UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Grandes modelos del lenguaje para la ayuda automática al cumplimiento de reglamentos y normativas: Aplicación de la tecnología a la regulación ISO14001

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Huertas Pastor, Jorge

Tutor/a: Colomer Granero, Adrián

CURSO ACADÉMICO: 2023/2024

NO COVER PAGE

Resumen

Últimamente he estado trabajando como becario en MAHLE Electronics SLU. Allí descubrí lo ineficiente y tediosa que es la gestión de la información técnica, y no se limita a esta única empresa.

Para resolver este problema, deberíamos encontrar una forma más eficiente de almacenar información o pasar horas memorizando archivos PDF para no tener la necesidad de buscarlos y, cuando se cambie una determinada norma, hacer todo el proceso otra vez. Hay demasiadas regulaciones, con demasiadas variables, se actualizan constantemente cada año y no existe un sistema para organizar toda esta información más allá del anticuado explorador de archivos. Por eso vamos a proponer una solución alternativa.

Esta solución se basa en la idea de un asistente de inteligencia artificial personalizado que extrae información de una base de datos predeterminada. La razón por la que esto es necesario es que los motores de búsqueda habituales están demasiado repletos de información y no son muy específicos cuando se trata de información técnica. Además, no pueden acceder a archivos privados o confidenciales que contengan los datos necesarios.

El objetivo es crear un asistente de inteligencia artificial que utilice algoritmos de "knowledge retrieval" para extraer y sintetizar datos de una colección de archivos PDF. La forma de interactuar con este asistente será a través de un chatbot de Telegram. Por lo tanto, necesitaremos crear el asistente con la información y luego vincularlo a un chatbot de Telegram para que podamos comunicarnos con él.

Para construir el asistente usaremos el código Python proporcionado por OpenAI y su API asistente, luego lo vincularemos al chatbot de Telegram a través del ID del bot. Todo esto se hará en código Python.

Finalmente, evaluaremos el rendimiento de este bot en la búsqueda y síntesis de información respecto a un trabajador de MAHLE Electronics SLU. Este TFG evalúa el desempeño del bot, su viabilidad y su potencial. Los resultados muestran cómo un espacio de trabajo podría beneficiarse con la inclusión de este bot.

Palabras Clave: ChatGPT, Telegram Bot, Inteligencia artificial, Procesamiento del Language Natural, Comunicación automatitzada.

Resum

Últimament he estat treballant com a becari a MAHLE Electronics SLU. Allà vaig descobrir que ineficient i tediosa és la gestió de la informació tècnica, i no es limita a aquesta única empresa.

Per resoldre aquest problema, hauríem de trobar una forma més eficient d'emmagatzemar informació o passar hores memoritzant fitxers PDF per no tenir la necessitat de buscar-los i, quan es canviï una determinada norma, fer tot el procés una altra vegada. Hi ha massa regulacions, amb massa variables, s'actualitzen constantment cada any i no hi ha un sistema per organitzar tota aquesta informació més enllà de l'antiquat explorador d'arxius. Per això proposarem una solució alternativa.

Aquesta solució es basa en la idea d'un assistent d'intel·ligència artificial personalitzat que extreu informació sobre una base de dades predeterminada. La raó per la qual això és necessari és que els motors de cerca habituals estan massa plens d'informació i no són molt específics quan es tracta d'informació tècnica. A més, no poden accedir a fitxers privats o confidencials que continguin les dades necessàries.

L'objectiu és crear un assistent d'intel·ligència artificial que utilitzi algorismes de "knowledge retrieval" per extreure i sintetitzar dades d'una col·lecció de fitxers PDF. La manera d'interactuar amb aquest assistent serà mitjançant un chatbot de Telegram. Per tant, necessitarem crear l'assistent amb la informació i després vincular-ho a un chatbot de Telegram perquè ens puguem comunicar amb ell.

Per construir l'assistent farem servir el codi Python proporcionat per OpenAI i el seu API assistent, després el vincularem al chatbot de Telegram a través de l'ID del bot. Tot això es farà en codi Python.

Finalment, avaluarem el rendiment d'aquest bot en la cerca i la síntesi d'informació respecte a un treballador de MAHLE Electronics SLU. Aquest TFG avalua l'exercici del bot, la seva viabilitat i el seu potencial. Els resultats mostren com un espai de treball es podria beneficiar amb la inclusió d'aquest bot.

Paraules clau: ChatGPT, Telegram Bot, Intel·ligència artificial, Processament del llenguatge natural, Comunicació automatitzada.

Abstract

Lately, I have been working as an intern in MAHLE Electronics SLU. There I discovered how inefficient and tiresome the management of technical information is, and it's not limited to this one company.

To solve this issue, we should come up with a more efficient way of storing information ore spend hours memorizing PDFs so that we don't have the need to search for them and, when a certain regulation is changed, do it all again. There are too many regulations, with too many variables, they are constantly getting updated every year and there is not a system in place to organize all this information beyond the antiquated file explorer. That is why we are going to propose an alternative solution.

To do so, we came with the idea of a personalized AI assistant that draws information from a pre-determined database. The reason this is needed is that regular search engines are too bloated with information and are not very specific when it comes to technical information. Also, they cannot access private or confidential files that contain the needed data.

The objective is to build an AI assistant that uses knowledge retrieval algorithms to extract and synthesize data from a collection of PDFs files. The way to interact with this assistant will be through a Telegram chatbot. So, we will need to build the assistant with the information, then link the assistant to a telegram chatbot so that we can communicate with it.

To build the assistant we will use the python code provided by OpenAI and its assistant API, then we will link it to the telegram chatbot through the bot's ID. All of this will be done in python code.

Finally, we will evaluate the performance of this bot at finding and synthesizing information compared to a worker in MAHLE Electronics SLU. This bachelor's degree thesis evaluates the performance of the bot, its feasibility, and its potential. The results show how a general workspace could benefit by the inclusion of this bot.

Key Words: ChatGPT, Telegram Bot, Artificial Intelligence, Natural Language Processing, Automatized Communication.

EXECUTIVE SUMMARY

The degree thesis must develop in the text the following concepts, appropriately justified and discussed, focusing on the FIELD OF STUDY

CONCEPT (ABET)	**Done? (Y/N)**	**Where? (page numbers)**
1. IDENTIFY:		
1.1. Problem statement and opportunity		
1.2. Constraints (standards, codes, needs, requirements & specifications)		
1.3. Setting of goals		
2. FORMULATE:		
2.1. Creative solution generation (analysis)		
2.2. Evaluation of multiple solutions and decision-making (synthesis)		
3. SOLVE:		
3.1. Fulfilment of goals		
3.2. Overall impact and significance (contributions and practical recommendations)		

Table of contents

Chapter 1. Introduction 1
1.1 Motivation 1
1.2 State of the Art 2
1.2.1 Combining Virtual and Robot Assistants 2
1.2.2 ChatPDF 3
1.2.3 Incorporating Natural Language Processing into Virtual Assistants 5
1.2.4 Assistant API 6
1.3 Outline 7
Chapter 2. Objectives 8
Chapter 3. Background 9
3.1 Computer Programming Language 9
3.2 Expert Systems 10
3.3 Natural Language Processing 11
3.4 GPT-4 12
3.5 Information Retrieval 14
Chapter 4. Methodology 16
4.1 Telegram API 16
4.2 Make.com 17
4.3 Python ChatGPT Implementation 19
4.3.1 Main (the whole code can be found in appendix C) 19
4.3.2 Utilities 21
4.3.3 OpenAI Helper 21
4.3.4 Telegram Bot 22
4.3.5 Evaluation 22
4.4 Building an assistant 23
Chapter 5. PDF Insight 27
5.1 Instructions 27
5.2 Telegram Integration (the whole code is present in appendix D) 28
5.3 Creating the assistant 29

5.4 Results 30
Chapter 6. Specifications 33
6.1 Pricing 33
6.2 Software 33
Chapter 7. Conclusion 34
Chapter 8. References 36
Chapter 9. Appendix A 38
Chapter 10. Appendix B 45
Chapter 11. Appendix C 46
Chapter 12. Appendix D 50
Chapter 13. Appendix E 52

Chapter 1. Introduction

1.1 Motivation

Have you ever noticed how badly Google Search works nowadays?[1]It is something that happens a lot, as soon as you search for something mildly technical you find yourself with plenty of useless information that does not really answer what you are looking for. It is a very frustrating feeling; it seems like there is an overload of nonspecific information and you just end up wasting time.

Now picture the next scenario, you are a laboratory technician a company has recently hired. You must do an experiment following an extremely specific set of rules. Every person that could help you is busy, since it is a big company. However, you know that the specific rules you must follow are in a PDF file. This file is in the folder of "Daimler", which is in the folder of "Client Rules", which is in the folder of "Rules", which is in the folder of "Data". It has a very nonspecific name, so you cannot search for it using the file explorer, even if you could, there are so many other folders in "Data" that it would take up to 5 minutes to complete the search. What do you do?

Well, usually, you scout every folder, find the PDF, build the experiment, and then someone who has done this experiment before comes by and tells you that you have done it wrong, since you have chosen the 2023 file, and you should have used the 2018 file. In the company I am working, this happens constantly. There is too much information, and the people that could guide you through it are often busy with something else.

It also can happen that after finding the correct file, you just cannot find the information, or you misread it or even that you misremember it from the previous time you had to search for it. There are many steps of this process that can go wrong, both from human error and from the lack of power of the current file explorer.

Something that most of the world agrees on is that time is money, and the current system not only wastes a lot of time, but it is also prone to error. So, how do we fix it?

The ideal would be a person that knows all the rules and requirements stored in the PDF files that you can ask for guidance. However, we humans have limited capabilities of information storages, we can remember things wrong, and we are usually busy with other work. So, we need to create a system that can store all the information, answers correctly and precisely and is always available.

If we remove the "always answers correctly" part, we think ChatGPT fits into this definition. However, ChatGPT comes with its own set of problems. Mainly, you cannot feed it information. It knows a lot about everything, but it lacks specific knowledge which most often than not, like Google Search it just makes you waste your time trying it to answer you correctly.

In summary, we need a ChatGPT that only draws information from a specific database, that is always available and that always answers correctly. In other words, we need an AI assistant.

1.2 State of the Art

The first modern digital virtual assistant installed on a smartphone was Siri, which was introduced as a feature of the iPhone 4S on 4 October 2011. Siri provided few advantages compared with the Google Search bar, like a more human like interaction and voice commands. The thing that impacted the most of Siri weren't the capabilities, it was the novelty of having a machine interact and talk like a human. But if we could apply the principles of virtual assistants to more specific needs?

1.2.1 Combining Virtual and Robot Assistants

The following article [2] provides a comprehensive case study of integrating virtual assistants, particularly Amazon's Alexa, into the realm of industrial robotics. The authors discovered an issue with human-machine communication when integrating intelligent robotic assistant systems that cooperate with humans in non-destructive disassembly processes. They attempted to fix it with a more tactile interface beyond the usual mouse and keyboard that this caused logistic issues where the tactile panel was out of reach. Due to this they decided for an alternative way of communication, natural language.

The application they developed uses Amazon Alexa embedded in the Amazon Echo Dot device. The Echo Dot requires an active internet connection to connect to the Alexa Voice Service (AVS) running on the Amazon Web Services (AWS). Within the AWS ecosystem, they further connect the Amazon Voice Service with the Amazon Lambda service (ALS) and the ALS with Amazon's Internet of Things (IoT) service. Furthermore, a microcomputer (Raspberry Pi) is used to route messages from the IoT cloud to a local network. Through the local network, the single devices of the intelligent robotic assistance system can be accessed to control or update the devices' state in the cloud.

The final product is a collaborative robot that can respond to preprogrammed voice commands which has increased in feasibility with nowadays safer robots. It has disadvantages such as being susceptible to noise, needing a stable internet connection and Alexa misinterpreting when a voice line ends or starts. However, despite these flaws it has proven to be a useful asset in the workplace.

This project focuses on an assistant to help the human-machine communication, but what about an assistant to help the human-PDF communication, is there something like that?

1.2.2 ChatPDF

ChatPDF provides a way of users to interact with their PDF files. It creates a semantic index over all paragraphs of the PDF. When answering a question, ChatPDF finds the most relevant paragraphs from the PDF and uses the ChatGPT API from OpenAI to generate an answer [3]. This is precisely what we had in mind when talking about human-PDF communication. The article *Enhancing PDF interaction for a more engaging user experience in library: Introducing ChatPDF* [4] makes a great point about the interactivity of the PDF file being quite lacking. It also makes a point to address the inefficient navigation in longer documents in which the user might have trouble finding the information they are searching. The article states ChatPDF is a soluction to these problems, so let's test it to see how well it fares with more technical information.

Before getting to the testing, we should delve into the interface and workings of the website, to have a complete picture of the tool we are testing. First of all, we must decide the version we are going to try. Chat comes with both a free and a plus version, the plus version having a billing of 111.9€/year or 15.99€/month.

The plus version comes with the following advantages:

- Unlimited PDFs
- Unlimited Questions
- 2000 pages/32 MB size PDFs
- 50 PDFs per folder

Compared to the free version:

- 2 PDFs
- 20 Questions
- 120 pages each
- No information on size or PDFs per folder

For the purpose of this bachelor's degree thesis, we will focus on the free version and feed it both the "ISO 10605 [2023] (ESD).pdf" and the "B21 7112-Ed.OR (EMC for HV parts).pdf" to evaluate its capabilities. Uploading the PDF files is as easy as dropping them with the mouse or exploring your computer for the file, it also can read PDF files if the URL is provided. Once everything is set up, we can explore the chatting interface.

The first thing to note in this interface is that it gives you the ability to interact with each PDF file individually or drop both in a folder to chat with them together, you have also the additional options of rename chat, export chat, reset chat or delete chat/folder. The option of export chat is quite interesting since it allows for easy storage of the obtained information. This option will be used to provide the examples of conversations present in appendix A.

Interface aside, we need to talk about how good ChatPDF performance is. Its main advantage is speed. ChatPDF can process a PDF in less than five seconds and every single answer it gave was preceded by at most a three second delay. The next thing we should measure is its accuracy. In **Figure 1** we can find a pie chart where all the answers ChatPDF gave are categorized. First let's start with the test questions "What is god according to Christianity?" and "what is Natural Language Processing?".

