.

The scaling law is an important part of understanding LLM behavior and can in practice be used to instruct the training of LLM. As these laws have demonstrated their effectiveness in providing a qualitative performance estimation for larger models, their application to smaller models becomes particularly advantageous, notably in terms of time efficiency, enabling reliable estimations. Using large-scale models can be very time-consuming and also involve issues such as training loss spikes. Instead Scaling law can be used to monitor the training status of LLMs, and overall the previous studies show a tendency for performance increase when increasing Model size (N), Data size (D), or Training compute (C).

Predicting on a task level

The majority of the research specifically scaling laws, has been done on language modeling. However, in this study, the performance of Large Language Models on a task level is the more interesting part. According to a study on LLMs, a model with smaller language modeling loss, usually tents to perform better on downstream tasks [64]. GPT-4 is one of the models that have been reported to have some accurate predictions via scaling law, but the readers have also been notified that a direct decrease in language modeling law does not always indicate an improvement in model performance on such tasks. In some cases, language modeling can obtain worse results when loss decreases, however, the study suggests that task-level scaling law is more difficult to characterize due to it being more dependent on the task-related information.

2.2 Customization techniques

Large Language Models, as described, are showing tendencies of great potential. However, if a user wants to obtain the maximum results, numerous articles suggest that a certain type of customization is needed. In a general matter, four basic layers of customization are commonly applied to higher the complexity of the LLM behavior. These layers include: Integrate with Pre-Build Components, Prompt Engineering, Internal Knowledge, and Fine Tuning [13].

In this project, the two relevant layers of customization are Prompt Engineering and Internal Knowledge. In the following subsections, the concepts and technology of the customization layers are discussed.

2.2.1 Prompt engineering

The first layer of customization is Prompt Engineering, meaning the process of designing, refining, and optimizing the prompts or queries given to a language model to obtain specific responses [28]. Generally, this is a process that is highly involved in NLP but is mostly correlated with interaction with Large Language Models, such as GTP-4. This discipline is still considered to be relatively new, though the method is used to better understand the capabilities and limitations of LLMs [41].

Why Prompt Engineering plays a critical role, is because of its ability to tailor the specific behavior of the model to certain tasks and applications. The initial step of the process is to create a prompt, also called the input query. This query will be provided to the LLM and can be practically provided in different forms such as a question, an instruction, or a partial sentence. A well-defined prompt should provide enough information for the model to understand the user's intent[48]. How to effectually conduct the prompt has been discussed and can be broken down into a couple of components [45]. In Table 1, the commonly used Prompt Engineering component is shown, as well as a general description and importance level.

Table 1: Components to include when conducting a prompt, sorted from most important to least important.

Prompt Engineering		
Component name	Description	Importance level
Task	What you want the LLM to do. Start the sentence with an Action word, e.g., Write, Generate or Give. *Give me 3 healthy recipes*	Mandatory
Context	Gives the LLM a context to the question. *I am a 35 year old female*	Important
Exemplar	Helps understand the reasoning process. *Use the STAR answer framework: Situation, Task, Action, and Results*	Important. Not necessary for every prompt
Persona	Gives context to the model about how it should behave. *You are an hiring manager*	Nice to have but not needed
Format	Define the output format needed. *Give me the recipes in a table format*	Nice to have but not needed
Tone	Define the tone of the answer. *Casual, formal, or enthusiastic tone*.	Optional

2.2.2 Retrieval Augmented Generation

Retrieval Augmented Generation, also called RAG is, similarly to prompt engineering, a way for customization in the context of Large Language Models. In the context of retrieving information from LLMs, the most commonly used methodology is to communicate thought instructive prompts in some form of chat context. However, when a user interacts with the language model, they're essentially extracting information from the model's pre-trained data. Meaning that the user can only get an answer related to the data that the model has priory trained on. For that reason, it is common that large language models are not able to answer questions related to for example news events that have recently happened.

Because of this, the base models that are trained only on pre-trained data, include limitations in knowledge. One of the downsides of explicitly using such a model is: that the model cannot easily expand or revise their memory, can't straightforwardly provide insight into their predictions, and may produce "hallucinations" [27] [17].

Retrieval Augmented generation is a hybrid model that combines both paramedic memory (pre-trained data) with non-paramedic memory (retrieval-based memories). The usage of this type of hybrid model has been showing positive results since its ability to directly revise, expand, and access knowledge [17]. A practical example of this is: When a user interacts with an LLM similar to chatGTP, the user is able to ask a lot of different ques-

tions. Logical and historical questions are just some of the examples that the model could answer if the user queries a question. But as soon as the same user would ask a specific question like: "When is my friend Lisa's birthday?", the LLM would not be able to provide an answer. This is because the answer related to that question has not been found in the pre-trained data. In order to answer that question, the model would need some additional information. Which can be provided through this non-paramedic memory [27].

Tasks that are considered knowledge-intensive tasks can use the method called Retrieval Augmented Generation, or RAG. Originally the method was introduced by Meta AI researchers, who argued that such technology can customize large models without needing retraining of the entire model [42].

Practically the technology works by taking an input, being the query of the user, and using this query to extract relevant documents from a source. The input query is embedded, meaning that it is, therefore, a number symbolizing a point in a vector space. The documents are also embedded similarly and are symbolized as individual points in the same vector space as the input query. The degree of similarity between the input query and the documents can therefore be decided by extracting the points with the shortest distance from the input query point. The amount of documents to be considered can be specified as well as the minimum degree of similarity. The selected documents are concatenated as context together with the original input prompt, which together will be introduced to the text generator, which will produce the final output combining both the pre-trained data and relevant information from the knowledge base. Figure 4, describes an overview of the method.

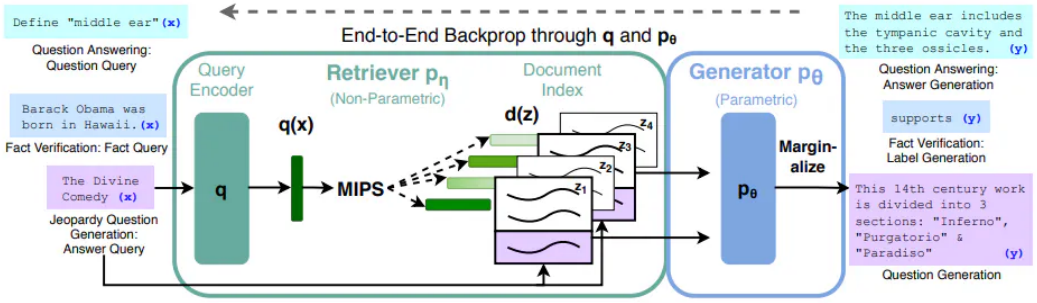

Figure 4: Illustration of Retrieval Augmented Generation [27]

As described in the previous paragraph, Retrieval Augmented Generation works by extracting relevant data to the input query. In the last paragraph, documents were described, but similarly, the model can extract relevant data on tokens or chucks. The model that is relevant for this study is called $RAG-token$ and allows the generator to choose content from several documents since the similarity depends on tokens instead of whole documents [27]. Formally, the $RAG-token$ model is defined with the following equation:

$$p_{\text{RAG-Token}}(y|x) \approx \prod_{i}^{N} \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z_i, y_{1:i-1})$$

2.3 Testing Method

In Software Development, Software Testing is a process aimed to evaluate and verify that the developed system is behaving as expected. As an essential part of assuring quality in products, this also required a variety of tests to be used that can mimic the different scenarios that the software can face. Unit testing, Integration testing, functional testing, and performance testing are just a few of the tests that are commonly required before realising a new product [23]. Among the commonly used techniques in Software Testing is White Box Testing.

2.3.1 White box testing

White Box Testing is a technique that mainly focuses on testing the structure of the application, meaning the code structure, logic, and flow [21]. The method practically uses the control structure to create a Test Case Design, which is created to map and prevent implementation errors that could appear in the application. Test Case Design as first introduced in section 1.1, focuses on checking that each stage of the products meets all the requirements in both expected and unexpected scenarios.