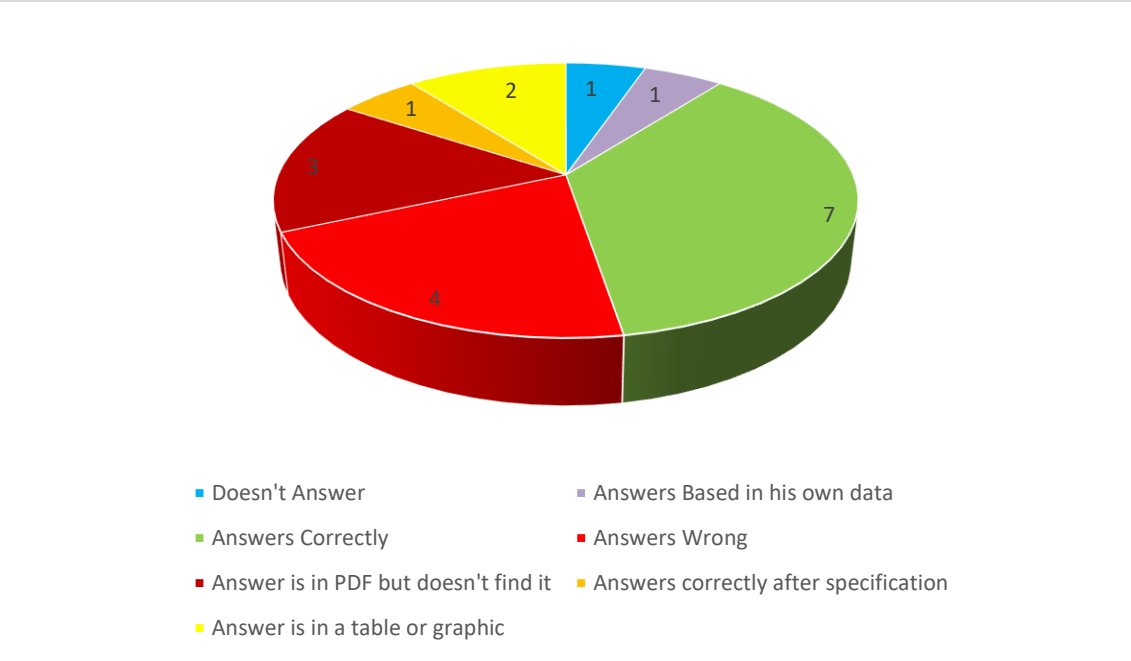


Figure 1. Accuracy of ChatPDF

These two questions are built to test if the application is searching for its information correctly, neither of them are specified in the PDF files we fed to it, so we expect the application to recognize this and don't answer. Instead, it refused to answer the "What is god according to Christianity?" question as it stated that only answers with information that is considered technical and factual. He answered the second question "what is Natural Language Processing?" correctly. This proves that the application can get information from its own knowledge outside of the PDF files which makes it prone to error, specially considering it might be searching in an outdated database. This is a meaningful risk considering how often EMC Validation rules are updated.

Once we have determined if the application can answer with information outside the PDF file, we should focus on determining how accurate can it be when answering questions which answer is contained within the PDF file of the fifteen questions asked about relevant information within the PDF file, ChatPDF answered seven correctly, four wrong and stated it couldn't find the answer to three of them. When specified the location of the answer it was able to find it. In general percentages this gives an approximate 50% success rate which is well below an acceptable margin, before diving into out conclusions, we should first talk about the last question.

The website specifies ChatPDF cannot read and understand images. However, I wanted to try if it could locate the data is present inside a table. Even if it cannot read the table, it should be able to see that the PDF file information is referencing it to a table or figure. This small experiment was a failure since ChatPDF could not identify that the required information was contained inside a table.

Another thing to note is that ChatPDF often links you the page when the obtained information is at, but sometimes it will not do it. It worked for the first PDF file, but not for the second nor the folder.

In conclusion, its success ratio is 8:7, but it heavily depends on how specific you are with your prompt. Specifying the segment the data is on will yield better results in the long run. however, I think it defeats the purpose of the tool since you mainly want it to optimize searching. ChatPDF works at its best when handling less technical information. Like long text form essays or even books in a PDF format. The fact that it makes information by itself is a known issue, but it's a decent risk considering its fast responses and upload time.

The main reason we are talking about ChatPDF is this will set the goal post of what we are trying to achieve with this bachelor's degree thesis. However, instead of focusing on the public, we want to implement an assistant that can help workers on a corporative setting like MAHLE Electronics SLU.

1.2.3 Incorporating Natural Language Processing into Virtual Assistants

This study [5] presents an innovative method aimed at enhancing the Natural Language Processing (NLP) capabilities of virtual assistant systems, addressing the dual challenges of efficient information transfer and model size optimization without compromising performance. At the heart of this method is the use of the SentencePiece tokenizer in unigram settings, which is crucial for creating a well-balanced vocabulary that ensures an optimal balance between task performance and resource efficiency. Drawing inspiration from successful models like BERT and RoBERTa, a novel pre-layernorm design is introduced, optimizing the placement of layer normalization within transformer layers during the pretraining phase. This approach involves training teacher models using masked language modeling objectives and the Deepspeed scaling framework, alongside exploring modifications to model operations and mixed precision training strategies to ensure stability. The method employs a two-stage distillation process that effectively transfers knowledge from the comprehensive teacher models to more compact student models, beginning with an intermediate model and carefully distilling the data using logit and hidden layer matching techniques. This transfer of information significantly enhances the final student model while maintaining an ideal size for applications requiring low latency. The approach incorporates innovative measurements, such as the precision of mask filling, to assess the effectiveness and quality of the methods, with the findings demonstrating substantial improvements over publicly available models. This underscores the strategy's effectiveness within complete virtual assistant systems, confirming the potential of the technique to enhance language comprehension and efficiency, particularly in addressing the challenges posed by real-world user inputs. Through extensive testing and rigorous analysis, the study validates the method's ability to meet these objectives, showcasing a significant advancement in the field of NLP for virtual assistants.

Integrating Natural Language Processing (NLP) into virtual assistants is a key area of research due to its potential to greatly enhance functionality. However, this technology faces several challenges and limitations. One major issue is the inherent ambiguity of natural language and its context-dependence, which can lead to misunderstandings of user requests. Additionally, supporting multiple languages efficiently is crucial since performance can vary significantly across languages, impacting user experience. Real-

time processing is essential for seamless interaction, yet NLP's computational demands can cause delays, reducing effectiveness. Many virtual assistants also struggle to generalize from user interactions, relying too much on preset responses and struggling to adapt to unique user needs.

Limitations include a lack of common-sense reasoning, which hampers the ability to engage in truly natural conversations and understand everyday contexts. Long conversations can lead to a loss of context, resulting in irrelevant responses. There's also a risk of spreading misinformation, especially in sensitive fields like healthcare, if virtual assistants cannot verify the information they provide. Furthermore, NLP models may lack depth in certain knowledge areas, leading to inaccurate or incomplete responses in specialized contexts. The high cost and resource requirements for developing and maintaining NLP-driven assistants pose significant barriers to access, especially for smaller entities and underserved communities. Overcoming these challenges requires ongoing research to improve models' contextual understanding, multilingual support, real-time processing, and adaptability, along with strong privacy measures and cost-effective implementation strategies.

But what if we had an assistant that we could update just by uploading a new PDF file?

1.2.4 Assistant API

The Assistant API from OpenAI is the main reason this project is feasible. The Assistants API allows you to build AI assistants within your own applications. An Assistant has instructions and can leverage models, tools, and knowledge to respond to user queries. The Assistants API currently supports three types of tools: Code Interpreter, Retrieval, and Function calling. [6]

The Assistant API, in a general context, refers to a set of programming interfaces designed to enable developers to create applications that can interact with users through natural language processing (NLP) and machine learning algorithms. These APIs are the backbone of virtual assistants, enabling them to understand, process, and respond to user queries in a conversational manner. The functionality provided by an Assistant API typically includes voice recognition, text-to-speech conversion, language understanding, and the ability to integrate with various web services and databases to retrieve information or perform actions based on user requests.

At the core of an Assistant API is its NLP capabilities, which allow the assistant to parse and understand the intent behind a user's text or voice input. This involves complex processes such as tokenization, part-of-speech tagging, named entity recognition, and dependency parsing, which collectively help the assistant grasp the nuances of human language. Once the intent is understood, the assistant uses predefined or dynamically generated responses to engage with the user, often accessing external APIs to fetch data or perform tasks like setting reminders, playing music, providing weather updates, or answering questions based on a wide range of knowledge.

Assistant APIs are built on advanced machine learning models that are trained on vast datasets of human interactions, enabling them to improve over time through continuous learning and adaptation to user preferences and speech patterns. This makes them increasingly efficient and personalized in their interactions. Developers can leverage

these APIs to integrate conversational AI capabilities into various applications, ranging from mobile apps and websites to smart home devices and customer service bots, enhancing user experience and engagement through natural and intuitive interfaces.

Using this API, we can build an assistant that serves a similar function to ChatPDF, but we can program it with certain tools so it's more efficient at working with technical information.

1.3 Outline

In this bachelor's degree thesis, we will attempt to build an assistant that can help the EMC Validation Lab Technicians of MAHLE Electronics SLU. With the help of the OpenAI assistant API we can create a personalized assistant that draws information from its database and helps us solve the doubts that could appear.

It attempts to give the workers an easier way to process, find and learn information, beyond the classic scrolling. Its focus is not to replace anyone, just help save time and better comprehend the reasons behind the constraints of policies such as the IEC CISPR 25 for Electro Magnetic Validation.

The structure of this bachelor's degree thesis is organized as follows:

- Chapter 2: We explain the objectives and goals of this bachelor's degree thesis.
- Chapter 3: We provide the background necessary to understand the progression of this bachelor's degree thesis.
- Chapter 4: We delve into the detailed theory developed throughout this work. We explain the different stages of development and we introduce the formulation and variables employed to address these challenges.
- Chapter 5: We discuss the actual implementation reached. Seeing its capabilities and experimenting with it.
- Chapter 6: We delve into the material aspects, such as specifications, budget, plans, hardware, etc.
- Chapter 7: We present conclusions, open questions, and theorize some testing we could do with more resources.

Chapter 2. Objectives

The main objective of this bachelor's degree thesis is to use the OpenAI assistant API to create a virtual assistant based on the GPT-4 model that can store PDF files in its database and answer questions related to that specific information. To communicate with this assistant, we will use a Telegram Chatbot.

This assistant is meant to be implemented in the MAHLE Electronics SLU EMC Validation department where it will assist engineers and technicians in building complex setups according to either global or client specifications and requirements.

The secondary objectives are:

- Provide the background necessary for understanding what we are trying to achieve. Delving into Computer Programming, Expert Systems and Natural Language Processing.
- Make the assistant versatile enough so it can be easily updated, even implemented in another workplace.
- Explain the series of step that have led to the creation of this assistant.
- Evaluate the assistant functionality comparing it mainly to ChatPDF one of its closest relatives.
- Provide some examples of the ideal testing we could do with more resources.
- Discuss other alternatives to the assistant implementation.

Chapter 3. Background

Recently, artificial intelligence is becoming more available for both the public and the corporate market. According to IBM, 35% of businesses have already embraced AI [7] and it is estimated that the AI powered Netflix Recommendation Engine, saved Netflix over one billion dollars in 2016[8].

Artificial intelligence is a specialty within computer science that is concerned with creating systems that can replicate human intelligence and problem-solving abilities [9]. The groundwork for this technology started in the early 1900s and it has made major strides in recent years with ChatGPT, DALL-E and most recently SORA which allows you to create video from text [10]. We mainly will focus on the processing and generating of text, since those are the key foundational concepts and theories that set the stage for this bachelor's degree thesis.

3.1 Computer Programming Language

A computer programming language serves as a medium of communication between humans and machines, enabling the former to instruct the latter to perform specific tasks or solve complex problems. These languages are constructed around a syntax and a set of semantic rules that define the structure and meaning of various code constructs. The diversity in programming languages arises from their design goals, which cater to different domains ranging from web development and data analysis to system programming and scientific computing.

Going more in depth, in **figure 2** we can see a diagram describing the different levels of computer programming language. High-level languages, such as Python, Java, and C++, abstract away the complexities of the hardware, allowing developers to focus on solving domain-specific problems using constructs that are closer to human language.

Assembly language and machine language however, interface directly with a computer's hardware. Machine language, the most basic level of programming, consists of binary code (a series of 0s and 1s) that represents the most fundamental instructions understood by the computer's central processing unit (CPU). It is highly specific to the architecture of the CPU, making it difficult for humans to read or write. Assembly language, one step higher than machine language, provides a more readable format using mnemonics (symbolic representations of machine language instructions) and allows programmers to write code that corresponds closely to the hardware operations.

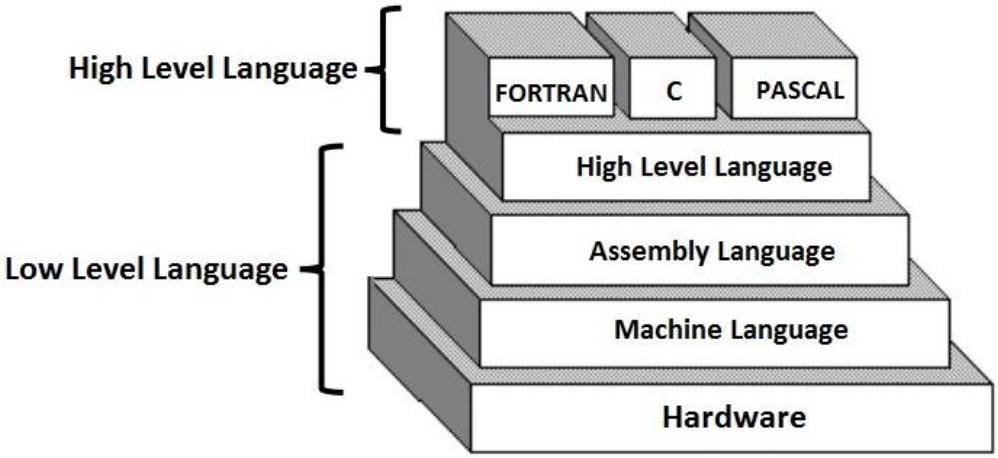


Figure 2. Computer Language and its types

3.2 Expert Systems

Expert systems leverage artificial intelligence to mimic human expertise in specialized domains, using a detailed knowledge base and an inference engine (as it can be seen in **figure 3**). XCON, developed for Digital Equipment Corporation (DEC), automated the complex task of configuring VAX computer systems by encoding the expert knowledge of DEC's specialists into a rule-based system [11]. The inference engine applied these rules through backward chaining, starting with the desired configuration outcome and working backward to determine the necessary steps. This architecture allowed XCON to handle complex configuration tasks efficiently, demonstrating the practical application of expert systems in automating decision-making processes that traditionally required specialized human knowledge. However, maintaining the system's extensive rule base posed challenges, highlighting the need for ongoing updates as domain knowledge evolved.

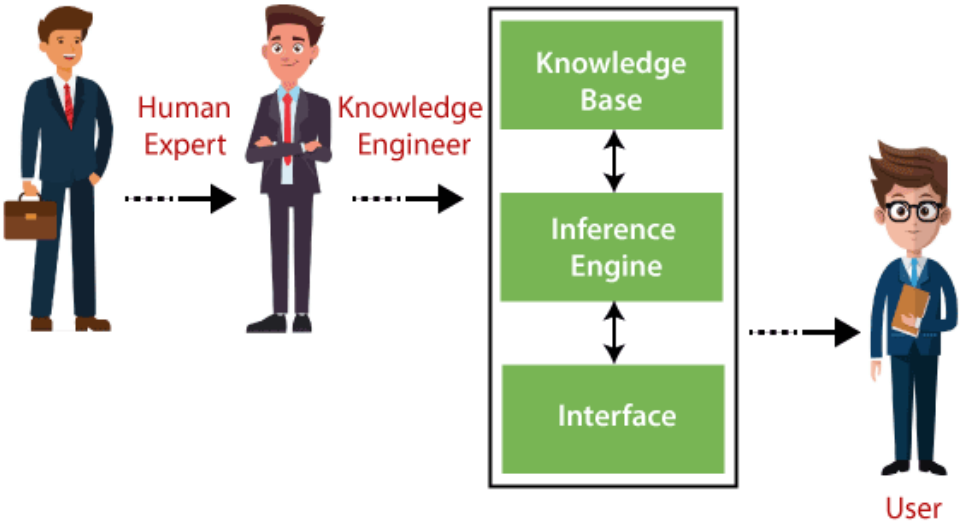


Figure 3. Expert System

Despite this, expert systems are nowadays widely used in the medical field [12], specifically for diagnosis purposes. The previous article talks about the fuzzy-based expert system, an advanced artificial intelligence system that uses unconventional thinking to reduce the uncertainty that is often associated with the diagnosis process of diseases. In the field of medical diagnostics, where symptoms can overlap, and patient reports are often uncertain, fuzzy logic provides a significant advantage to medical expert systems in identifying diseases. This form of logic extends beyond the binary framework of traditional decision-making by accommodating the variability present in medical information, reflecting the complex decision-making process of healthcare professionals. Fuzzy logic recognizes the variability in symptom presentation, enabling expert systems to process a broad range of patient data with high accuracy. This method improves the dependability of diagnoses and supports the development of treatment plans that are more closely aligned with each patient's specific needs. Additionally, the flexible nature of fuzzy logic allows these systems to stay up-to-date with the latest medical findings, ensuring their continuous improvement and relevance. Through these capabilities, fuzzy logic plays a crucial role in enhancing the performance of medical expert systems, contributing to more accurate and individualized patient care.

Although we wont use fuzzy logic in our assistant, expert systems are a great example of how AI can improve performance in the workplace. Picture an assistant that can be uploaded with a set of predetermined rules and teaches you how to correctly set up your EMC Validation test or any other kind of test. It shows how the world is adopting AI to cover the shortcoming of regular people.

3.3 Natural Language Processing

Natural Language Processing (NLP) is a pivotal technology in the field of artificial intelligence that enables machines to understand, interpret, and generate human language [13]. At its core, NLP combines computational linguistics (rule-based modeling of human language) with statistical, machine learning, and deep learning models. These technologies allow computers to process human language in the form of text or voice data and to understand its full meaning, complete with the speaker or writer's intentions and sentiments.

The technical foundation of NLP involves several key tasks and processes:

1. **Tokenization**: This is the process of breaking down text into its basic units, called tokens, which can be words, phrases, or symbols. This step is crucial for understanding the structure of the text and preparing it for further analysis.
2. **Part-of-Speech Tagging**: Once the text is tokenized, each token is tagged with its part of speech (noun, verb, adjective, etc.), which helps in understanding the grammatical structure of sentences and the roles of individual words.
3. **Parsing**: This process involves analyzing the grammatical structure of a sentence to understand the relationships between tokens. It helps in identifying the sentence's subject, predicate, and objects, thereby contributing to the comprehension of the sentence's meaning.
4. **Named Entity Recognition (NER)**: NER involves identifying and classifying key elements in text into predefined categories, such as the names of people, organizations, locations, dates, and times. It is essential for extracting useful information from text.

5. **Sentiment Analysis**: This involves analyzing text to determine the sentiment expressed in it, such as positive, negative, or neutral. This is particularly useful in understanding opinions, reviews, and social media commentary.
6. **Machine Translation**: One of the most complex NLP tasks, machine translation involves automatically translating text from one language to another, considering the grammatical structure and meaning of the original text.
7. **Semantic Analysis**: This process goes beyond the literal meaning of words to understand the nuances and intended meaning in a given context. It involves word sense disambiguation (determining the meaning of a word based on context) and understanding relationships between words and phrases in a sentence.
8. **Question Answering**: This involves building systems that can answer questions posed by humans in a natural language. It requires the system to understand the question, retrieve relevant information, and present it in a coherent answer.

Modern NLP systems leverage deep learning models, particularly those based on transformer architecture, like BERT (Bidirectional Encoder Representations from Transformers)[14] and GPT (Generative Pretrained Transformer)[15], to perform these tasks with a high degree of accuracy. These models are trained on vast amounts of text data and can capture complex patterns in language, enabling more nuanced understanding and generation of text. So, how does the latest of these models look like?

3.4 GPT-4

GPT-4, an advancement in the Generative Pre-trained Transformer series, is built on the transformer architecture (**figure 4**), which significantly improves machine understanding and generation of human language through technical sophistication. Central to this architecture is the multi-head attention mechanism, dependent on the scaled Dot-Product attention mechanism and designed to analyze and interpret complex relationships within textual data more effectively than previous models limited by recurrent or convolutional structures.

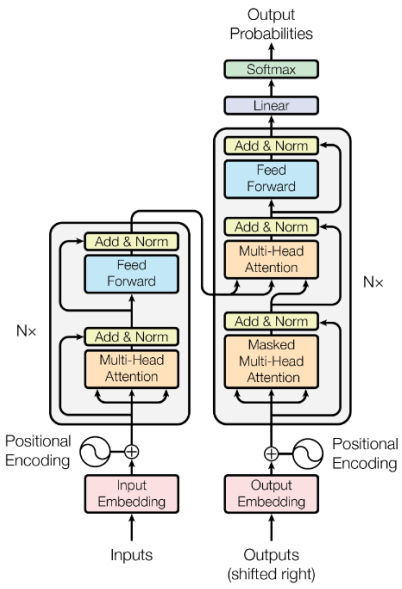


Figure 4. The Transformer – model architecture.

Within the scaled Dot-Product attention mechanism, each input element is initially encoded into three distinct vectors: the query (Q), the key (K), and the value (V). These vectors are generated through tailored linear transformations, setting the stage for the subsequent computation of attention scores.

The crux of the scaled-dot product attention involves calculating the dot product between the query vector of one element and the key vectors of all other elements in the sequence. This operation yields raw attention scores that signify the relevance of each element to the others. However, to prevent these scores from escalating with the increase in dimensionality, which could destabilize the model, a scaling factor is applied[16].

The scaling is mathematically represented as:

$$Scaled\ Score = \frac{QK^T}{\sqrt{d_k}} \quad (1)$$

Here, $\mathbf{d_k}$ is the dimensionality of the key vectors, and this division ensures the scores remain manageable, preserving computational stability.

Following the scaling, the softmax function is employed to normalize these scores into a probability distribution, emphasizing the relative importance of each element's value vector in the context of the sequence. This distribution is calculated using the formula:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2)$$

This step is critical as it highlights the contextual relevance of each word, contributing to a composite output that encapsulates the semantic and syntactic nuances of the input.

Building on this foundational mechanism, the multi-head attention framework augments the model's capacity to interpret data by parallelizing the attention process across multiple "heads." Each head operates independently, employing unique sets of linear transformations for Q, K, and V, thereby enabling the model to concurrently analyze different aspects of the input sequence. The outputs from these heads are then unified through concatenation and subjected to a final linear transformation to synthesize the comprehensive output of the multi-head attention module. This process is encapsulated in the equation:

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \quad (3)$$

$$head_i = Attention\left(QW^Q{}_i, KW_i{}^K, VW_i{}^W\right) \quad (4)$$

Where $head_i$ represents an individual attention process, and W^O is the weight matrix for the concluding transformation.

This intricate melding of scaled-dot product attention with the multi-dimensional perspective offered by multi-head attention equips transformer-based models like GPT-4 with an exceptional ability to dissect and assimilate complex sequences. Through these mechanisms, the models adeptly capture the diverse linguistic structures and dependencies present in natural language, powering their sophisticated generative and analytical functionalities.

3.5 Information Retrieval

Information Retrieval (IR) is a pivotal field within computer science that focuses on the organization, storage, retrieval, and management of information. It encompasses the processes and systems designed to collect, parse, and store data, and to retrieve relevant information in response to a specific query or need. The essence of IR lies in its ability to filter through vast amounts of data to find pieces of information that match a user's criteria, a task that has become increasingly complex with the exponential growth of digital data. [17]

Modern IR systems utilize advanced algorithms and machine learning techniques to improve the accuracy and relevance of search results. These systems employ natural language processing to understand the intent behind queries and to parse and index documents more effectively. Furthermore, the use of distributed computing allows IR systems to scale and manage the vast quantities of data on the web and in large databases. Semantic search and knowledge graphs represent the latest advancements in IR, aiming to understand the context and relationships between entities within the data, thus enabling more intuitive and accurate information retrieval.

The article Information Retrieval: Recent Advances and Beyond [18] makes a great job at summarizing some of the modern information retrieval techniques such as the diagram present in **Figure 5**.

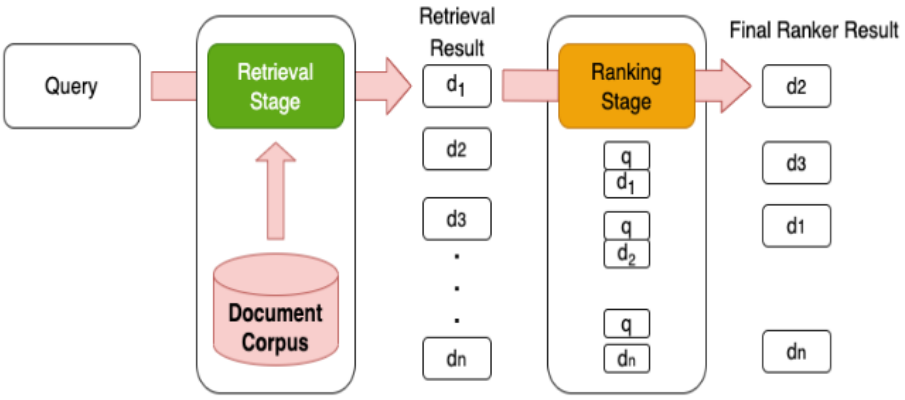


Figure 5. Overview of Modern Information Retrieval System

It also distinguishes between representation-based models and interaction-based models. Representation models use separate neural networks to turn queries and documents into feature vectors. They then calculate relevance scores between these vectors using simple methods like cosine similarity or more complex ones like a Multi-Layer Perceptron. For instance, the Deep Structure Semantic Model uses a Siamese network structure, while others like ARC-I and various Convolutional Neural Network based models use convolutional layers for feature extraction. Recurrent Neural Networks, including Long Short-Term Memory variants, are also employed for generating fixed-length vectors from sentences, capturing semantic relationships within texts. These models ultimately compute relevance by analyzing interactions between query and document features, often employing pooling and MLPs for final scoring.

While Interaction-focused models, like the Deep Relevance Matching Model and K-NRM, concentrate on capturing crucial matching signals between queries and documents that representation-focused models might miss. They start by creating interaction matrices using basic representations of queries and documents, then use deep learning to identify key interactions. These models employ various similarity measures and feature extraction techniques to enhance the accuracy of document retrieval, addressing challenges like non-differentiability and computational inefficiency in earlier models.

Taking a more wide approach, in text retrieval, the objective is to rank a set $L=[d1,d2,...,dn]$ of documents from a larger collection $D=[d1,d2,...,dm]$ in response to a query q. Dense retrieval models achieve this by representing both queries and documents as dense vectors through functions $\phi(\cdot)$ and $\psi(\cdot)$, which map them to l-dimensional vector spaces. The relevance $Rel(q,d)$ between a query and a document is determined by a similarity function $fsim(\phi(q),\psi(d))$ often implemented using neural networks.

$$Rel(q,d) = fsim(\varphi(q),\psi(d)) \quad (5)$$

Information retrieval is a crucial tool that our assistant will use to obtain relevant information from the PDFs in its database.

Chapter 4. Methodology

At this point, we know everything we need to start working in the creation of our AI Assistant, that we have named PDF Insight, as it is meant to contain a PDF files database with which the engineers and technicians of MAHLE Electronics SLU can interact through a telegram chatbot. This makes clear what out structural pipeline (represented as a flow diagram on **figure 6**) must be. First, we will start with a way of linking the OpenAI API and the Telegram API, then we will adapt it so we can communicate with our personalized assistant instead of with ChatGPT. Finally, we will compile all the code in an online server so the assistant can always be running. Now we will delve deeper into each module.

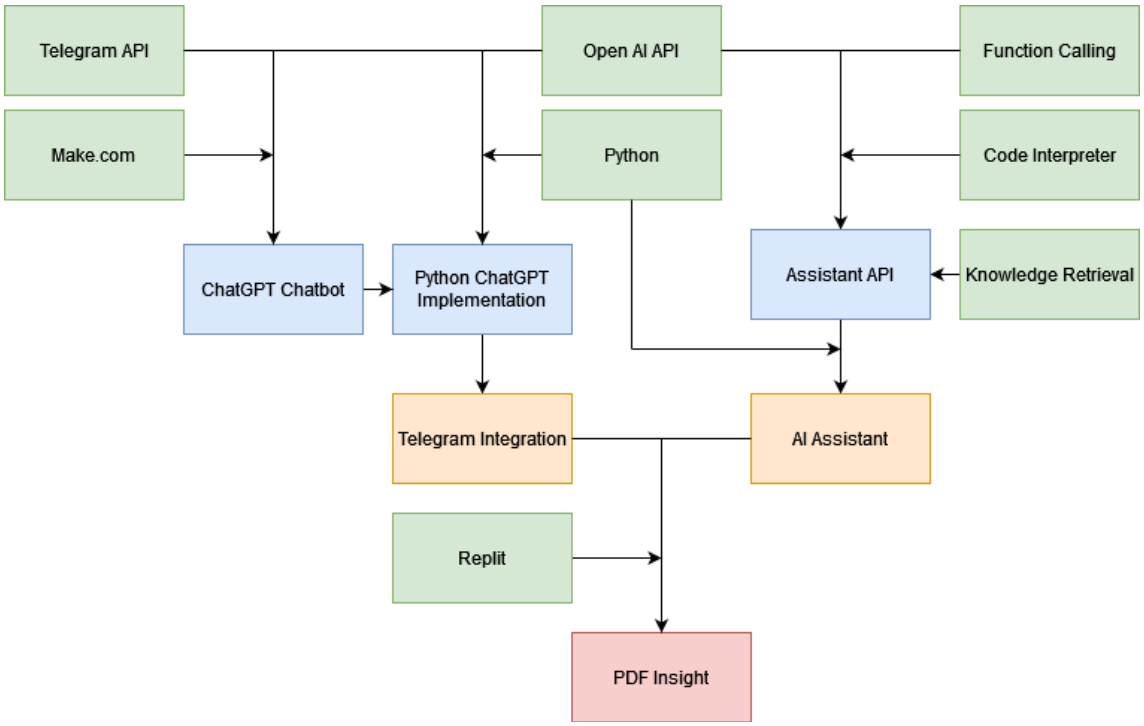


Figure 6. Overview of Modern Information Retrieval System

4.1 Telegram API

So, at first, we had to decide how will the assistant interact with the employees. Would we create a customized APP? Would we have them interact with the assistant directly? Eventually we decided on using the telegram API to create a chatbot that can relay the messages from the assistant directly to out phone, laptop, or pc.

This seemed the fastest way to give the assistant an intuitive interface so it could communicate easily with the workers. It also makes it easy to implement and spread and requires no installation whatsoever.

How do we create a chatbot? It can be done following simple steps:

1. Open Telegram. Either on your PC App, your phone, or the webpage.
2. Type /start to see the commands then /newbot.
3. Give it a name.
4. Give it a username (must end in _bot)
5. Congratulations, you've created a bot. With its own token that you must store.

All of this will be shown step by step with screenshots in appendix B.

Now that we have our bot, we must look for a way to connect it with ChatGPT, for that we will need the OpenAI API key that can be found in our OpenAI account. Once we have both we should be able to communicate with ChatGPT through telegram. This is how we do it.

4.2 Make.com

Make.com [19] is a powerful visual platform that allows users to design workflows, tasks, and automate processes across apps and systems. It enables the creation and automation of virtually anything within a unified interface, making it a versatile tool for individuals and businesses looking to streamline their operations.

In this section of the bachelor's degree thesis, we delve into the practical application of integrating Telegram with ChatGPT via Make.com, exemplifying the process of automating interactions between these platforms. The objective is to create a scenario where messages sent to a Telegram channel are automatically forwarded to ChatGPT, with the AI's response subsequently relayed back to the Telegram channel. **Figure 7** shows an example of the scenario we want to create.