The method of using White Box Testing is applicable on multiple layers of the Software Testing process, including Integration, Unit, and System levels. The testing technique requires source code access and understanding of the source code since this is the base of the process. Some of the testing techniques used in white box testing include control flow testing, path testing, data flow testing, loop testing, and branch coverage testing [55] [23]. Figure 5 illustrates the different types of white box testing. These can be discussed more in-depth but from the context of the study, will only be mentioned [57].

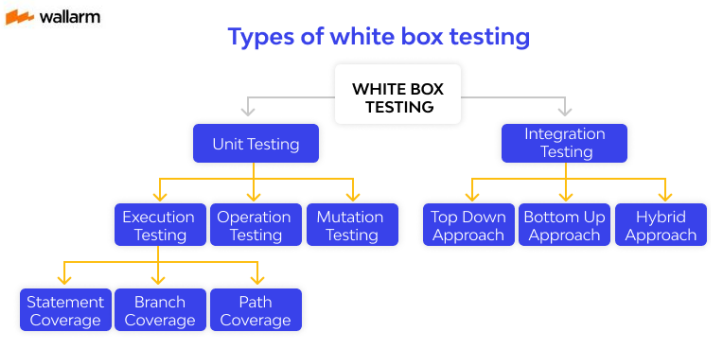

Figure 5: Illustration of different types of White box testing

In order to understand the study, the practical working process of White Box Testing needs to be understood. The general approach to performing White Box Testing is to:

1. Study the source code
2. Find Test Cases and execute

Even if the first point of the general approach is fairly straightforward, the second point

follows a more generic framework. To obtain this, the following procedure is applied [20] [57]:

1. **Input:** A List of requirements, functional specifications, design documents, and source code.
2. **Processing:** Perform risk analysis, meaning identifying and analyzing potential future events that may adversely impact a company.
3. **Test Planning:** Design test case, meaning map which specific actions are required to verify a specific feature or functionality in the software. This is presented as detailed steps, data, prerequisites, and post conditions necessary to verify a feature [5].
4. **Output:** A final, often manually written.

The traditional way of creating test cases is to manually write the different scenarios as text, which describes all possible actions that are required for a step. These steps shall also discover all the different possibilities a system could face. If the application or product is large in size this causes problems of complexity, since the tester needs to understand and identify all possible paths. This type of testing is also considered highly expensive since each redesign of code or rewriting of code requires a new set of test cases. Another noticeable disadvantage is that the tester requires a high level of expertise, which not only contributes to the expenses being high but also elongates the knowledge gap between different domains in a software team.

3 Method

The following chapter contains the detailed methodology for conducting the experiment. In order to achieve the desired results, the methodology used follows a structure equivalent to the Design Thinking process. Design Thinking as a concept can be described as an iterative, non-linear process of close collaboration between designers and users. Figure 6, describes the commonly known stages of Design Thinking, namely, Identify and Define, Research and Plan, Produce, and Implement and lastly, Test and Evaluate [22].

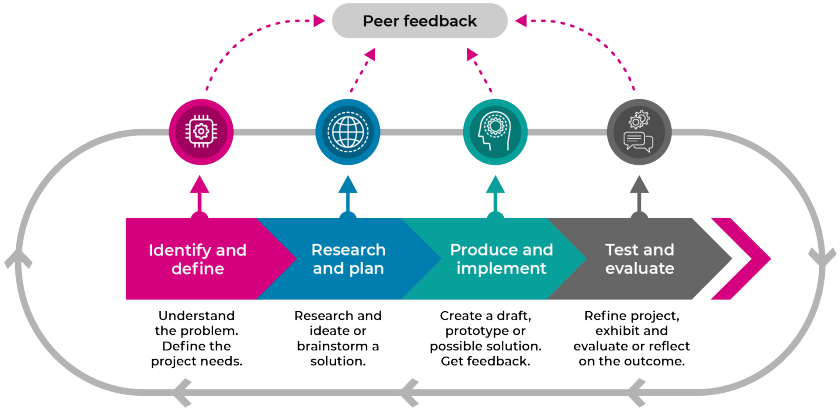

Figure 6: Design Thinking exemplar [59]

3.1 Identify and Define

The first phase of the experiment involves the steps of Identification and Definition. This stage is dedicated to carefully identifying and presenting the underlying problem. The primary focus is on the systematic gathering of key information, to attain a comprehensive understanding of the current issue. To obtain this: the study uses Structured Interviews as the primary method for information gathering. Subsequently, these interviews will later be decoded and used to construct a Persona, enriching our insights into the user experience [62] [22].

3.1.1 Interviews

The first method used to gather information and understanding of the problem is by using interviews. The type of interview used is a Structured interview, meaning a systematic approach where the interviewee asks the same predetermined questions to all candidates in the same order. This method is considered to be approximately twice as effective as a traditional interview where the interview follows an unstructured standard [3]. For the sake of this project, structured interviews have been used, but similar results can be obtained through other interview styles such as semi-structured or unstructured interviews [61].

The interview was conducted with 15 questions and was estimated to take 30 minutes to complete. The target group to participate in the interview was people of any gender with a Quality Assurance background. All interviews were held in-house in the PayPal office, and conducted between myself and the selected PayPal employees. The structure of the interview followed the following category.

1. Question with User Profile focus
2. Questions with Test Case focus
3. Questions with Artificial Intelligence focus
4. Questions about ethics and Artificial Intelligence

The whole list of questions is available in the project appendix A.

3.1.2 Persona

The second method for identifying the problem is to also understand whom the project aims to develop a system for. In order to identify the user needs, one part of the methodology is to create a persona. A persona can be described as a specific model of a user, that aims to resemble a typical user profile. Typically this persona is used to represent a specific user pattern, behavior, or other motives [4].

According to the study, one important part of the creation of a persona is to define the goal. Meaning that the goal of the user should be used to direct the design. A persona is also an effective way of mapping and understanding the motives behind the user's actions [4]. By using this purpose throughout the whole process of Design Thinking we can obtain a user-centered and goal-directed design. Figure 7 illustrates the used template when creating a persona.

Figure 7: Persona Template

3.2 Research and plan

The second stage of the Design Thinking process is to research and start conducting a brainstorming session for possible solutions to the identified problem. This can be created from

different methods such as *Mind mapping* or *Rapid ideation* [33].In the process of this study, the Research and plan phase was heavily conducted based on a literature review. The review has focused on the current exploration stages of automatically generating test cases.

3.2.1 Literature study

The literary study was conducted by reviewing an extensive amount of scientific articles and studies, which all have a base in Artificial Intelligence in Software Testing. The literature used for this part of the study was obtained through websites like IEEE or similar. In order to get relevant articles to base the literary review on, keywords such as Automatic Software Testing, AI for Software Testing, or AI Test Cases were used.

3.2.2 Exploring the company resources

The second methodology used in the Research and plan phase was to explore the company-wide resources. If this study were reproduced, this method is not necessary to the same extent, and other solutions can be found. Naturally, the exploration of company resources will vary depending on the company, but overall the following key points were taken into consideration when exploring the solution from a company point of view:

- What already existing tools can be found on the company cloud?
- What restrictions exist for using the specific tools?
- What access requests are needed?
- What is the legal point of view?

3.3 Produce and Implement

The third phase in the design thinking process is called Produce and Implement. This stage can also be called the prototyping stage in other similar models to the design thinking process. From a technical point of view, the phase focuses on prototyping a possible solution, based on the information of the previous stages. In this context, the previous stages of the process gave sufficient information to set the direction of the project. The way of automatically generating test cases was chosen to be through Generative AI, and the specific model to use was chosen according to the company standards. In order to prototype and implement a possible solution. The following model was used:

Model name	Llama-2-7b-chat-hf
Model size	7B
Version	Version 1
Framework	OpenAI

Table 2: Model information for the Llama2 model

The model described is Llama2, which is provided by Meta [36]. This model is used as the pre-trained model to communicate with. The model is hosted in-house and accessed through a private endpoint, but the study could in practice be implemented by saving the model locally. To further implement the system to not only understand the knowledge of

the outside world (pre-trained data) but also understand the specific organization. The next steps of the implementation include setting up the knowledge base that is used to set up the Retrieval Augmented Generation.

3.3.1 Create Knowledge base

To create a knowledge base with the needed information from the company, the knowledge base is conducted through two main steps. Firstly the needed information is gathered, and then secondly the information needs to be stored.

Gather the data

The first step to creating the knowledge base is to gather the required data to store. For this specific implementation, the only supported file type is Text file (.txt). The files can in practice contain all forms of information that you want the answers to the LLM to take into consideration. This can for example be specific User Journeys or other useful information to solve your specific task.

In this specific use case, the end goal of the interaction with the LLM is to create Test Cases. Therefore the information that is relevant to store is documents explaining the specific company domain, which in this study is limited to Onboarding Experience (OBEX). For example, these documents can contain user journeys, screens, components, etc. In an optimal solution the data should also container previously created Test Cases for other projects in the same domain. Including project: Description, Requirements, and Acceptance criteria.

All the desired documents are stored in one local folder, and loaded to the system by using the class `SimpleDirectoryReader` which is provided by llama_index [32]. The documents are loaded according to the following:

```
reader = SimpleDirectoryReader(input_dir="../Name_of_folder/")
docs = reader.load_data()
```

This information can be gathered through numerous ways, such as manually extracting text from the company management tool or through API. In this implementation, the texts are imported manually to the local folder.

Save as Embedding

After the desired data has been gathered, each of the documents is loaded into the system, before each documents is slit into smaller pieces [50]. This is a process called Chunking and is a commonly used process in Natural Language Processing. There exist different methods with the goal of chucking but for this implementation, a simple approach is used.

The approach for chucking the documents is to follow a similar approach as with the method `CharacterTextSplitter` from `Langchain`. However, this method is not compatible with the kernel where the in-house LLM is implemented, meaning that the Langchain package cannot be used. A manual way of chunking is instead applied with similar character as `CharacterTextSplitter`, which splits text based on characters (by default ”\n \n”), however, this is not recommended if the study would be replicated.

The next step of the process is to embed the chucks. When embedding the different chucks, this is made by using an embedding model. There are numerous versions of em-

bedding models that could be used, but similarly to the LLM model, the in-house embedding model is used, namely `llama-2-7b-chat-hf`. The following code section is an example provided on the OpenAI website, on how to obtain embeddings:

```
from openai import OpenAI
client = OpenAI()

def get_embedding(text, model="text-embedding-ada-002"):
   text = text.replace("\n", " ")
   return client.embeddings.create(input = [text], model=model).data[0].
       embedding

df['ada_embedding'] = df.combined.apply(lambda x: get_embedding(x, model
    ='text-embedding-ada-002'))
df.to_csv('output/embedded_1k_reviews.csv', index=False)
```

The way to produce this can vary depending on the organization's standard and procedure but the results of the embedding have during this implementation been decided to be stored locally. Similarly, as the example provided above, this implementation saves the embeddings locally in a CSV file. Alternative solutions to this, are to save the embedded documents in an embedding database such as Chroma [8], or in a cloud solution such as Azure OpenAI [37].

3.3.2 Overview of the system

Figure 8 illustrates an overview of the system as a whole. The figure shows different documents being embedded and saved in a vector space, which in this implementation means the locally stored CSV file. From a user perspective, the user writes an input query that similar to the documents are embedded with the same embedding model. The embedded query and the embedded document chunks are compared to each other and the `K` best document chucks are selected. As described in section 2.2.2, the document chucks with the highest degree of similarity, meaning the embedding points with the lowest distance from the input query embedding point. The chunks with the highest degree of similarity are then sent to the LLM together with the input query as context, to combine the information from the knowledge base with the pre-trained data, and from here produce an output.

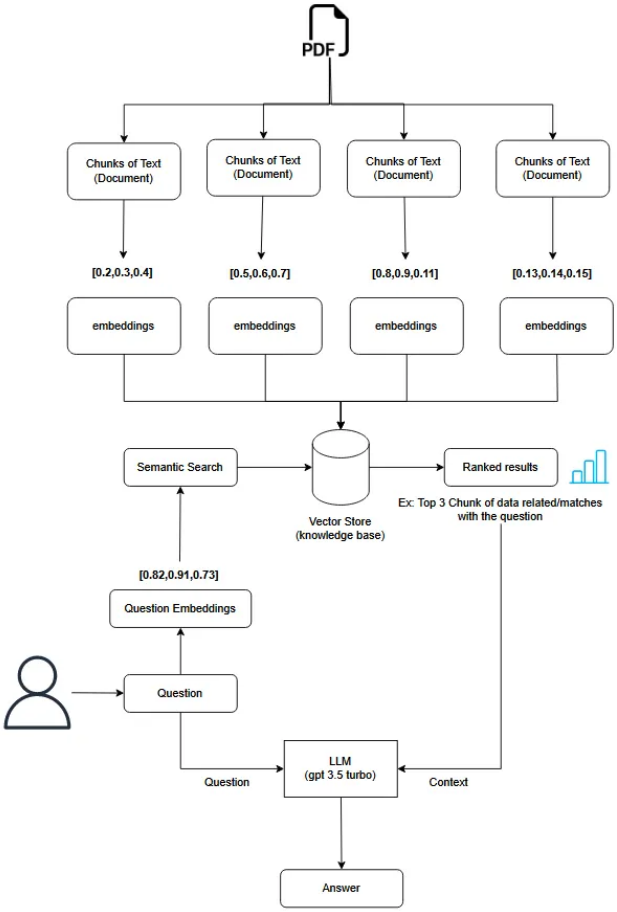

Figure 8: An overview of the system. It's important to note that this implementation is specifically designed for txt files, even though the image depicts a similar process for PDFs

3.3.3 Prompt Template and Llama parameters

The last part of the methodology in the Produce and Implement phase, has a focus point in finding a good template for prompt engineering. From the previous stages in the Design Thinking process, the end-user of the system has been identified. In the process of this, we have also identified some potential noise when using the system. To minimize this noise, and to guarantee that the results given to the end-user are similar in quality, a suited Prompt Template needs to be identified.

To find the most appropriate template for the context, the information provided in the Background 2.2.1, is used to create a well-suited prompt. In order to test this template, and to guarantee that the results were matching the expectation. A playground of Llama2 is used [31]. This playground makes it available for the user to tune the following parameters:

- Model size *Larger size means smarter but slower.*
- System prompt *This is used to help guide system behavior*

- Temperature *Adjusts randomness of outputs*
- Max Tokens *Maximum number of tokens to generate*
- Top P *Samples from the top p percentage of most likely tokens*

The playground is used to adjust parameters and find the most suited for the context. In order to test this the following prompt template was used:

I am a Quality Assurance Engineer and need you to act as an expert in software testing. Generate all test cases for project DTPAAABBBDG-1241. The test cases should follow a table format with columns: number, scenario, test steps, and expected outcome

To facilitate the implementation, the knowledge base includes a document with the project id DTPAAABBBDG-1241, where in the same document the project description, requirements, and acceptance criteria are located. The same results can be obtained by stating these directly in the prompt.

3.4 Test and Evaluate

The fourth and final phase in the design thinking process is called Test and Evaluate. This phase focuses on exhibiting and evaluating the created system. To do this, the project is refined and a specification on how to practically evaluate the system is created.

For this project, the evaluation of the system will be conducted through both quantitative and qualitative results. The way of evaluating the system is by practically answering the following questions:

1. Is the output of the system responding to the accurate task. Meaning the system gives Test Cases with the correct format, columns, names, etc.
2. How many of the test cases are accurate to the specific project? This means giving the defined inputs, how many of the Test Cases are useful (%). This result is obtained by comparing the output from the created system and comparing this to test cases created manually by a QA.
3. How many of the Test Cases are considered noise? Meaning non-accurate data, hallucinations, etc (%). This result is obtained by comparing the output from the created system and comparing this to test cases created manually by a QA.
4. Specific results representing Happy paths respective Edge Cases.