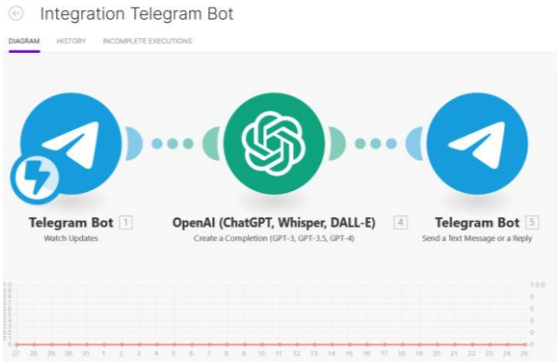


Figure 7. Make Scenario

The initiation of this process involves selecting Telegram from a plethora of available applications on Make.com and configuring it to "Watch updates." This setup ensures that any message posted to the designated Telegram channel triggers an event, captured via a webhook, thereby bypassing the need for periodic polling. A crucial step in this configuration is defining the webhook, which serves as a conduit for messages to be directly relayed to Make.com in real-time.

For the webhook to function, it necessitates linkage to the Telegram account using an API token, a unique identifier obtained during the bot's creation within Telegram. This token authenticates the connection between Make.com and Telegram, allowing for the seamless transmission of messages.

Upon establishing this connection, it is advisable to conduct a test to verify the operational integrity of the module. By sending a test message to the Telegram channel and observing its acknowledgment on Make.com, one can confirm the successful setup. This test also facilitates the retrieval of the chat ID, a critical identifier that distinguishes the specific Telegram chat, ensuring that responses are accurately directed.

The subsequent phase involves the integration of ChatGPT into the workflow. By adding another module within Make.com and selecting OpenAI's GPT, we can configure it to generate responses based on the incoming messages from Telegram. This step requires the specification of an account within Make.com, linked to OpenAI via an API key and organization ID, ensuring secure and authenticated access to GPT services.

In configuring the interaction with ChatGPT, one must define the type of completion desired, along with the selection of an appropriate model, such as GPT-3.5 Turbo. The critical component here is mapping the incoming Telegram message as the input to ChatGPT, necessitating the selection of the message content from the data structure provided by the Telegram module.

The final leg of the scenario entails relaying ChatGPT's response back to the original Telegram chat. This is achieved by adding a module for sending a text message via the Telegram bot, utilizing the previously acquired chat ID to pinpoint the exact destination within Telegram. The content of this message is derived from ChatGPT's output, effectively closing the loop of interaction.

Upon configuring these modules, a test run is essential to validate the entire workflow. By initiating a query and observing the response loop from Telegram to ChatGPT and back to Telegram, one can ensure the system's functionality.

To operationalize this scenario, it is imperative to activate the scheduling feature on Make.com, setting the scenario to run continuously. This activation guarantees that the workflow remains operational, ready to process messages as they arrive, thereby automating the interaction between Telegram and ChatGPT.

This exploration within the bachelor’s degree thesis not only demonstrates the practical application of Make.com in automating interactions between diverse platforms but also highlights the potential for further customization and expansion, leveraging the vast array of applications supported by Make.com to create complex, conditional workflows tailored to specific needs.

Although this is a good first attempt at the telegram implementation, we have many problems with it. The main issue is its versatility. Due to it being based on prebuilt modules, you really cannot work around those modules. There was no Assistant API module at the time, so we needed to get creative with http website modules. This proved to be way harder than to look for another implementation that allowed more coding and debugging. Of course, we are talking about a python implementation that lets you run code connecting telegram and ChatGPT. What we just did, but in python.

4.3 Python ChatGPT Implementation

Python seemed to be the best way to implement an assistant powered telegram chatbot. There is already information on how to build an assistant using python code, present in the assistant API website [6]. Luckily for us there are already plenty of implementations of python meant to connect ChatGPT with Telegram. We will now explain one of these implementations. The ChatGPT Telegram Bot is a github created by the use n3d1117 [20]. We will now briefly explain how this code works, making an emphasis on the telegram implementation aspect that is what we really want to take out from here.

4.3.1 Main (the whole code can be found in appendix C)

The "main.py" file contains the setup and execution process for a ChatGPT Telegram bot, starting with loading essential environment variables for secure API interactions and configuring logging to monitor the bot's activities. It validates the presence of critical environment variables like the Telegram bot token and OpenAI API key to ensure the bot has the necessary credentials to function. The setup includes configuring interaction parameters with the OpenAI API and Telegram settings, such as the API model and bot language. Additionally, a Plugin Manager is initialized to extend the bot's capabilities through plugins, and an OpenAI Helper object is created to manage interactions with OpenAI's services. The bot, encapsulated within the ChatGPT Telegram Bot class, is then initialized with these configurations, and set to run, processing incoming Telegram messages through OpenAI's GPT model, effectively enabling dynamic, AI-driven conversations within a Telegram channel.

In setting up the environment for a ChatGPT Telegram bot, the process begins by loading environment variables from a .env file using "load_dotenv()" from the "dotenv" package (**figure 8**). This step is crucial for securely managing sensitive information such as API keys and bot tokens, which are essential for interacting with external services like Telegram and OpenAI.

```
from dotenv import load_dotenv
    load_dotenv()
```

Figure 8. Load .env file

Following the environment setup, logging is configured to provide insightful information about the bot's operation. The "basicConfig" method from the logging module (**figure 9**) is used to define the format and level of log messages, ensuring that important events are recorded for debugging and monitoring purposes. Additionally, to avoid unnecessary log clutter from the "httpx" library, its logging level is set to "WARNING", reducing the verbosity of messages related to HTTP requests.

```
    logging.basicConfig(
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        level=logging.INFO
    )
    logging.getLogger("httpx").setLevel(logging.WARNING)
```

Figure 9. basicConfig

An essential part of the initialization process involves validating the presence of critical environment variables (**figure 10**), specifically the Telegram bot token and the OpenAI API key. This validation step checks for these variables and logs an error message listing any missing values, preventing the bot from running without the necessary credentials, which could lead to failures or unintended behavior.

```
    required_values = ['TELEGRAM_BOT_TOKEN', 'OPENAI_API_KEY']
    missing_values = [value for value in required_values if
os.environ.get(value) is None]
    if len(missing_values) > 0:
        logging.error(f'The following environment values are missing in
your .env: {", ".join(missing_values)}')
        exit(1)
```

Figure 10. Validation of Required Values

The configuration process for the ChatGPT Telegram Bot is meticulously designed to ensure seamless operation and interaction with the OpenAI and Telegram platforms. Initially, the setup delineates critical parameters within the “openai_config” and “telegram_config” dictionaries, which encompass essential settings like the OpenAI API key, the choice of the GPT model (defaulting to 'gpt-3.5-turbo'), the Telegram bot token, and the bot's default language. These configurations act as the backbone of the bot's functionality, dictating how it interfaces with the respective APIs and processes information.

To further augment the bot's capabilities, a “PluginManager” is strategically employed, drawing from a predefined list of plugins specified in the environment variables. This modular approach not only enhances the bot's functionality by incorporating additional features and behaviors but also promotes scalability and adaptability by allowing new plugins to be seamlessly integrated as the bot's operational requirements evolve.

Building upon this foundation, an “OpenAIHelper” object is instantiated, serving as the conduit for all interactions with OpenAI's API. This pivotal component, configured with the OpenAI settings and the PluginManager, ensures that the bot's functionality is enriched with the capabilities provided by the plugins, thereby enabling a more sophisticated and versatile interaction with OpenAI's services.

The culmination of this setup process is the initialization of the ChatGPT Telegram Bot, which is meticulously configured with the tailored Telegram settings and the “OpenAIHelper”. Upon activation, the bot vigilantly listens for incoming messages on Telegram, adeptly processing them with OpenAI's GPT model to deliver insightful and contextually relevant responses. This entire orchestration of the bot's functionality is encapsulated within the “main()” function, a safeguard that ensures the bot's setup and execution are meticulously managed and initiated only when the script is executed directly. This thoughtful design underscores the intricate interplay between configuration, functionality enhancement, and execution, ensuring the ChatGPT Telegram Bot operates efficiently and effectively within its designated ecosystem.

4.3.2 Utilities

This code includes utility functions and some core logic for a Telegram bot that interacts with users in various ways, using OpenAI's services for processing user inputs. OpenAI Helper will leverage these utilities in its functions. A few of the relevant ones are:

- 'message_text': Extracts the text from a Telegram message excluding bot commands.
- 'get_reply_to_message_id': Determines the appropriate message ID to which the bot should reply, supporting features like quoting the original message in the bot's response.
- 'is_allowed': Determines if a user is allowed to interact with the bot based on configuration settings like allowed user IDs.

4.3.3 OpenAI Helper

This Python script provides utility functions and a class "OpenAIHelper" designed to facilitate interaction with OpenAI's GPT models, including vision models. This class is the main reason the code works since it can retrieve messages and images generated from ChatGPT. It outlines a structured approach for developing a ChatGPT-based Telegram bot, incorporating OpenAI's GPT models to handle a wide range of tasks. It starts by importing necessary libraries and defining various GPT model configurations to support text and image processing tasks, demonstrating the bot's adaptability to different operational requirements.

At the core of the script is the "OpenAIHelper" class, designed to streamline interactions with the OpenAI API. Initialized with operational configurations and a "PluginManager", this class emphasizes a modular architecture, allowing for the easy integration of new features through plugins. This not only facilitates the expansion of the bot's capabilities but also ensures that it can be updated to meet future demands.

The "OpenAIHelper" class includes several key methods like "get_conversation_stats", "get_chat_response", and "get_chat_response_stream", which are crucial for managing chat interactions. These methods keep track of conversation history, enabling the bot to provide contextually relevant responses. This mechanism ensures the bot's interactions are coherent and aligned with the context of the conversation, enhancing user engagement.

The script also includes functionality for processing visual inputs, as seen in the "interpret_image" and "interpret_image_stream" methods. This feature expands the bot's utility beyond text, allowing it to analyze and respond to visual data. It showcases the bot's ability to engage with a wider variety of content, making it more versatile.

Error handling and rate limit management are integral parts of the script, incorporating retry mechanisms to address potential API rate limit errors effectively. This ensures the bot remains operational and reliable, even under high demand or when facing API limitations.

Additionally, the script supports localization and customization through the "localized_text" function and various configurable parameters. This allows the bot to serve a diverse user base by adapting to different languages and specific user preferences.

Overall, the script provides a comprehensive framework for building a ChatGPT-based Telegram bot capable of text and image processing. It combines a range of functionalities with a focus on modularity, reliability, and user-centric customization, laying the groundwork for a versatile and dependable bot.

4.3.4 Telegram Bot

This code defines a sophisticated Telegram bot built to handle a variety of tasks, including responding to user messages, generating images and text-to-speech (TTS) audio, transcribing audio messages, interpreting images, and supporting inline queries. The bot is initialized with a configuration dictionary and an "OpenAIHelper" object, which facilitates communication with the OpenAI API for generating responses.

The bot supports a range of commands, such as /help, /reset, /stats, and /resend, each with a specific function like displaying a help menu, resetting the conversation, showing usage statistics, or resending the last message. Additionally, the bot can generate images and text-to-speech (TTS) audio from prompts, and it has capabilities for transcribing audio messages and interpreting images using vision models.

The script utilizes the telegram library to define command handlers and message handlers for processing commands and messages. For instance, the prompt method handles incoming text messages, generating responses based on the content of the message and the context of the conversation. The bot also supports inline queries, allowing users to interact with the bot directly from any chat by typing "@botusername <query>".

Finally, the script also includes error handling to ensure the bot can manage errors and continue operating smoothly. Moreover, it features a usage tracking system to monitor user activity and enforce usage limits. The core of the script is designed to support a ChatGPT-based Telegram bot with multiple features for user interaction. It can handle text, images, and audio, making it a versatile tool for different use cases.

4.3.5 Evaluation

This code provides a solid groundwork to implement an integration between the assistant API and Telegram. However, it is much more complex than what we need to the point in which trying to alter it seems unfeasible. It has many plugins and capabilities that we don't care about for the purpose of this bachelor's degree thesis. It is a very complete and powerful code, which makes it difficult to adapt for our purposes. Even then the code snippets it provides will be very useful when forming the final integration.

4.4 Building an assistant

OpenAI's assistant API represents a significant leap forward in the development of artificial intelligence, aiming to establish a centralized AI agent. This agent differs fundamentally from previous iterations of AI interactions, such as chat completions. Unlike the endless and often directionless conversations enabled by chat completions, the AI agent is designed with a clear, singular purpose. Users can input information, pose questions, and engage in dialogue, all the while the AI maintains a focus on its predefined goal, offering more structured and goal-oriented interactions.

This approach allows for the creation of a tailored knowledge base, distinct from the vast, generalized pool of data used in chat completions. The assistant relies solely on the information provided to it, ensuring that its responses are aligned with the specific purpose it has been assigned. This marks a departure from the AI's previous reliance on external data sources, confining its responses to the user-defined knowledge base.

The development of the assistant API introduces several new concepts and components, including the assistant object, threads, messages, and runs, each playing a crucial role in the AI's operation. The assistant object acts as the core of the AI agent, housing the instructions and tools necessary for its function. Threads represent individual conversations or lines of inquiry within the broader context of the assistant's purpose. Messages, which can be either user-generated or produced by the assistant, are the building blocks of these threads. Runs, on the other hand, are the processes that execute the thread, requiring activation and monitoring to progress towards completion.

This structured approach to AI interaction offers enhanced capabilities and precision, allowing for the development of highly specialized and efficient AI agents. However, this complexity also necessitates a deeper understanding and more intricate management of the AI's components, from the creation and expansion of knowledge bases to the initiation and tracking of threads and runs.

In practice, this means that developers must now navigate a more segmented and detailed process to achieve the desired outcomes from their AI interactions. Each step, from ingesting data and defining the assistant's purpose to managing threads and monitoring runs, requires careful consideration and execution. This granularity offers the potential for highly customized and effective AI solutions but also demands a greater investment of time and resources in the development process.

The introduction of the assistant API by OpenAI is a testament to the evolving landscape of artificial intelligence, offering new possibilities and challenges in the pursuit of more purpose-driven and efficient AI agents. As developers and users alike adapt to this new framework, the potential for innovation and specialized application of AI is vast, promising a new era of AI utility and impact.

To build an AI assistant using OpenAI's API, you begin by specifying the location of your data sources. This initial step involves a function that allows the assistant to access and process PDF files or other supported document types. This function is critical as it defines where the assistant will retrieve its information, ensuring it has a comprehensive knowledge base for generating responses, it can be seen in **figure 11**.

```
from openai import OpenAI
import time


def saveFileOpenAI(location):
  client = OpenAI(api_key='OPENAI_API_KEY')
  file = client.files.create(file=open(location, "rb"),
purpose='assistants')
  return file.id
```

Figure 11. Location

Following the data specification, the next step is the creation and configuration of the assistant itself. This process is handled by a function that initializes the assistant with a unique set of instructions and selects an appropriate model for it to operate on. The function (present in **figure 12**) also incorporates tools for information processing and links the assistant to the previously defined data sources by including file IDs. This step is pivotal in establishing the core operational framework of the assistant, setting the stage for its interactive capabilities.

```
def startBotCreation(file_id):
  client = OpenAI(api_key='OPENAI_API_KEY')
  instructions = "You are a knowledge assistant. Use your knowledge
to best respond to customer queries"
  model = "gpt-3.5-turbo-1106"
  tools = [{"type":"retrieval"}]
  file_ids=[file_id]
  assistant =
client.beta.assistants.create(instructions=instructions,model=model
,tools=tools,file_ids=file_ids)
```

Figure 12. Assistant

With the assistant created, the focus shifts to initiating conversations, or "threads," through another function, that we can see in **figure 13**). Each thread represents a separate interaction with the assistant and requires a prompt to kickstart the dialogue. This aspect of the process underscores the conversational nature of the assistant, allowing for dynamic exchanges between the user and the AI.

```
def startThreadCreation(prompt):
  messages = [{"role": "user", "content":prompt}]
  client = OpenAI(api_key='OPENAI_API_KEY')
  thread =client.beta.threads.create(messages=messages)
  return thread.id
```

Figure 13. Thread

However, for the assistant to respond within a thread, it must first be activated or "run." This action is performed by a specific function that takes the thread ID and the assistant ID (**figure 14**), essentially waking the assistant to participate in the conversation. Without this crucial step, the thread exists in a dormant state, awaiting interaction.

```
def runAssistant(thread_id, assistand_id):
  client = OpenAI(api_key='OPENAI_API_KEY')
  run =
client.beta.threads.runs.create(thread_id=thread_id,assistant_id=as
sistant_id)
```

Figure 14. Run

To monitor the assistant's activity and ensure it is actively engaging in the conversation, another function allows you to check the run status. This function is essential for understanding whether the assistant is processing the query, has completed its response, or is encountering any issues.

```
def checkRunStatus(thread_id, run_id):
  client = OpenAI(api_key='OPENAI_API_KEY')
  run =
client.beta.threads.runs.retrieve(thread_id=thread_id,run_id=run_id
)
  return run.status
```

Figure 15. Run Status

Once the assistant has processed the interaction, you can retrieve its response through a function designed to pull the assistant's messages from the thread. This function not only fetches the direct response from the assistant but also provides references or "annotations" from the knowledge base, offering insights into how the assistant formulated its reply. This feature enhances the transparency and reliability of the assistant's responses.

Lastly, the capability to add new messages to an existing thread allows for ongoing conversations with the assistant. This functionality is crucial for maintaining a fluid and natural dialogue, mimicking human-like interactions. Each new message requires the thread to be "run" again, prompting the assistant to provide further responses. The code snippets to retrieve messages and add messages to the current thread can be found at **figures 16 and 17** respectively.

```
def retrieveThread(thread_id):
  client = OpenAI(api_key='OPENAI_API_KEY')
  thread_messages=client.beta.threads.messages.list(thread_id)
  list_messages = thread_messages.data
  assistant_message = list_messages[0]
  reference =
assistant_message.content[0].text.annotations[0].file_citation.quot
e
  message_text = assistant_message.content[0].text.value
  return message_text, reference
```

Figure 16. Receive Message

```
def addMessageToThread(thread_id, prompt):
  client = OpenAI(api_key='OPENAI_API_KEY')
  thread_message = client.beta.threads.messages.create(thread_id,
role="user", content=prompt)
```

Figure 17. Add Message

By combining these steps and utilizing the provided Python commands, we can construct an AI assistant tailored to our specific needs. This assistant can perform a range of tasks, from information retrieval and transactional tasks to scheduling, communication, and even entertainment, all depending on the tools and functions we integrate into its framework. For this project, the focus will be on enhancing the assistant's accuracy in analyzing PDF files, leveraging custom tools to refine its capabilities and ensure more precise and relevant responses.

Chapter 5. PDF Insight

PDF Insight is how we call the final implementation of our project. Since all the characteristics that it uses are already built in the Assistant API, such as code interpreter and knowledge retrieval, it doesn't need any additional tools to work. This resource hub, by Jannis Moore is the foundation upon which the final implementation of this bachelor's degree thesis is built [21]. It gives us both a good code structure with telegram already implemented and prompt that we can use to build our own customized tools and instructions. Finally, lets see how the actual AI Assistant looks like.

5.1 Instructions

The instructions are stored in a '.txt' file that the assistant will read. These instructions are built by ChatGPT with a specified prompt to assure that the assistant comprehends perfectly what it must do.

PDF Assistant Instructions

This assistant is specifically designed to provide information based on content from uploaded .pdf files. It's optimized for users seeking detailed answers exclusively from these documents.

Key Functions and Approach:

1. PDF Content Query Handling:

- When users ask questions, the assistant should first confirm that the query pertains to the content of uploaded .pdf files.

- Deliver precise answers derived solely from the .pdf content, ensuring accuracy and relevance. Do not deliver information that is not in the assistant files.

2. Data Synthesis:

- When the user asks a question, and the answer is not directly in the .pdf file the assistant should check if a table is mentioned where the information could be located.

- Then inform the user in which table the data is located, even if you cannot extract the actual data.

- Deliver precise answers derived solely from the .pdf content, ensuring accuracy and relevance.

Interaction Guidelines:

- Maintain a focused, knowledgeable, and user-friendly demeanor, emphasizing the assistant's unique capability to extract and rely on .pdf content.

- Clearly communicate the scope of assistance, ensuring users understand that answers are based exclusively on the content of uploaded .pdf files.

- Encourage users to provide specific questions or context to improve search accuracy and relevance.

- In cases where the assistant cannot find an answer within the .pdf resources, politely inform the user and suggest uploading additional .pdf files if necessary.

Remember, the assistant's effectiveness lies in its ability to accurately search and convey information from .pdf files, making it an invaluable resource for users requiring specialized knowledge contained within these documents. Use Markdown to format text and links.

These instructions are meant to ensure a couple of things:

- The assistant checks if the information is in the PDF.
- The assistant only answers with information contained in the PDF.
- The assistant checks for tables that may contain the information.
- The assistant references you to these tables.

This way we can optimize the flow of information and prevent the assistant of making things up when asked about information outside its database.

5.2 Telegram Integration (the whole code is present in appendix D)

The integration of a Telegram bot with OpenAI's conversational AI system is achieved using the “telebot” library for handling Telegram interactions and a bespoke “core_functions” module, which is tasked with database operations and other essential functionalities. The process begins with the bot acquiring the “TELEGRAM_TOKEN” from the environment variables, a crucial step for authenticating requests with the Telegram API. This token is then used to initialize a “telebot.TeleBot” instance, effectively bringing the bot to life on the Telegram platform.

Upon initialization, the bot is programmed to handle the /start command, a typical interaction that users initiate upon their first engagement with the bot. In response, the bot greets the user with a friendly message, setting a welcoming tone for the interaction. Beyond the initial greeting, the bot is equipped with a general message handler designed to catch and process any message that doesn't correspond to predefined commands like /start. This handler acts as the core of the bot's functionality, where user messages are processed, and queries are dispatched to the OpenAI API for generating responses, which are then relayed back to the user within the Telegram chat interface.

For managing conversations, the bot leverages OpenAI's thread and run creation capabilities. When a new conversation is initiated or when an existing chat mapping is absent, the bot sets up a new thread with OpenAI and triggers a run within that thread. This setup is pivotal for maintaining a coherent conversational context with the AI, ensuring that interactions are seamless and contextually aware.

An essential aspect of the bot's operation involves checking for and establishing a database mapping between Telegram chat IDs and OpenAI thread IDs. This step ensures that ongoing conversations are correctly associated with their respective threads in OpenAI, facilitating a consistent and personalized interaction for each user. In cases where a mapping doesn't exist, the bot proceeds to create a new thread and updates the database with the new mapping, thereby laying the groundwork for a structured conversation flow.

The bot's interaction with the OpenAI API extends to processing user inputs and generating AI-driven responses. Upon receiving a message from the user, the bot sends this input to the designated OpenAI thread and then retrieves the AI's response from the thread. This response, encapsulated within the latest message from the thread, is then presented back to the user in the Telegram chat, creating a dynamic and interactive user experience.

To ensure the bot remains responsive and agile, it employs thread-based polling, a technique that allows the bot to continuously monitor Telegram for updates without blocking the main thread. This approach is facilitated using Python's threading module, which enables the bot to maintain an attentive stance, ready to engage with incoming messages at any moment.

In summary, this code sets up a Telegram bot that uses OpenAI's conversational AI to respond to user messages. It handles the ‘/start’ command, processes general text messages, manages database mappings between Telegram chats and OpenAI threads, and ensures responses from the AI are relayed back to the appropriate Telegram user. The bot uses threading to keep polling for messages in the background.

5.3 Creating the assistant

We went over this code in the segment 4.4. The only difference from the functions presented is that now we store every variable in a JSON file. This way the program checks if the JSON file exists and uses the already built assistant instead of creating a new one. Everything else we need like the location of the files and the API keys are already present in the Replit project.

5.4 Results

Now that we have our assistant up and running, loaded with the information we want it to know we must test if the assistant answers correctly and if it is a viable option to implement as an asset in MAHLE Electronics SLU EMC Validation department. First, we loaded the assistant with 10 randomly selected PDFs related to EMC validation:

- CISPR 16-2-1 - Methods of measurement of disturbances and inmmunity - conducted disturbance measurements.
- CISPR 25 Ed5 2021
- CISPR 22 - Equipos de tecnologia de la informacion - Caracteristicas de las perturbaciones radioelectricas - Limites y metod
- PSA B21 7110 [2015-07] Version E
- B21 7112-Ed.OR (EMC for HV parts)
- B21_7112_-_Ind.A_-_EN
- PSA B21 7110 - Ind.F - EN (002)
- ISO 10605 [2023] (ESD)
- dg-11846-1.1 [PHEV] EMC and Electrical_testplan_draft
- Elektra Manual

Then we asked the assistant the following questions:

- What is God according to Christianity?
- What is Natural Language Processing?
- According to CISPR25 what are the ALSE Emissions?
- What is Elektra Fundamental Principles?
- According to the PSA B21 what is CE_03?
- Explain the CE03 test to me?
- Where should you put the probe in CE04? In which lines?
- How long must be the HV lines in an AN test?
- What are the options for recovering data?

Some of these questions were asked multiple times across multiple days to test the assistant's reliability at relaying the same information. In Appendix E we can find all the questions asked and the answers PDF Insight gave, but I have chosen to categorize them in **Figure 18**. First let's start with the test questions "What is god according to Christianity?" and "what is Natural Language Processing?".

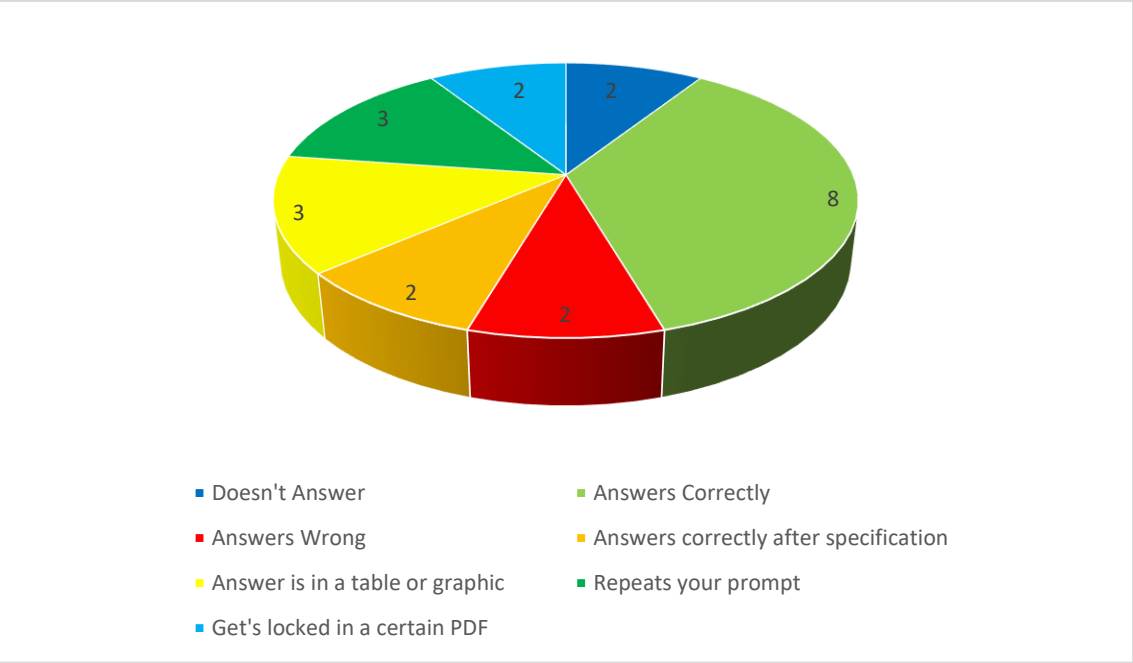


Figure 18. Accuracy of PDF Insight

As I said in segment 1.2.2, these two questions are built to test if the application is searching for its information correctly. Since neither of them are specified in the PDF files we fed to it, we expect the application to recognize this and don't answer. PDF Insight did the correct thing and refused to answer both questions, specifying that it would need to be built with a PDF File that contained that information if the user wanted to learn about the topic. I consider this a success as we do not want the valuable information from the PDF files to be contaminated by some outdated source.

Once we determined that the assistant cannot answer with information outside the PDF file, we focused on determining how accurate can it be when answering questions which answer is contained within the PDF file. Of the twenty questions asked about relevant information within the PDF file, PDF insight answered eleven correctly, three of them after specification; it also two wrong, saw three of them were in a table that it could not read and the five that are left he either repeated your prompt back to you or stated that the answer was not present in the currently open PDF file. When instructed to look in other PDF files in its database it was able to find the answer.

Let's focus on the two most strange responses, the repeated prompt and the assistant getting stuck in the current PDF. This two happen when the assistant must make too many requests and reaches its limit. If the assistant has not managed to formulate response the telegram bot will just send the last message left in the assistant interface. Comparatively, the assistant will try to search in the current PDF file before moving to other PDF files, so if the PDF file is too long it will run out of time before it can search other PDF files. This shows the limitations of having to work with the limited capabilities of the Assistant API that cannot handle this number of requests.

In a more positive note, when asked for the Limits of ALSE Emissions, located in the CISPR25, the assistant was able to realize the information was in a table although it could not access it. The assistant was also great at handling contradictory information. It simply stated the two pieces of information and the source of each one and when given a more specific prompt it was able to detect which of the contradictory pieces of information was relevant to the question.

In summary, the success ratio considering specifications, is close to 10:7 with a response rate of 60 seconds per question. This is far from ideal, but I will reserve my thoughts and evaluation for the conclusion.

Wanting to make a brief experiment I gave the same set of questions to a college:

- According to CISPR25 what are the ALSE Emissions?
- What is God according to Christianity?
- What is Natural Language Processing?
- What is Elektra Fundamental Principles?
- According to the PSA B21 what is CE_03?
- Explain the CE03 test to me?
- Where should you put the probe in CE04? In which lines?
- How long must be the HV lines in an AN test?
- What are the options for recovering data?

Then I gave him the PDF database and told him to please answer all the questions by searching in it, but that the information might not be in the PDFs so he should account for that.

He was able to intuitively eliminate the God according to Christianity question and a quick search through the PDF files also eliminated the Natural Language Processing question. However, it took for him over 20 minutes to answer the rest of the questions correctly and he was not certain of having done so until I confirmed it.

I think this proofs that an optimized version of this assistant with a reduced rate of failure could be incredibly helpful in a workplace specialized in following a strict set of rules through thorough testing such as the EMC Validation department from MAHLE Electronics SLU.

Chapter 6. Specifications

6.1 Pricing

This assistant utilizes and is intended to work with gpt4-turbo. This model is priced at 0.01$/1k Tokens or 0.0092€ / 1k Token in the input, and three times as much in the output.

So how many tokens does it use? Well, the last day of testing we reached over 3.5 million. That's about 35$. 7655 of these tokens were generated tokens. The other 3.5 million were context tokens. So, what are context tokens?[22] Apparently, no one really knows it. They seem to be additional tokens the Assistant API uses due to the accumulated number of messages in a thread, but there is no way to destroy messages in a thread without deleting the thread or the assistant itself.