Defining expectation of the system

Giving the following input the system is expected to give the following output:

- Input: A prompt including project-id linked to the project Description, Acceptance criteria, and Requirements as a string.

- Output: A table with columns; Test case number, Scenario, Test steps, and Expected outcome. All test cases involved in each scenario should be in the same cell.

Useful terminology to interpret the results [52]:

- Test Case: Tasks needed to validate a particular feature or functionality during software testing.
- Happy path: Specific type of test case, conducting testing solely based on the acceptance criteria for the feature.
- Edge Case: Specific type of test case, exploring scenarios beyond the foundational assumptions, discovering alternative ways to utilize a feature that may not have been initially intended.
- Noise: Unwanted or random interference. This can for example be hallucinations, incorrect outputs, or similar.

4 Results

The following chapter contains the main results of the study. In order to present the results in an effective way, the results are divided into parts representing the steps of the Design Thinking process. The first part of the result section displays the findings from the interview, which is taken into consideration to create the Persona for the project. The interview section below provides an overall understanding of the results, which is decoded from the interview answers.

4.1 Interviews

The interview conducted was answered by 5 interviewees, all of the whom are a part of the Quality Assurance chapter. The 15 questions asked during the interview have given a variation of answers that are sufficient to gather an end-user understanding. In the following paragraph, the interview results will be summarized according to section 3.1.1.

From the first section of the interview, the goal was to get to know the overall user being interviewed. The results from the first section indicate that all the people being interviewed have a Quality Assurance background, but the length and involvement in the current company can vary in a range of 2-5 years. One of the interviewees also has a background in Academic Research which is also important to keep in mind when analysing the results.

The second section of the interview had the goal to gather an overall understanding on the Test Case generation and experience of the interviewees. The entire process of creating test cases is documented and the journey and collaboration with other roles in the company is carefully being considered. Thought this section, we could gather understanding of what the relevant documents are when creating test cases in the current stage. As the interview reveals, currently the interviewees use documents such the project requirement and acceptance criteria. Overall the interviewees tend to collaborate together with the developers, designers and project managers in order to gather an overall understanding on what is expected in the scope of the project.

The interview results also indicated a variety of answers in the question regarding frequency of creating test cases. Most of the interviews suggested that the amount of test cases is highly related to the type of feature. If the feature is new or considered more experimental. Then the test cases tend to be created and updated more frequently. From the answers we can also conclude that the process related to creating test cases isn't explained as annoying but specifically the word *repetitive* is used for the majority of interviewees. This could be explained because of the Test Cases being created for the specific QA to map the feature and possible user journeys. But since the document is rarely reviewed by other project roles, the act of updating the document, falls behind and becomes less prioritised than other responsibilities related to the QA role. This leads to the documents sometimes being outdated.

The third part of the interview focuses on Artificial Intelligence, both with question related to experience and more practical exercises. All of the interviewees had experience in using Large Language Models, but all of them had only used ChatGTP and not another LLM. The majority of the interviewees have not used such LLM tools to solve work related tasks, but mostly uses the tools to solve linguistic tasks such as rephrasing or summarizing. One of the interviewees has stated that the tool also is commonly used for that person to get inspiration or to used as a starting point in creative sessions. Non of the interviewees had any clear knowledge of the company specific LLMs nor where these can be found.

From the more practical exercises. The last questions of the section involves a practical exercises that reveals important capabilities related to prompt engineering. The simplified instruction that was given to the user to was the following questions. The goal of the exercise was to formulate a prompt to retrieve as precise and qualitative answers as possible.

- Q1: 3 recipes for a blog.
- Q2: An analysis of the book A Little Life.
- Q3: Test case for a simple login function at Facebook.

The outputs provided by ChatGTP was positively received by all of the interviewees. The interviewed people expressed a positive response both from the matter of quality of the test case provided, but also about the coverage. The results however clearly indicate a lack of knowledge of Prompt Engineering. From the suggested theory 2.2.1 the results show a clear indication of prompt engineering not being applied in a sufficient matter. Figure 9 illustrates how many of the interviewees actually use the components previously discussed in section 2.2.1. As the figure illustrates, all five interviewees use the component Task, while some interviewees occasionally use Context. But a very limited representation of the other components is found.

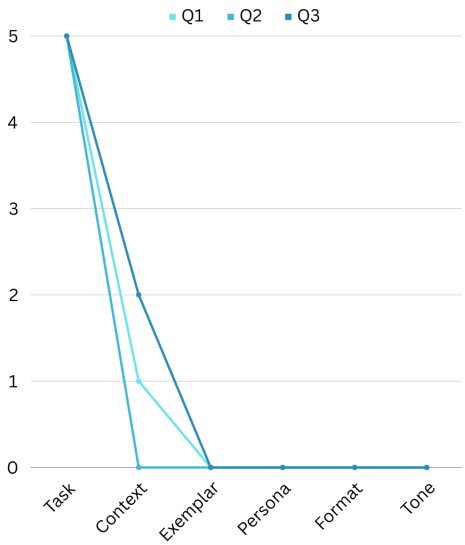

Figure 9: Component representation during interview prompting exercise

The last part of the interview contextualise the fears and worries of involving Artificial Intelligence in the process of Software testing. From an ethical point of view, considering Generative AI possibly being applied to create Test Cases, the interviewees expressed little

fear for the specific task. However multiple interviewees expressed a worry regarding the possibility for Code and Company leakage. This is however not directly applicable in this case but rather a general worry when involving AI around more sensitive information. All of the interviewees however, did express that Test Case generation usually does not involve a very high degree of sensitive data, or has a business critical function in the product, which contributes to the fear and worry being decreased.

4.1.1 Persona

From the interview answers a persona was able to be created. This persona is intended to act as a reflection of the intended end user.

Figure 10: Created Persona

4.2 Evaluate the system

By evaluating the system according to section 3.4, the following results where obtained. For the first evaluation point: *Is the output of the system responding the accurate task. Meaning is the system giving Test Cases with correct format, column names relevant to the specific project*. The results show that the implementation correctly uses the project-id to gather information and overall produces test cases in the correct format. Specifically meaning that the format given by the system is in a table format with the corrected columns and overall composition. However, the column ”Test Steps”, seems to at times, be getting inconsistent result. All steps in a specific scenario are not always formatted in a numerical list, which is likely to be due to the limited data of previous test cases.

The following Figure 11 represent a data comparison between the AI system output and the test cases created manually of one specific project. The bar plot shows a side-by-side comparison between these, with y-axis representing the amount and x-axis representing categories being Happy Paths, Test Cases, Edge Cases and Hallucinations. As seen in the figure, the category Hallucination only appear for the AI-system bar since this is only applicable here. The figure also reveals a result of 0.5 in hallucinations on that specific project, which symbolises that the test case is not in particular fully noise but rather contains parts of noise.