6.2 Software

My main code is stored in Replit which has a lot of ram limitations in its free version. I think this is the main cause of the 60 second delay since asking the assistant directly from the assistant playground causes faster responses of at least half the time. The RAM is capped to 500MiB which is less than ideal, and it is partially guilty for the lengthy response time that PDF Insight has.

Chapter 7. Conclusion

We have managed to implement a useful and certainly pricey assistant that can help employees by giving them almost immediate solutions to their queries. It is at least competitive with the nowadays implementations of similar technology like ChatPDF and, although it is still on its early stages it certainly has potential. It relayed messages slower than ChatPDF but we must consider it was working with a larger database. Testing was done with only two PDFs and the answers were significantly shorter, around ten seconds for answer. It seems to be more precise at first glance, although further testing should be done in this aspect.

The aspect we can agree on is that currently this is not viable for an implementation as an asset in EMC Validation, or in any other workplace for that matter. The technology is way to modern and the pricing is a drawback that we cannot work around. “Context tokens” are still a mystery within the AI community and OpenAI support experts have not yet clarified what are they, what purpose they serve and how can we reduce their elevated usage. In a scenario where pricing was not an issue, it would be viable and beneficial to have an assistant in the workplace, even with its limited success it does not fare badly compared with a human employee. Even in this state and with a reduced database the assistant finds information way quicker than any human. Imagine in a database of 2 Gb of data. How long would a human take to find anything? While an assistant can do it in a matter of seconds.

There is a lot of testing that we simply have not done due to a lack of resources. Ideally, we should implement this assistant in a department and analyze its efficiency over a trimester. I think this would give us enough research to draw a better understanding of how useful an assistant can be. It does not seem to be any company that is publicly using the Assistant API. So, we do not have any concrete data yet.

Finally, I would like to propose an alternative solution to the inefficient system of information handling, specifically PDF files, present in MAHLE Electronics SLU. This is a github repository [23] that uses the RAG approach using the Chat Completion API. It is more scalable, less expensive and will give us at a minimum the same results we are getting now, if not better.

Chapter 8. References

[1] J. Bevendorff, M. Wiegmann, M. Potthast, and B. Stein, "Is Google Getting Worse? A Longitudinal Investigation of SEO Spam in Search Engines", Accessed: Mar. 02, 2024. [Online]. Available: https://github.com/webis-de/ECIR-24

[2] J. Jungbluth, K. Siedentopp, W. Krieger, and P. Plapper, "Combining Virtual and Robot Assistants—A Case Study about Integrating Amazon's Alexa as a Voice Interface in Robotics," *... -Academy Conference for ...*, no. June, 2018.

[3] "ChatPDF - Chat with any PDF!" Accessed: Feb. 24, 2024. [Online]. Available: https://www.chatpdf.com/

[4] S. Panda, "Enhancing PDF interaction for a more engaging user experience in library: Introducing ChatPDF," *IP Indian Journal of Library Science and Information Technology*, vol. 8, no. 1, pp. 20–25, Jun. 2023, doi: 10.18231/J.IJLSIT.2023.004.

[5] F. Antonius *et al.*, "Incorporating Natural Language Processing into Virtual Assistants: An Intelligent Assessment Strategy for Enhancing Language Comprehension," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 14, no. 10, p. 2023, Accessed: Mar. 02, 2024. [Online]. Available: www.ijacsa.thesai.org

[6] "Assistants overview - OpenAI API." Accessed: Feb. 25, 2024. [Online]. Available: https://platform.openai.com/docs/assistants/overview

[7] "IBM Global AI Adoption Index 2022," 2022.

[8] "How Netflix Utilizes Data Science - Lighthouse Labs." Accessed: Feb. 24, 2024. [Online]. Available: https://www.lighthouselabs.ca/en/blog/how-netflix-uses-data-to-optimize-their-product

[9] "What is the history of artificial intelligence (AI)? | Tableau." Accessed: Feb. 24, 2024. [Online]. Available: https://www.tableau.com/data-insights/ai/history

[10] "Sora." Accessed: Feb. 24, 2024. [Online]. Available: https://openai.com/sora

[11] "XCON – AI – The Future." Accessed: Feb. 24, 2024. [Online]. Available: https://aithefuture.wordpress.com/2018/05/08/xcon/

[12] L. J. Muhammad and E. A. Algehyne, "Fuzzy based expert system for diagnosis of coronary artery disease in nigeria," *Health Technol (Berl)*, vol. 11, no. 2, 2021, doi: 10.1007/s12553-021-00531-z.

[13] "What is Natural Language Processing (NLP)? A Beginner's Guide | DataCamp." Accessed: Feb. 24, 2024. [Online]. Available: https://www.datacamp.com/blog/what-is-natural-language-processing

[14] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", Accessed: Feb. 24, 2024. [Online]. Available: https://github.com/tensorflow/tensor2tensor

[15] "Generative Pre-Trained Transformer (GPT) definition | Encord." Accessed: Feb. 24, 2024. [Online]. Available: https://encord.com/glossary/gpt-definition/

[16] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[17] I. C. Mogotsi, "Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: Introduction to information retrieval," *Inf Retr Boston*, vol. 13, no. 2, 2010, doi: 10.1007/s10791-009-9115-y.

[18] K. A. Hambarde and H. Proenca, "Information Retrieval: Recent Advances and beyond," *IEEE Access*, vol. 11, 2023, doi: 10.1109/ACCESS.2023.3295776.

[19] "Make | Automation Software | Connect Apps & Design Workflows." Accessed: Mar. 03, 2024. [Online]. Available: https://www.make.com/en

[20] "GitHub - n3d1117/chatgpt-telegram-bot: A Telegram bot that integrates with OpenAI's official ChatGPT APIs to provide answers, written in Python." Accessed: Feb. 25, 2024. [Online]. Available: https://github.com/n3d1117/chatgpt-telegram-bot

[21] "How I build OpenAI Assistants in 2024 (GPT Chatbot Framework 2.0) | Integraticus." Accessed: Feb. 25, 2024. [Online]. Available: https://hub.integraticus.com/how-i-build-openai-assistants-in-2024-gpt-chatbot-framework-2-0/

[22] "Assistant API - What are Context Tokens in the Billing calculation? - API - OpenAI Developer Forum." Accessed: Feb. 26, 2024. [Online]. Available: https://community.openai.com/t/assistant-api-what-are-context-tokens-in-the-billing-calculation/497675/17

[23] "GitHub - mayooear/gpt4-pdf-chatbot-langchain at feat/vectordbqa." Accessed: Feb. 26, 2024. [Online]. Available: https://github.com/mayooear/gpt4-pdf-chatbot-langchain/tree/feat/vectordbqa?tab=readme-ov-file

Chapter 9. Appendix A

ISO 10605 [2023] (ESD).pdf

PDF: Hello and welcome to the helpful PDF file on Road vehicles test methods for electrical disturbances from electrostatic discharge! This document, ISO 10605:2023, provides essential information on testing procedures to ensure the safety and reliability of road vehicles in relation to electrostatic discharge.

1. What is the purpose of ISO 10605:2023 and why is it important for road vehicles?

2. How can this standard help manufacturers and regulators in the automotive industry?

3. Are there any specific requirements or guidelines outlined in the document for conducting tests on electrical disturbances from electrostatic discharge?

Me: Describe the ESD powered-up test?

PDF: The ESD powered-up test, as described in ISO 10605:2023, is a component immunity test method that involves applying direct and indirect discharges to the Device Under Test (DUT) while it is powered up. This test is crucial for evaluating the immunity of electronic modules in road vehicles to electrical disturbances from electrostatic discharge.

1. **Direct Type Discharges**: In the direct type discharges, either contact or air discharge mode is applied directly to the DUT and to remote parts accessible by vehicle users, such as surfaces of switches, diagnostic connectors, and buttons. This simulates discharges that may occur directly on the components themselves.

2. **Indirect Type Discharges**: The indirect type discharges simulate discharges that occur to other conductive objects in the vicinity of the DUT. These discharges are applied through an intervening metal, such as to a ground point. This type of discharge helps evaluate the immunity of the DUT to disturbances from electrostatic discharge in the surrounding environment.

By subjecting the DUT to direct and indirect discharges while it is powered up, the ESD powered-up test assesses the ability of electronic modules in vehicles to withstand and operate reliably in the presence of electrostatic discharge events. This test is essential for ensuring the overall performance and safety of electronic components in road vehicles under real-world conditions where ESD events can occur.

For more detailed information on the ESD powered-up test and its specific procedures, you can refer to ISO 10605:2023, particularly in section 8 of the document.

B21 7112-Ed.OR (EMC for HV parts).pdf

PDF: Hello and welcome to the PSA PEUGEOT - CITROËN Specifications for Electrical and Electronic High Voltage Equipments PDF file! This document provides detailed information on the environmental specifications, electrical characteristics, and restrictions for high voltage equipment. It was created on 16/09/2015 and is available for internal use. If you have any questions, please contact normespsa@mpsa.com.

Example questions:

1. What are the key environmental specifications outlined in this document?

2. Who were the participants involved in drafting and checking this norm?

3. How can I access the full content of the PSA PEUGEOT - CITROËN B21 7112 standard?

Me: How long should the HV lines be in a test setup?

PDF: In the PSA PEUGEOT - CITROËN document, the length of the shielded high voltage ("200V") lines should be 1700 ± 300 mm. Additionally, the length of the high voltage lines should be 1700 ± 300 mm. These specifications ensure proper testing conditions and compliance with the standards outlined in the document.

Me: Where should the probe me connected in CE_04 test?

PDF: In the CE_04 test, the current probe measurements have to be performed on the HV+ and HV- power supply lines, and the three-phase lines of the electric motor, separately (if applicable) and commonly. The emission should be measured with the probe positioned at d = 50 mm and d = 750 mm (depending on harness length) from the EUT. The current measuring probe should be placed successively in various positions along the concerned branch as indicated in the document.

Me: Should the communication lines be measured?

PDF: In the PSA PEUGEOT - CITROËN document, there is no specific mention of measuring communication lines in the context of EMC testing for high voltage equipment. The focus is primarily on the measurement of high voltage power lines, three-phase lines, and related components. However, it is always recommended to refer to the specific testing requirements and standards applicable to the communication lines in the given system to ensure comprehensive EMC testing and compliance.

Me: How does the CE_03 test work?

PDF: In the PSA PEUGEOT - CITROËN document, the CE_03 test involves the measurement of conducted emissions on the high voltage power supply lines (HV+ and HV-) and the three-phase lines of the electric motor. The emissions are measured with the probe positioned at specific distances from the Equipment Under Test (EUT) along the concerned branches. The test setup and procedure for CE_03 are detailed in the document to ensure accurate measurement of conducted emissions on the specified lines.

Me: No, it doesn't

PDF: I apologize for the oversight. In the PSA PEUGEOT - CITROËN document, the CE_03 test procedure is not explicitly described. For detailed information on how the CE_03 test works, it is recommended to refer to the specific testing standards and guidelines provided by the relevant regulatory bodies or consult with EMC testing experts familiar with the CE_03 test requirements.

Me: Yes it is, in BC/CE_03: RF CONDUCTED EMISSION ON AC OR DC LINES

PDF: I apologize for the confusion. In the PSA PEUGEOT - CITROËN document, the CE_03 test, which involves RF conducted emission on AC or DC lines, is detailed in the document. The test procedure for CE_03 includes measuring the level of radio frequency conducted disturbances generated by equipment connected to power mains through its AC or DC lines. The specific test setup, measurement points, and requirements for CE_03 are outlined in the document.

Me: Describe the objective of the CI_05 test

PDF: In the PSA PEUGEOT - CITROËN document, the objective of the CI_05 test is to demonstrate the immunity of equipment connected to power mains, such as a battery charger, to harmonics and interharmonics along AC power lines. The test aims to ensure that the equipment can withstand and operate effectively in the presence of harmonics and interharmonics on the AC power lines. The test procedure for CI_05 complies with the CEI 61000-4-13 standard, and the test level requirements are established based on the test method specified in the standard.

Both PDFs

Me: What is ESD?

PDF: ESD stands for Electrostatic Discharge. It is the sudden flow of electricity between two electrically charged objects caused by contact, an electrical short, or dielectric breakdown. ESD can damage electronic components and integrated circuits, leading to malfunctions or complete failure of electronic devices. Standards such as ISO 10605 provide guidelines for testing electronic modules to evaluate their resistance to electrical disturbances from electrostatic discharge.

Me: What is god according to christianity?

PDF: I specialize in technical and factual information. If you have any questions related to science, technology, or general knowledge, feel free to ask!

Me: What is natural language processing?

PDF: Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between computers and humans using natural language. It involves the development of algorithms and models that enable computers to understand, interpret, and generate human language. NLP techniques are used in various applications such as language translation, sentiment analysis, chatbots, speech recognition, and text summarization. By processing and analyzing large amounts of natural language data, NLP systems can extract meaningful information and provide valuable insights.

Me: Describe the ESD powered up test

PDF: The ESD powered-up test, as described in ISO 10605:2023, involves the application of direct and indirect discharges to the Device Under Test (DUT) while it is powered on. This test is designed to evaluate the immunity of electronic components to electrostatic discharges in real-world operating conditions.

1. **Direct Discharges**: Direct discharges can be in contact or air discharge mode and are applied directly to the DUT and other remote parts accessible by users, such as switches, diagnostic connectors, and buttons.

2. **Indirect Discharges**: Indirect discharges simulate discharges that occur to conductive objects near the DUT and are applied through an intervening metal, such as a ground plane.

By subjecting the DUT to direct and indirect discharges while it is powered up, the test assesses the component's ability to withstand electrostatic discharges during normal operation. This helps in ensuring the reliability and robustness of electronic components in the presence of ESD events.

Me: Can you tell me the limits of the signal lines in CE_04?

PDF: In the document provided, PSA PEUGEOT - CITROËN B21 7112, the limits for RF conducted emission on signal lines in test BC/CE_04 are specified according to CISPR 22 standards. Unfortunately, the specific limits for signal lines in CE_04 are not mentioned in the excerpts you provided. For detailed information on the limits of RF conducted emissions on signal lines in test CE_04, you may need to refer to the complete document or the referenced standard CISPR 22.

Me: Describe me an ESD setup

PDF: An Electrostatic Discharge (ESD) setup typically consists of the following components and elements:

1. **ESD Generator**: This is the device that generates controlled electrostatic discharges. It can be of various types such as Human Body Model (HBM), Machine Model (MM), Charged Device Model (CDM), or Contact and Air Discharge modes.

2. **Device Under Test (DUT)**: The electronic component or system being tested for its immunity to electrostatic discharges.

3. **Ground Plane**: A conductive surface that provides a reference ground for the ESD setup to ensure proper discharge paths.

4. **ESD Mat**: A dissipative mat placed on the work surface to prevent the buildup of static charges and provide a controlled path for discharges.

5. **Ground Connection Wire**: Used to connect the ESD generator, DUT, and other components to the ground plane to ensure proper grounding.

6. **Test Fixture**: Depending on the test requirements, a test fixture may be used to hold the DUT in place during testing.

7. **Monitoring Equipment**: Instruments such as oscilloscopes, current probes, and voltage probes may be used to monitor the effects of electrostatic discharges on the DUT.

8. **Safety Equipment**: Personal protective equipment such as ESD wrist straps, gloves, and safety glasses may be used to protect the operator and prevent accidental damage to sensitive components.

9. **Test Procedure**: A defined test procedure outlining the test parameters, discharge levels, test points, and measurement methods to ensure consistent and repeatable testing.

By setting up these components in a controlled environment following standard procedures, an ESD setup can effectively evaluate the susceptibility of electronic devices to electrostatic discharges and ensure their reliability in real-world scenarios.

Me: What about the positions of the components?

PDF: In an ESD setup, the positions of components play a crucial role in ensuring accurate and reliable testing. Here are some key considerations for positioning components in an ESD setup:

1. **Device Under Test (DUT)**:

- The DUT should be positioned securely in the test fixture or on the test table to prevent movement during testing.

- Ensure that the DUT is oriented correctly according to the test requirements, especially for direct discharge testing where the angle of discharge matters.

2. **Ground Plane**:

- The ground plane should be positioned beneath the DUT to provide a reference ground for the ESD setup.

- The test points on the DUT should be aligned with the ground plane for accurate discharge paths.

3. **ESD Generator**:

- Position the ESD generator in a convenient location for easy access during testing.

- Ensure that the discharge tip or air discharge tip is positioned correctly for contact or air discharge testing, respectively.

4. **Ground Connection Wire**:

- Connect the ground connection wire securely between the ESD generator, DUT, and ground plane to establish proper grounding.

5. **Monitoring Equipment**:

- Position monitoring equipment such as oscilloscopes and probes in a way that allows for easy observation of the DUT's response to electrostatic discharges.

6. **Safety Equipment**:

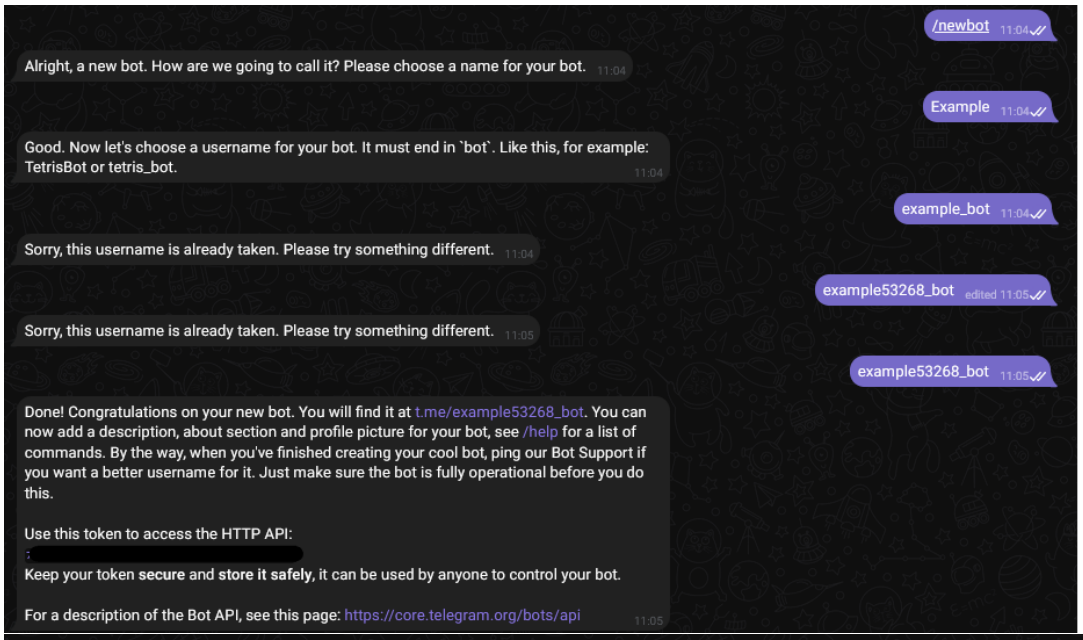
- Ensure that safety equipment such as ESD wrist straps and gloves are worn by operators and positioned within reach for quick access.

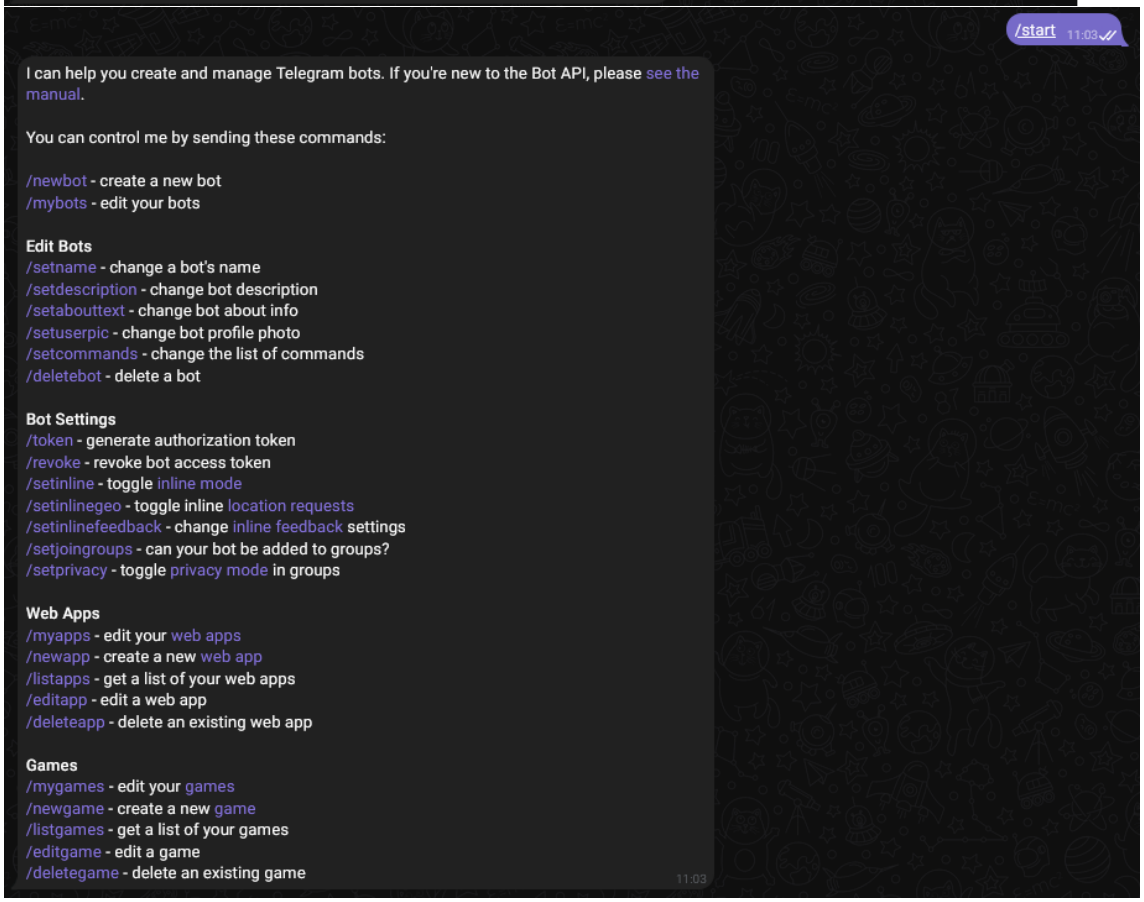
7. **Test Procedure**:

- Follow the defined test procedure to ensure that components are positioned according to the specified test points, distances, and orientations for consistent and accurate testing.

By carefully positioning components in the ESD setup according to the test requirements and procedures, you can conduct effective and reliable electrostatic discharge testing to evaluate the immunity of electronic devices to ESD events.

Chapter 10. Appendix B





Chapter 11. Appendix C

```
import logging
import os

from dotenv import load_dotenv

from plugin_manager import PluginManager
from openai_helper import OpenAIHelper, default_max_tokens,
are_functions_available
from telegram_bot import ChatGPTTelegramBot


def main():
    # Read .env file
    load_dotenv()

    # Setup logging
    logging.basicConfig(
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        level=logging.INFO
    )
    logging.getLogger("httpx").setLevel(logging.WARNING)

    # Check if the required environment variables are set
    required_values = ['TELEGRAM_BOT_TOKEN', 'OPENAI_API_KEY']
    missing_values = [value for value in required_values if
os.environ.get(value) is None]
    if len(missing_values) > 0:
        logging.error(f'The following environment values are missing in
your .env: {", ".join(missing_values)}')
        exit(1)

    # Setup configurations
    model = os.environ.get('OPENAI_MODEL', 'gpt-3.5-turbo')
    functions_available = are_functions_available(model=model)
    max_tokens_default = default_max_tokens(model=model)
    openai_config = {
        'api_key': os.environ['OPENAI_API_KEY'],
        'show_usage': os.environ.get('SHOW_USAGE', 'false').lower() ==
'true',
        'stream': os.environ.get('STREAM', 'true').lower() == 'true',
        'proxy': os.environ.get('PROXY', None) or
os.environ.get('OPENAI_PROXY', None),
        'max_history_size': int(os.environ.get('MAX_HISTORY_SIZE', 15)),
        'max_conversation_age_minutes':
int(os.environ.get('MAX_CONVERSATION_AGE_MINUTES', 180)),
```

```
        'assistant_prompt': os.environ.get('ASSISTANT_PROMPT', 'You are a
helpful assistant.'),
        'max_tokens': int(os.environ.get('MAX_TOKENS',
max_tokens_default)),
        'n_choices': int(os.environ.get('N_CHOICES', 1)),
        'temperature': float(os.environ.get('TEMPERATURE', 1.0)),
        'image_model': os.environ.get('IMAGE_MODEL', 'dall-e-2'),
        'image_quality': os.environ.get('IMAGE_QUALITY', 'standard'),
        'image_style': os.environ.get('IMAGE_STYLE', 'vivid'),
        'image_size': os.environ.get('IMAGE_SIZE', '512x512'),
        'model': model,
        'enable_functions': os.environ.get('ENABLE_FUNCTIONS',
str(functions_available)).lower() == 'true',
        'functions_max_consecutive_calls':
int(os.environ.get('FUNCTIONS_MAX_CONSECUTIVE_CALLS', 10)),
        'presence_penalty': float(os.environ.get('PRESENCE_PENALTY',
0.0)),
        'frequency_penalty': float(os.environ.get('FREQUENCY_PENALTY',
0.0)),
        'bot_language': os.environ.get('BOT_LANGUAGE', 'en'),
        'show_plugins_used': os.environ.get('SHOW_PLUGINS_USED',
'false').lower() == 'true',
        'whisper_prompt': os.environ.get('WHISPER_PROMPT', ''),
        'vision_model': os.environ.get('VISION_MODEL', 'gpt-4-vision-
preview'),
        'enable_vision_follow_up_questions':
os.environ.get('ENABLE_VISION_FOLLOW_UP_QUESTIONS', 'true').lower() ==
'true',
        'vision_prompt': os.environ.get('VISION_PROMPT', 'What is in this
image'),
        'vision_detail': os.environ.get('VISION_DETAIL', 'auto'),
        'vision_max_tokens': int(os.environ.get('VISION_MAX_TOKENS',
'300')),
        'tts_model': os.environ.get('TTS_MODEL', 'tts-1'),
        'tts_voice': os.environ.get('TTS_VOICE', 'alloy'),
    }

    if openai_config['enable_functions'] and not functions_available:
        logging.error(f'ENABLE_FUNCTIONS is set to true, but the model
{model} does not support it. '
                      f'Please set ENABLE_FUNCTIONS to false or use a
model that supports it.')
        exit(1)
    if os.environ.get('MONTHLY_USER_BUDGETS') is not None:
        logging.warning('The environment variable MONTHLY_USER_BUDGETS is
deprecated. '
                        'Please use USER_BUDGETS with BUDGET_PERIOD
instead.')
```

```
    if os.environ.get('MONTHLY_GUEST_BUDGET') is not None:
        logging.warning('The environment variable MONTHLY_GUEST_BUDGET is
deprecated. '
                        'Please use GUEST_BUDGET with BUDGET_PERIOD
instead.')

    telegram_config = {
        'token': os.environ['TELEGRAM_BOT_TOKEN'],
        'admin_user_ids': os.environ.get('ADMIN_USER_IDS', '-'),
        'allowed_user_ids': os.environ.get('ALLOWED_TELEGRAM_USER_IDS',
'*'),
        'enable_quoting': os.environ.get('ENABLE_QUOTING',
'true').lower() == 'true',
        'enable_image_generation':
os.environ.get('ENABLE_IMAGE_GENERATION', 'true').lower() == 'true',
        'enable_transcription': os.environ.get('ENABLE_TRANSCRIPTION',
'true').lower() == 'true',
        'enable_vision': os.environ.get('ENABLE_VISION', 'true').lower()
== 'true',
        'enable_tts_generation': os.environ.get('ENABLE_TTS_GENERATION',
'true').lower() == 'true',
        'budget_period': os.environ.get('BUDGET_PERIOD',
'monthly').lower(),
        'user_budgets': os.environ.get('USER_BUDGETS',
os.environ.get('MONTHLY_USER_BUDGETS', '*')),
        'guest_budget': float(os.environ.get('GUEST_BUDGET',
os.environ.get('MONTHLY_GUEST_BUDGET', '100.0'))),
        'stream': os.environ.get('STREAM', 'true').lower() == 'true',
        'proxy': os.environ.get('PROXY', None) or
os.environ.get('TELEGRAM_PROXY', None),
        'voice_reply_transcript':
os.environ.get('VOICE_REPLY_WITH_TRANSCRIPT_ONLY', 'false').lower() ==
'true',
        'voice_reply_prompts': os.environ.get('VOICE_REPLY_PROMPTS',
'').split(';'),
        'ignore_group_transcriptions':
os.environ.get('IGNORE_GROUP_TRANSCRIPTIONS', 'true').lower() == 'true',
        'ignore_group_vision': os.environ.get('IGNORE_GROUP_VISION',
'true').lower() == 'true',
        'group_trigger_keyword': os.environ.get('GROUP_TRIGGER_KEYWORD',
''),
        'token_price': float(os.environ.get('TOKEN_PRICE', 0.002)),
        'image_prices': [float(i) for i in os.environ.get('IMAGE_PRICES',
"0.016,0.018,0.02").split(",")],
        'vision_token_price': float(os.environ.get('VISION_TOKEN_PRICE',
'0.01')),
        'image_receive_mode': os.environ.get('IMAGE_FORMAT', "photo"),
        'tts_model': os.environ.get('TTS_MODEL', 'tts-1'),
```