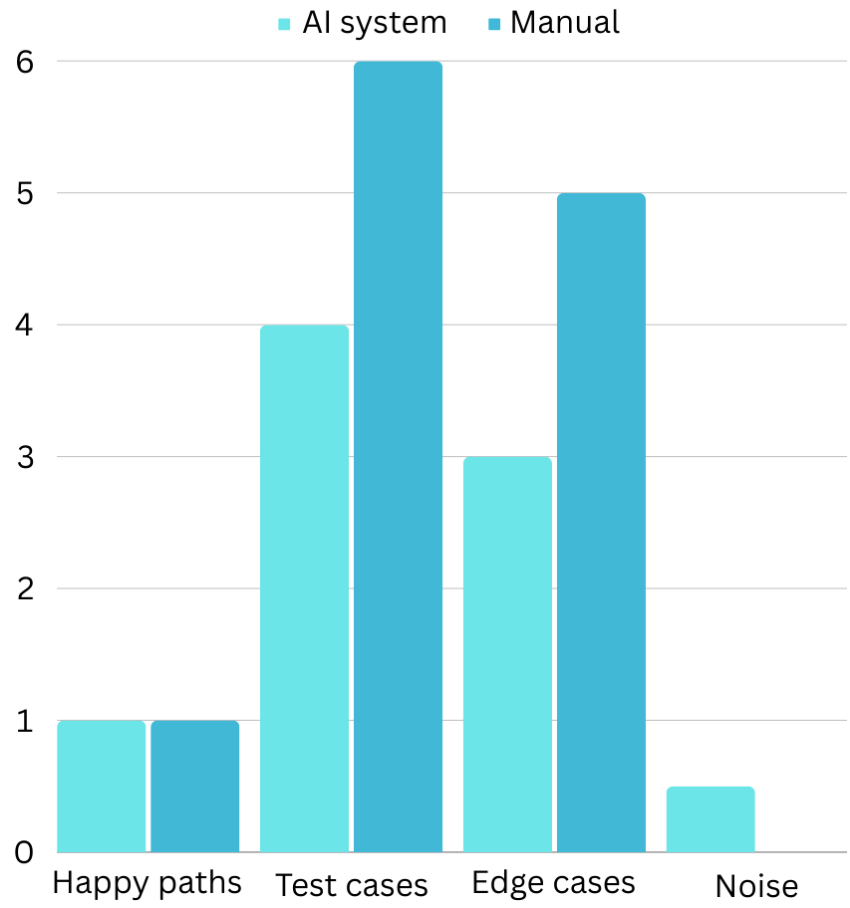

Figure 11: Data comparison between AI system and manually created test cases

The following Table 3, reveals some specific comparisons between the AI system and Manual way of writing test cases. As seen in the table, the time decreased significantly by using AI-driven test case generation, however it is important to keep in mind that only 66.67% of the test cases are practically useful but could contain minor noise. The project used to test this, is the same project revealing the numbers from Figure 11 and is a relatively specific and small project. The table also reveals the similarity score discussed in Section 2.2.2. As seen in the table the K, in this case five, most similar chucks selected to be used to solve the query have a similarity score between 0.669 and 0.511, where a score between 0 and 1 is possible. A higher similarity score indicates a higher probability that the selected document will contain relevant information to the input query.

The following Figure 12, deep dives into the reason why a specific test case have been categorised as Noise in Figure 11. As previously discussed, noisy data or meaningless data can be due to many reasons. Some of the data is categorised as a Hallucination can be described as ”generated content that is nonsensical or unfaithful to the provided source content” [63]. For example this is when the system starts producing something entirely unrelated to the prompt. Another reason why something can be categorised as noisy is if the produced outcome is non accurate. Meaning it is not complete nonsensical, as for a hallucination, but still isn’t accurate to the project. For example this could be including screens or components that don’t exists etc. Another reason why test cases can be categorised as noise is due to Non accurate tech definition. Which in this project is defined as a tech def-

Table 3: Time and usefulness summary

title	
Time per use case	AI system: 23.75 seconds per Use Case, Manual: 600 seconds per use case [40]
Overall usefulness of AI system	Useful test cases: 66.67%
Highest similarity between query and documents (K=5)	• relatedness= 0.669 • relatedness= 0.656 • relatedness= 0.615 • relatedness= 0.556 • relatedness= 0.511
QA found useful	Yes 4/5

inition that is not understood correctly. For example a tech acronym SPI is confused in the system to be Serial peripheral interface (SPI) instead of the meaning in the organisation, which is Service Provider Interface (SPI). Because of confusion in tech acronyms this can cause inconsistency in the system and can make it harder for the system to draw an accurate conclusion. Since the results presented in Figure 11 are reflecting the results of a specific project, the pie chart 12 instead starts exploring outside the specific project. This pie chart represent the most commonly appearing noise the system tend to produce when retrieving information in order to produce queries. In the process of exploring the commonly produced noise, the Figure 12 reveals that most commonly, Non accurate data is produced, with a result of 62.5%. This implies that the generated data is not entirely nonsensical; instead, it may include information that falls within the project's scope but is not necessarily applicable. For instance, the system might generate a step in a test case commonly found in similar projects, even if it is not implemented in the specific organization at hand.

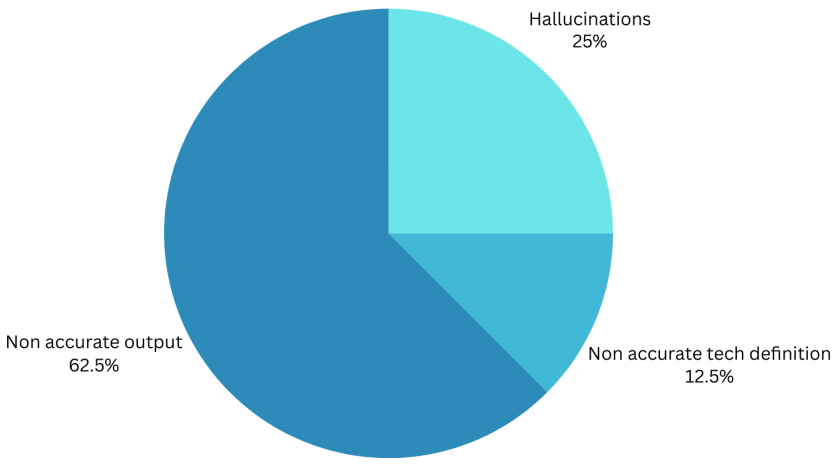

Figure 12: Reason for Noise label

Table 4 illustrates an output instance generated by the system. The output provided by the system, displays a Happy Path following a User Journey of a specific costumer type. The system output initiates with the message, ”Great, I’d be delighted to assist you in creating test cases for the *project-id* project. In accordance with the provided articles, here are some test cases for your consideration:” followed by the presentation of table 4.

Table 4: Example of system output given the defined input prompt from section 3. In order to make this table more readable the columns and rows are inverted in comparison to the actual system output. This table also just shows one specific test case, where the actual output is a collection of all test cases of a project.

Generated output	
Test case	1
Scenario	Successful signup flow for an existing PayPal merchant
Preconditions	The user has an existing PayPal account + The user has not previously signed up for the Zettle product.
Test steps	1. Navigate to the signup URL (*Specific sign up flow in stage*) 2. Enter the required information for the signup flow, including the PayPal account email address, password (correct), and business type (INDIVIDUAL). 3. Fill out the form with valid information (e.g. monthly sales of $1000, SSN number 123-45-6789, work mobile number 555-123-4567) 4. Verify that the signup process completes successfully and the user is redirected to the Zettle dashboard.
Expected outcome	The user should be able to successfully sign up for the Zettle product using their existing PayPal account.

5 Discussion

In Section 4, the given results are presented. As this study aims to investigate how the integration of AI-driven test case generation in software development pipelines impacts the overall software quality and development efficiency, we can from these results draw some concrete conclusions. The results show that the AI-driven test cases, generated through LLMs can generate output test cases in the correct format relevant for a given input. The system draws some conclusions resulting in the test case structure and uses the information from the knowledge base to create organisation specific test cases. During the Design Thinking first phase, *Identify and Define*, we can also conclude that the prompt engineering knowledge appears to be limited among the employees involved in the study. This finding was relevant to create a more coherent end-user friendly system for the intended end user presented in Figure 10. The quality of the test cases is shown in Figure 3, where the study compares the results of the AI-driven test cases against the Manually written test cases. To not only compare quality but also time, in order to draw sufficient conclusions towards the objective, Table 3, also displays results in the form of time, similarity score, and feeling of the system.

By specifically looking into certain areas of Section 4, we can make more discussions relevant to the study. As previously described, during the first phase of the design thinking process, Identify and Define, we can observe a clear indication concluding that the knowledge of Prompt Engineering is limited. Even though direct knowledge isn't tested during the interview, the interviewees show signs pointing toward that the practical application of prompt engineering is limited. In Figure 9 the usage of prompt engineering components is displayed in correlation to the practical exercises in the interview. As seen in the figure, only the *Task* component is consistently used by the interviewees, while *Context* is occasionally used. All the other components are not used in any degree which according to the Theory 2.2.1, is not recommended.

Based on the prompt style and in combination with the analysed quality of response from the LLM during the interview, we can conclude that the limited knowledge of prompt engineering could lead to inconsistent responses to the QAs. To limit this, the decision to use a given prompt template was made. This was created to provide the user an easy access to a proper way of prompting, which would potentially lower the risk of the answer quality being affected by the knowledge. Since the system is intended to be used by a typical user, as presented in Figure 10, the system outputs must provide reliable and consistent answers. Why the application of prompt engineering components is as limited as the results show in Figure 9 could be because of a lack of knowledge but also from the misconception of using a chatbot. Just as customers commonly formulate messages for *help chat-bots*, users might anticipate a similar level of ease when interacting with the LLM. While, from a practical standpoint, this interaction bears similarities, it's crucial to note that, for this particular system, the task demands a greater level of data and comprehension. This also leads to the user needing to make some more specific customization techniques such as Prompt Engineering. From a company point of view, the lack of knowledge in the domain of

Prompt Engineering could be avoided by teaching and spreading knowledge and awareness of the subject on a company level.

In Figure 11 in combination with Table 3, we can observe the direct relevant results for the objective of the study. Figure 11 shows the direct comparison between the AI-driven test cases against the Manually written test cases. As seen in the results we can see some capabilities that the system can write Test Cases. For example, the figure displays to which degree the system, with its current implementation, can identify the Happy Path in the test cases and some degree of edge cases. The representation of Edge Cases in these results shows a lower representation in the AI-driven test cases. Why this is appearing can be discussed to be because of the limitation in data. The implementation is created corresponding to the theory described in section 2.2.2. The system uses an external knowledge base in order to make conclusions not only depending on the pre-trained data but also take into consideration new data. The limitations of data in the knowledge base could be a reason why these specific results are appearing. One observed issue is that certain documents in the knowledge base may be somewhat outdated. Other problems might include that the lack of data in the knowledge base forces the LLM to make conclusions without necessarily having all the needed information, which could lead to assumptions being made and noise being produced.

On the other hand, one interesting observation is the Similarity Score provided in Table 3. The five best chunks to the given input gave similarity scores between 0.511 and 0.669, however even if the score does not necessarily appear high, the quality of the output is still fairly positive. This result is interesting due to the methodology presented in section 3. The parameters used when interacting with the model can be specified, taking into consideration this study. By these results, we can conclude that similarity scores don't necessarily need to be at a very high degree to produce interesting results.

If we look into the results in Table 3, the time-efficiency increase is clear in comparison to the manually written test cases. Which directly affects the *Core Needs* and *Frustrations* which are identified and summarised in the Persona 10. One other frustration that was identified during the interviews was the difficulties that QAs face when having to which between domains. Since this specific study is mainly investigating the generation of test cases, this switch between domains is not part of the scope. But with the technology used in the implementation, meaning Retrieval Augmented Generation, described in section 2.2.2. With more data and also data from other domains, this could facilitate for QAs to easily find information and therefore also facilitate switches between domains. Since the user is not bound to only prompt questions resulting in test case generation, the user could in practice ask anything including summarizing information from other domains.

In the same Table 3, we can also observe some general attitudes towards the usage of AI for generating test cases. The participants had a positive view of using AI, which also corresponds with the initial thought from the first interview. In a general matter, the common reflection is that this tool could be used for facilitating and increasing work efficiency. Even if the implementation is not itself sufficient at the current state to fully use the test cases. 66.67% of the test cases are considered useful. Important to notice is also the attitudes and specifically fears with using AI. As seen in the interview section of the results chapter 4, the interviewees have also expressed a certain degree of fear towards company information leakage. Since this specific implementation locally stores documents and embedding, the possibility of leakage is fairly slim. However, if this study would continue as a future work. The way of storing data is optimally going to change. This case would require a safer way

of storing a larger amount of data, to minimize the risk of company information leakage.

Lastly, Table 4 displays an example of a system output. The table illustrates one specific test case in else a collection of test cases that the system produces. The example output is a Happy Path, which in general is the type of test case the AI tends to find more easily, as seen in 11. The table reveals how the AI correctly identifies both the task and the necessary information for the task. As seen in the initial message before the table, the AI finds articles relevant to the query. The language used is straightforward and could be considered casual, which likely is due to the component *Tone* discussed in section 2.2.1 was never used in the Prompt Template used in the Method 3. However, the results were not always displayed with the same quality. As described in the results 4, the project information used for the study, was a of simpler sort. Increasing the complexity, or asking about test cases for projects that require significantly higher knowledge might likely change these results.

By comparing the results already discussed to my initial expectations, some of these results are unexpected. Initially, as I did not have sufficient knowledge about any Customization techniques, discussed in Section 2.2, I did not know about the critical role Prompt Engineering would have in this project. Initially, my thought was that a slightly different prompt would drastically change the quality of the output. But from reading and understanding what Transformer Architecture 2.1.2 is, and how the system is connected, the results started to be more understandable. As described in Section 2.2.2, the input query are embedded in the same way as the documents composing the knowledge base. The documents used to answer the input query is selected depending on the closest distance between the input query embedding and the document embedding. Having this in mind, the importance of prompting the input query sufficiently is much more understandable.

Another unexpected finding was the results obtained and how these results were in some way dependent on the chunk size and model size. The scaling Laws described in section 2.1.3 describe the practical explanation of why LLM tends to perform better than NLP. From the implementation using Retrieval Augmented Generation, I initially thought the effect would be more positive. However, the system seemed to be confused with the lack of data composing the knowledge base, which led to assumptions being made. In the same section 2.1.3, it is also discussed how this scaling law works to predict on task level. The discussion touches on how no direct correlation to heightened performance can necessarily be made on a task level. This would be an interesting comment for this study since AI-driven test case generation is highly task-driven.

5.1 Difficulties and solutions

During the time of the study, some difficulties were faced. When working with a language model in a big organisation, multiple limitations are set. For example, the specific Kernel used was not compatible with the recommended packages for the solution. Which leads to changes being forced to be made that could potentially have a negative effect on the results. One of the packages being affected by this incompatibility issue is `CharacterSplitter`. Because the package was not compatible with the current kernel, the package was excluded and a type of manual slit was made. The manual split however is very simplified and does not exactly behave as the package.

Another problem faced was the token-size sent into the LLM was too big when collecting the 5 best points for the Retrieval Augmented Generation. At the start of the project, it

was noted that fewer points would be considered, but because of the composition of the documents, this was changed. By using more points, meaning taking into consideration more text, this often leads to the token size exceeding its limit, forcing the API to fail. To solve this, instead of directly sending in the documents together with the input query, as discussed in section 3, the documents selected are summarized first, to avoid the vast token size. This way of solving the problem showed pleasant results, but could in the further continue to be worked on to create even better summaries. By taking into consideration the section regarding transformer architecture 2.1.2. Since the component *Outputs (shifted right)*, predicts the next word in a sequence by looking at the words before it. The summarisation of the text plays a very critical role, considering this is the text that will be sent to the LLM.

6 Conclusion

In this study, the main objective was to investigate how the integration of AI-driven test case generation in software development pipelines impact the overall software quality and development efficiency. In particular the study uses Large Language model and Embedding model provided by Llama, specifically Llama2 of size 7B, in order to generate test cases given a defined input. The implementation uses the customization technique Retrieval Augmented Generation (RAG) in order to locally story organisation information and from this stored information make conclusions resulting in test cases.

The study indicates that by using the customization techniques Retrieval Augmented Generation, in combination with Prompt engineering, the implementation is able to generate test cases in the correct format and to some degree useful. One of the results obtained, 66.67% of the test cases for a particular project where considered useful. However, since the project was of more simple character these results might be different depending on the project complexity. As seen in the results 4, the most common noise in the system is due to the system producing non accurate data. From the results we can also conclude that by using the AI system the time efficiency increased significantly. Even if all results where not useful, and some useful results created minor noise. The overall time efficiency is still provides a very pleasant impact on the development efficiency.

Despite the finding of the investigation, the study is not sufficient enough to draw strong conclusions. As the study is an *early phase exploratory study*, these results might be helpful for the future investigations on the subject. Large Language Models and the chosen customization techniques have shown great potential in customizing LLMs to understand, interpret and draw conclusions on company information. The implementation show positive effect on development efficiency and could be positive to overall software quality. Even if the implementation do not show a direct positive effect on overall software quality, the time efficiency could be argued to help the overall software quality since the implementation can be used for test cases of more simple character. Helping the QAs to focus on finding edge cases and overall gain a better software quality.

6.1 Future work

Even if the results presented in section 4, show a positive indication for using AI-driven test cases, in terms of work efficiency. This study is still very limited, both in performance and in terms of coverage. In the future, the implementation should ideally be based on much more data. This to cover all areas of the desired domain and to minimize the amount of confusion caused by assumptions. One type of data that was not used but should ideally be interesting to include is Manually written Test Cases from other projects. This was originally intended to be a part of the implementation but since previous test cases in this specific domain were limited, only other domains test cases would be available to include, which was not suited to the implementation since this study is a prototype and only covers

one specific domain. Introducing manually and well-written test cases could help the AI to solve the specific problem since it has previous data on how this has been made. At this time, the implementation makes conclusions and writes test cases with the information obtained, in combination with the information of test cases from the pre-trained data. But gathering specific ways of writing test cases in the organisation would potentially be much better. Lastly, as discussed in section 2.1.3, more data could lead to better performance, even if this is not guaranteed.

Another way of approaching the study in the future is to explore more layers from the section 2.2. In this specific implementation, the study has explored the usage of Prompt Engineering 2.2.1 and Retrieval Augmented Generation 2.2.2. But in the future, it would be interesting to see how the implementation would perform by using Fine Tuning instead. Fine tuning in comparison to Retrieval Augmented Generation does not introduce a new set of data but rather uses a dataset of labeled examples to update the weights of LLM and make the model improve its ability for specific tasks [47]. While the usage of Fine Tuning requires an extensive amount of data, this method also tends to give much better results [51]. The possibility of the system to improving by introducing this method could be interesting in future work. Another way to most likely improve the performance of the system in this study is by using a different LLM model. As described in section 3, the Large Language Model and Embedding Model are both of size 7B. This means that the model was pre-trained on 2 trillion tokens of data from publicly available sources [14]. In the documentation available on the Huggingface website, a table representing the different model's evaluation results. Figure 13 shows the results on standard academic benchmark.

Model	Size	Code	Commonsense Reasoning	World Knowledge	Reading Comprehension	Math	MMLU	BBH
Llama 1	7B	14.1	60.8	46.2	58.5	6.95	35.1	30.3
Llama 1	13B	18.9	66.1	52.6	62.3	10.9	46.9	37.0
Llama 1	33B	26.0	70.0	58.4	67.6	21.4	57.8	39.8
Llama 1	65B	30.7	70.7	60.5	68.6	30.8	63.4	43.5
Llama 2	7B	16.8	63.9	48.9	61.3	14.6	45.3	32.6
Llama 2	13B	24.5	66.9	55.4	65.8	28.7	54.8	39.4
Llama 2	70B	**37.5**	**71.9**	**63.6**	**69.4**	**35.2**	**68.9**	**51.2**

Figure 13: Evaluation Results from Llama documentation

As previously discussed, this uses the 7B size of Llama, which is the poorest performing Model version according to the evaluation results in Figure 13. Therefore exploring Llama

models of size 13B and 70B, would also potentially be an interesting future work of the study. Especially in relation to the scaling law described in section 2.1.3. Specifically, it would be interesting to investigate how much better the system could perform by using bigger models but still performing a specific task, being to create Test Cases.

Another interesting future work topic, similar to changing the LLM size, is to change the way the embedding is being made. As discussed in the text both Contextual embedding and Word embedding, among others, are ways we can in practice embed our documents 2.1.1. In this implementation the embedding model `llama-2-7b-chat-hf`, however, there are other specific algorithms for contextual respective text embedding that could be interesting to investigate to increase the performance of the system.

References

[1] Airfocus. What are Quality Assurance Engineers (QA Engineers)? Definition & FAQ — airfocus — airfocus.com. https://airfocus.com/glossary/what-are-quality-assurance-engineers/. [Accessed 04-01-2024].

[2] Amazon. What are Large Language Models? - LLM AI Explained - AWS — aws.amazon.com. https://aws.amazon.com/what-is/large-language-model/#:~:text=Large%20language%20models%20(LLM)%20are,decoder%20with%20self%2Dattention%20capabilities. [Accessed 04-01-2024].

[3] Nikoletta Bika. How to conduct a structured interview. https://resources.workable.com/tutorial/conduct-structured-interview#:~:text=A%20structured%20interview%20is%20a,likelihood%20of%20a%20bad%20hire. [Accessed 08-01-2024].

[4] Stefan Blomkvist. Persona–an overview. *Retrieved November*, 22:2004, 2002.

[5] Browserstack. How to write Test Cases (with Format & Example) — BrowserStack — browserstack.com. https://www.browserstack.com/guide/how-to-write-test-cases#:~:text=What%20is%20a%20Test%20Case,necessary%20to%20verify%20a%20feature. [Accessed 08-01-2024].

[6] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S Yu, and Lichao Sun. A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt. *arXiv preprint arXiv:2303.04226*, 2023.

[7] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *arXiv preprint arXiv:2307.03109*, 2023.

[8] Chroma. Home — Chroma — docs.trychroma.com. https://docs.trychroma.com/. [Accessed 08-01-2024].

[9] Coursera. Machine Learning vs. AI: Differences, Uses, and Benefits. https://www.coursera.org/articles/machine-learning-vs-ai#:~:text=In%20simplest%20terms%2C%20AI%20is,can%20perform%20such%20complex%20tasks. [Accessed 04-01-2024].

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Google. Duet AI for Developers — cloud.google.com. https://cloud.google.com/duet-ai. [Accessed 08-01-2024].

[12] Harvard. From Transformer to LLM: Architecture, Training and Usage — scholar.harvard.edu. https://scholar.harvard.edu/binxuw/classes/machine-learning-scratch/materials/transformers. [Accessed 08-01-2024].

[13] Christina Hsiao. From Simple to Sophisticated: 4 Levels of LLM Customization With Dataiku — blog.dataiku.com. https://blog.dataiku.com/4-levels-of-llm-customization-with-dataiku. [Accessed 04-01-2024].

[14] Hugginface. meta-llama/Llama-2-7b · Hugging Face — huggingface.co. https://huggingface.co/meta-llama/Llama-2-7b. [Accessed 07-01-2024].

[15] Purva Huilgol. Top 4 Sentence Embedding Techniques using Python — analyticsvidhya.com. https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/. [Accessed 08-01-2024].

[16] IBM. What is Deep Learning? — IBM — ibm.com. https://www.ibm.com/topics/deep-learning#:~:text=Deep%20learning%20is%20a%20subset,from%20large%20amounts%20of%20data. [Accessed 08-01-2024].

[17] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*, 2023.

[18] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[20] Mohd Ehmer Khan et al. Different approaches to white box testing technique for finding errors. *International Journal of Software Engineering and Its Applications*, 5(3):1–14, 2011.

[21] Mohd Ehmer Khan and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, 3(6), 2012.

[22] Lucy Kimbell. Rethinking design thinking: Part i. *Design and culture*, 3(3):285–306, 2011.

[23] Manish Kumar, Santosh Kumar Singh, RK Dwivedi, et al. A comparative study of black box testing and white box testing techniques. *International Journal of Advance Research in Computer Science and Management Studies*, 3(10), 2015.

[24] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966. PMLR, 2015.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[26] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014.

[27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[28] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[29] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[30] Qi Liu, Matt J Kusner, and Phil Blunsom. A survey on contextual embeddings. *arXiv preprint arXiv:2003.07278*, 2020.

[31] Llama. Chat with Llama 2 — llama2.ai. https://www.llama2.ai/. [Accessed 08-01-2024].

[32] Llamaindex. Simple Directory Reader - LlamaIndex. https://docs.llamaindex.ai/en/stable/examples/data_connectors/simple_directory_reader.html. [Accessed 08-01-2024].

[33] Lucidspark. 5 Group Brainstorming Techniques for Winning Teams — lucidspark.com. https://lucidspark.com/blog/5-group-brainstorming-techniques. [Accessed 08-01-2024].

[34] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[35] Pradeep Menon. Introduction to Large Language Models and the Transformer Architecture. https://rpradeepmenon.medium.com/introduction-to-large-language-models-and-the-transformer-architecture-534408ed7e61. [Accessed 08-01-2024].

[36] Meta. Llama 2 - Meta AI — ai.meta.com. https://ai.meta.com/llama/. [Accessed 08-01-2024].

[37] Microsoft. Azure OpenAI Service – Advanced Language Models. https://azure.microsoft.com/en-us/products/ai-services/openai-service. [Accessed 08-01-2024].

[38] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 2011.

[39] Aakanksha Patil. Top 5 Pre-trained Word Embeddings. https://patil-aakanksha.medium.com/top-5-pre-trained-word-embeddings-20de114bc26. [Accessed 04-01-2024].

[40] QA Programmer. Techniques for Estimating the Time Required for Software Testing — linkedin.com. https://www.linkedin.com/pulse/techniques-estimating-time-required-software-testing-qa-programmer/. [Accessed 08-01-2024].

[41] Promptingguide. Prompt Engineering Guide — Prompt Engineering Guide — promptingguide.ai. https://www.promptingguide.ai/. [Accessed 04-01-2024].

[42] Promptingguide. Retrieval Augmented Generation (RAG) — Prompt Engineering Guide — promptingguide.ai. https://www.promptingguide.ai/techniques/rag. [Accessed 04-01-2024].

[43] Vineet Raina, Srinath Krishnamurthy, Vineet Raina, and Srinath Krishnamurthy. Natural language processing. *Building an Effective Data Science Practice: A Framework to Bootstrap and Manage a Successful Data Science Practice*, pages 63–73, 2022.

[44] Gourav Singh. Introduction to Artificial Neural Networks — analyticsvidhya.com. https://www.analyticsvidhya.com/blog/2021/09/introduction-to-artificial-neural-networks/. [Accessed 04-01-2024].

[45] Jeff Su. Master the perfect chatgpt prompt formula (in just 8 minutes)! — youtu.be. https://youtu.be/jC4v5AS4RIM?si=eAJLgs8iL1IsW-Ov.

[46] Deepthi Sudharsan. Decoding the Relationship: Language Models and Natural Language Processing. https://medium.com/@deepthi.sudharsan/decoding-the-relationship-language-models-and-natural-language-processing-bbc0cd6754e2. [Accessed 04-01-2024].

[47] Superannotate. Fine-tuning large language models (LLMs) in 2023 — SuperAnnotate — superannotate.com. https://www.superannotate.com/blog/llm-fine-tuning#:~:text=science%20educational%20platform.-,Fine%2Dtuning%20methods,its%20ability%20for%20specific%20tasks.. [Accessed 07-01-2024].

[48] MLM Team. Prompt Engineering for Effective Interaction with ChatGPT - MachineLearningMastery.com — machinelearningmastery.com. https://machinelearningmastery.com/prompt-engineering-for-effective-interaction-with-chatgpt/. [Accessed 04-01-2024].

[49] Testsigma. Test Design in Software Testing - A Comprehensive Guide — testsigma.com. https://testsigma.com/blog/test-design/. [Accessed 04-01-2024].

[50] Solano Todeschini. How to Chunk Text Data - A Comparative Analysis — towardsdatascience.com. https://towardsdatascience.com/how-to-chunk-text-data-a-comparative-analysis-3858c4a0997a. [Accessed 08-01-2024].

[51] Hoang Tran. Which is better, retrieval augmentation (RAG) or fine-tuning? Both. — snorkel.ai. https://snorkel.ai/which-is-better-retrieval-augmentation-rag-or-fine-tuning-both/#:~:text=Fine%2Dtuning%20vs.,set%20of%20long%2Dterm%20tasks. [Accessed 07-01-2024].

[52] Treehouse. Happy Path vs. Testing Edge Cases. https://teamtreehouse.com/library/introduction-to-qa-engineering/happy-path-vs-testing-edge-cases#:~:text=Definitions%3A,or%20interact%20with%20the%20application. [Accessed 08-01-2024].

[53] Princeton University. cs.princeton.edu. https://www.cs.princeton.edu/courses/archive/spring20/cos598C/lectures/lec3-contextualized-word-embeddings.pdf. [Accessed 04-01-2024].

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[55] Akanksha Verma, Amita Khatana, and Sarika Chaudhary. A comparative study of black box testing and white box testing. *International Journal of Computer Sciences and Engineering*, 5(12):301–304, 2017.

[56] Amol Wagh. What's Generative AI? Explore Underlying Layers of Machine Learning and Deep Learning. https://s.com/@amol-wagh/whats-generative-ai-explore-underlying-layers-of-machine-learning-and-deep-learning-8f99272e0b0d. [Accessed 08-01-2024].

[57] Wallarm. What is White Box Testing? Types, Techniques, Examples — wallarm.com. https://www.wallarm.com/what/white-box-testing. [Accessed 08-01-2024].

[58] Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C-C Jay Kuo. Evaluating word embedding models: Methods and experimental results. *APSIPA transactions on signal and information processing*, 8:e19, 2019.

[59] NSW Goverment website. Phases of design thinking — education.nsw.gov.au. https://education.nsw.gov.au/teaching-and-learning/curriculum/stem/early-stage-1-to-stage-3/project-based-learning-and-design-thinking/phases-of-design-thinking. [Accessed 08-01-2024].

[60] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[61] Susan C Weller. Structured interviewing and questionnaire construction. *Handbook of methods in cultural anthropology*, pages 365–409, 1998.

[62] Janet BW Williams. A structured interview guide for the hamilton depression rating scale. *Archives of general psychiatry*, 45(8):742–747, 1988.

[63] Jia-Yu Yao, Kun-Peng Ning, Zhen-Hui Liu, Mu-Nan Ning, and Li Yuan. Llm lies: Hallucinations are not bugs, but features as adversarial examples. *arXiv preprint arXiv:2310.01469*, 2023.

[64] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

A Interview Questions

Script start
Welcome to this short interview, let's start by saying thank you for taking your time. The interview will take approximately 20 minutes. The answers from the interview will be used as a part of a research project. None of the answers will be connected to you as a person and will only be used in the context of this study. To participate in the study is voluntary and you can decide to stop the interview at any time. Before starting the interview, do you agree with these conditions?
Script end

Open questions for user understanding
Q1: What is your role at PayPal?
Q2: How many years of experience do you have in that role and in the company?

Questions with Testing Focus
Q3: One part of the QA role is to create Test Cases. How often would you estimate you create test cases per quarter when being of full capacity?
Q4: If we would now focus on the process of creating Test Cases, how does this process look like for you? Please explain the process from start to finish
Q5: During the process of creating Test Cases do you tend to create these test cases alone or in collaboration with another employee? If in collaboration, what roles do these employees have?
Q6: What would you describe as your biggest challenge when creating Test Cases?
Q7: Is there any specific part of the process that you would describe as repetitive or annoying?

Questions with AI
Q8: Have you ever integrated with a Large Language Model such as ChatGTP?
Q9: Have you ever used such tool to solve a work related task?
Q10: How often do you find yourself searching this kind of help? Context?
Q11: Do you know which LLMs have been cleared to access through PayPal and how you can access these?

Questions with AI
Here are some question with a practical exercise:
In front of you, you will have ChatGTP. I will give you some instructions and I want you to use ChatGTP to get an appropriate answer to what I am asking for. The answer from ChatGTP should be as specific and as useful as possible, meaning you should aim to get an answer that does not need to be rewritten.

Q12: 3 recipes for a blog
Q13: An analysis of the book A Little Life
Q14: Test case for a simple login function at Facebook

Questions with AI and security/fear
Q15: What are your initial thought regarding the usage of AI in the context of Software testing? Do you see any ethical issues that can potentially be faced?