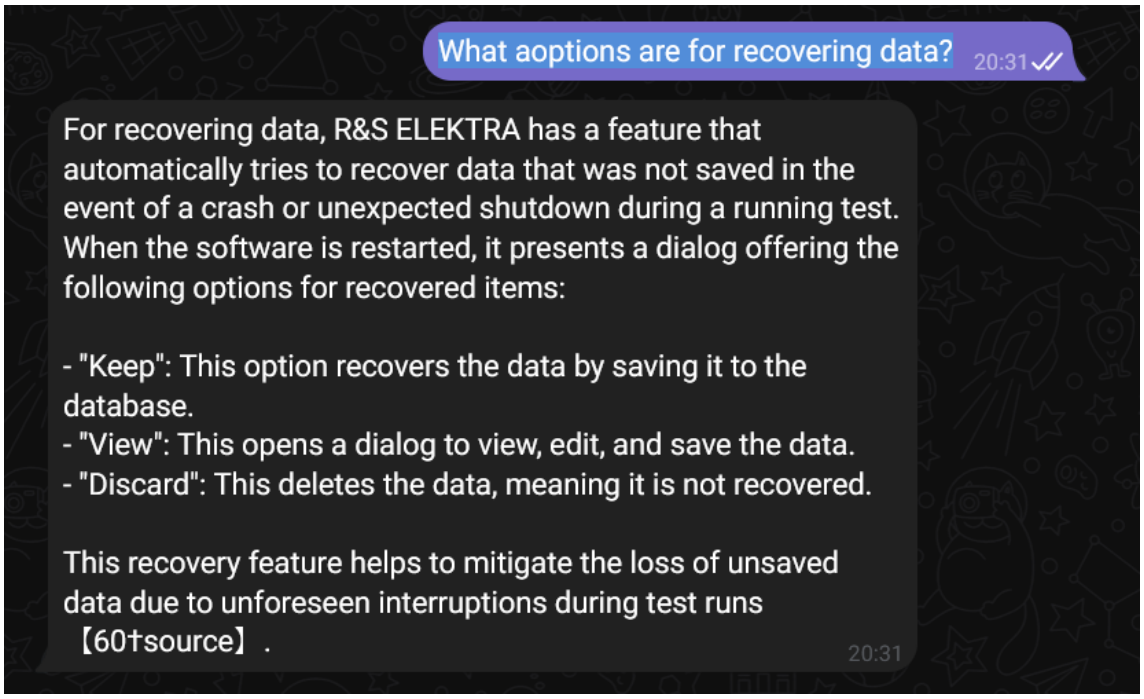
```
        'tts_prices': [float(i) for i in os.environ.get('TTS_PRICES',
"0.015,0.030").split(",")],
        'transcription_price':
float(os.environ.get('TRANSCRIPTION_PRICE', 0.006)),
        'bot_language': os.environ.get('BOT_LANGUAGE', 'en'),
    }

    plugin_config = {
        'plugins': os.environ.get('PLUGINS', '').split(',')
    }

    # Setup and run ChatGPT and Telegram bot
    plugin_manager = PluginManager(config=plugin_config)
    openai_helper = OpenAIHelper(config=openai_config,
plugin_manager=plugin_manager)
    telegram_bot = ChatGPTTelegramBot(config=telegram_config,
openai=openai_helper)
    telegram_bot.run()


if __name__ == '__main__':
    main()
```

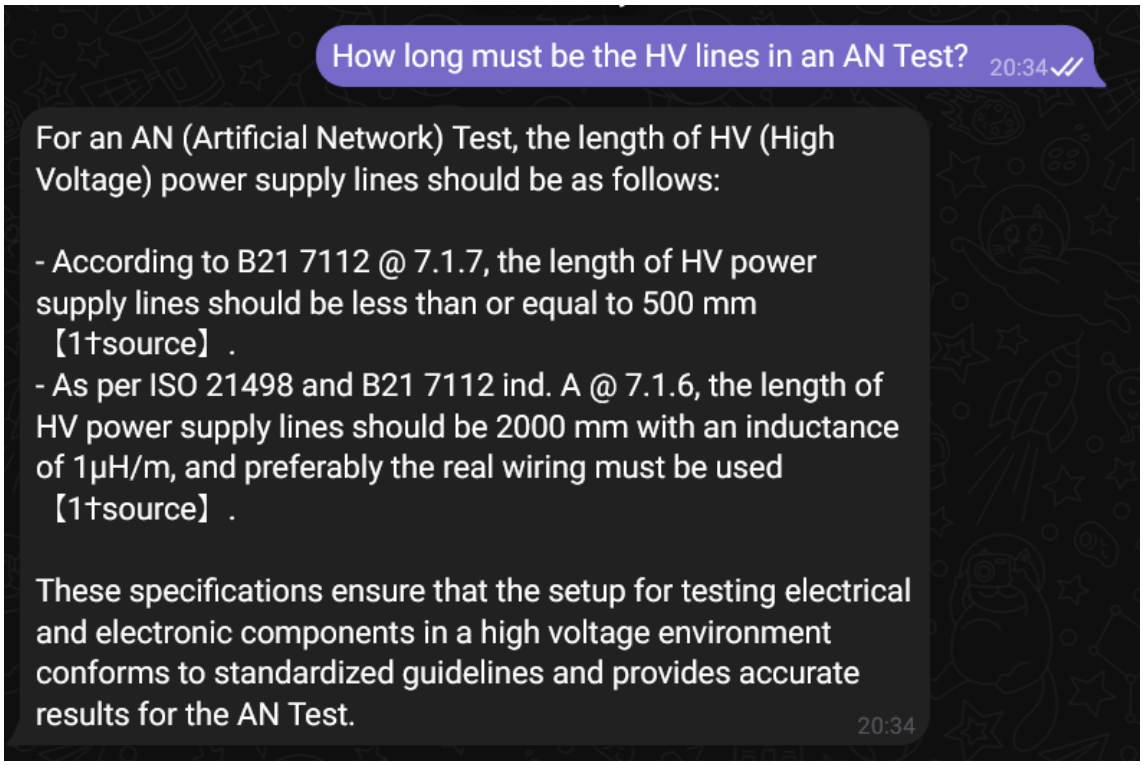
Chapter 12. Appendix D

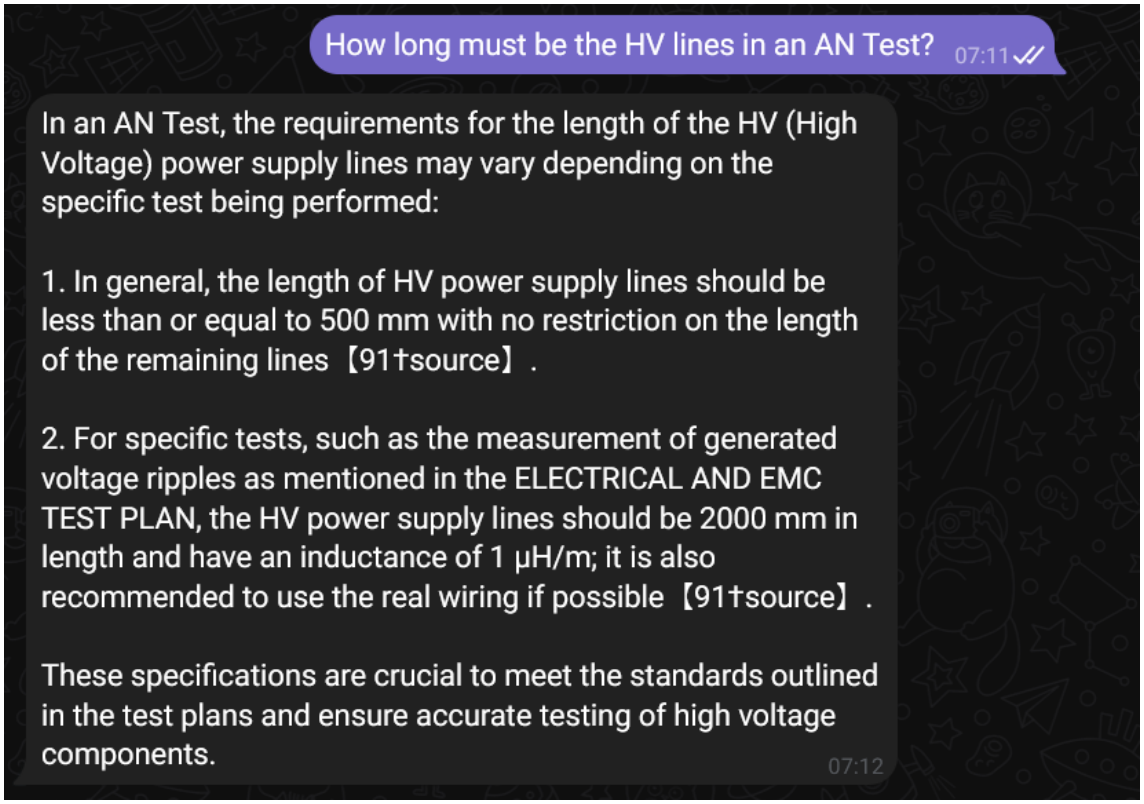
```
import os
import logging
import telebot
import core_functions

# Configure logging
logging.basicConfig(level=logging.INFO)


# Defines if a DB mapping is required
def requires_mapping():
  return True


def setup_routes(app, client, tool_data, assistant_id):
  # Initialize Telegram
  TELEGRAM_TOKEN = os.environ.get('TELEGRAM_TOKEN')
  if not TELEGRAM_TOKEN:
    raise ValueError("No Telegram token found in environment variables")
  bot = telebot.TeleBot(TELEGRAM_TOKEN)

  @bot.message_handler(commands=['start'])
  def send_welcome(message):
    telegram_chat_id = message.chat.id
    logging.info("Starting a new conversation...")

    chat_mapping = core_functions.get_chat_mapping("telegram",
                                                   telegram_chat_id,
                                                   assistant_id)


    # Check if this chat ID already has a thread ID
    if not chat_mapping:
      thread = client.beta.threads.create()

      # Save the mapping
      core_functions.update_chat_mapping("telegram", telegram_chat_id,
                                         assistant_id, thread.id)

      logging.info(f"New thread created with ID: {thread.id}")

    bot.reply_to(message, "Howdy, how can I help you?")

  @bot.message_handler(func=lambda message: True)
  def echo_all(message):
    telegram_chat_id = message.chat.id
    user_input = message.text
```

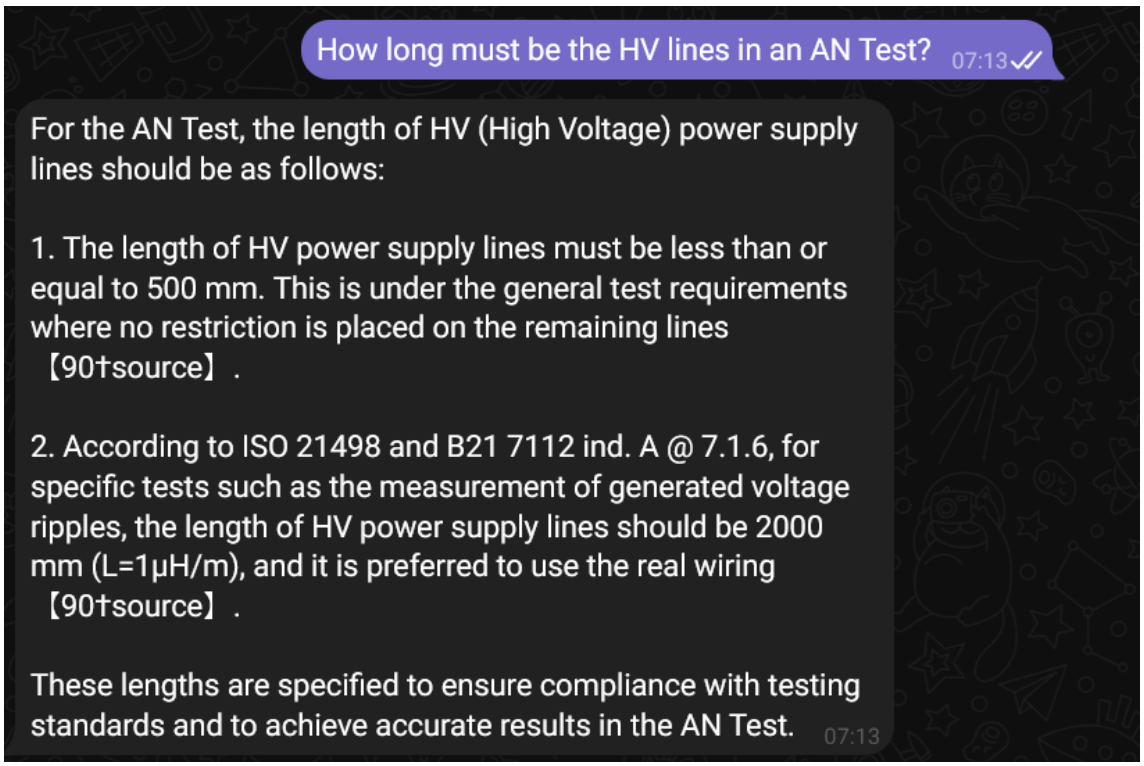
```
    db_entry = core_functions.get_chat_mapping("telegram",
telegram_chat_id,
                                               assistant_id)

    thread_id = core_functions.get_value_from_mapping(db_entry,
"thread_id")

    if not thread_id:

      thread = client.beta.threads.create()

      # Save the mapping
      core_functions.update_chat_mapping("telegram", telegram_chat_id,
                                         assistant_id, thread.id)

      thread_id = thread.id

      logging.info(f"XXXXXX: {thread_id}")

      if not thread_id:
        logging.error("Error: Missing OpenAI thread_id")
        return

    logging.info(
        f"Received message: {user_input} for OpenAI thread ID:
{thread_id}")

    client.beta.threads.messages.create(thread_id=thread_id,
                                        role="user",
                                        content=user_input)

    run = client.beta.threads.runs.create(thread_id=thread_id,
                                          assistant_id=assistant_id)
    # This processes any possible action requests
    core_functions.process_tool_calls(client, thread_id, run.id,
tool_data)

    messages = client.beta.threads.messages.list(thread_id=thread_id)
    response = messages.data[0].content[0].text.value

    # Use the original Telegram chat ID here, not the OpenAI thread ID
    bot.send_message(telegram_chat_id, response, parse_mode='Markdown')

  # Start polling in a separate thread
  from threading import Thread
  Thread(target=bot.polling).start()
```

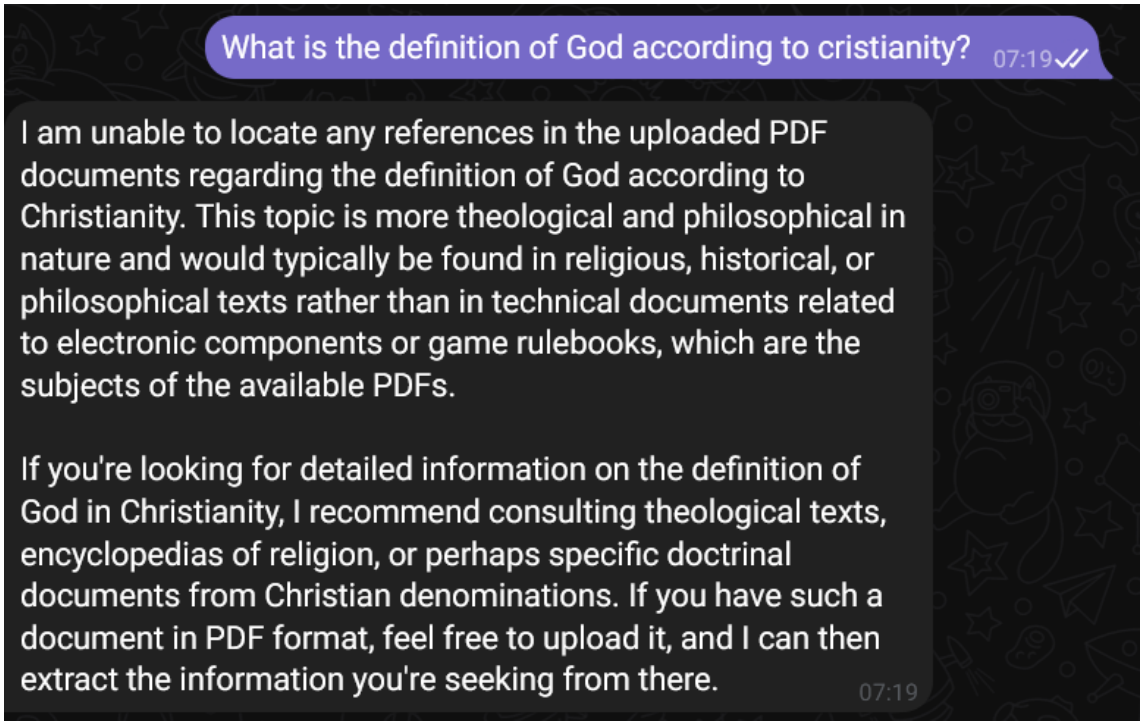
Chapter 13.Appendix E

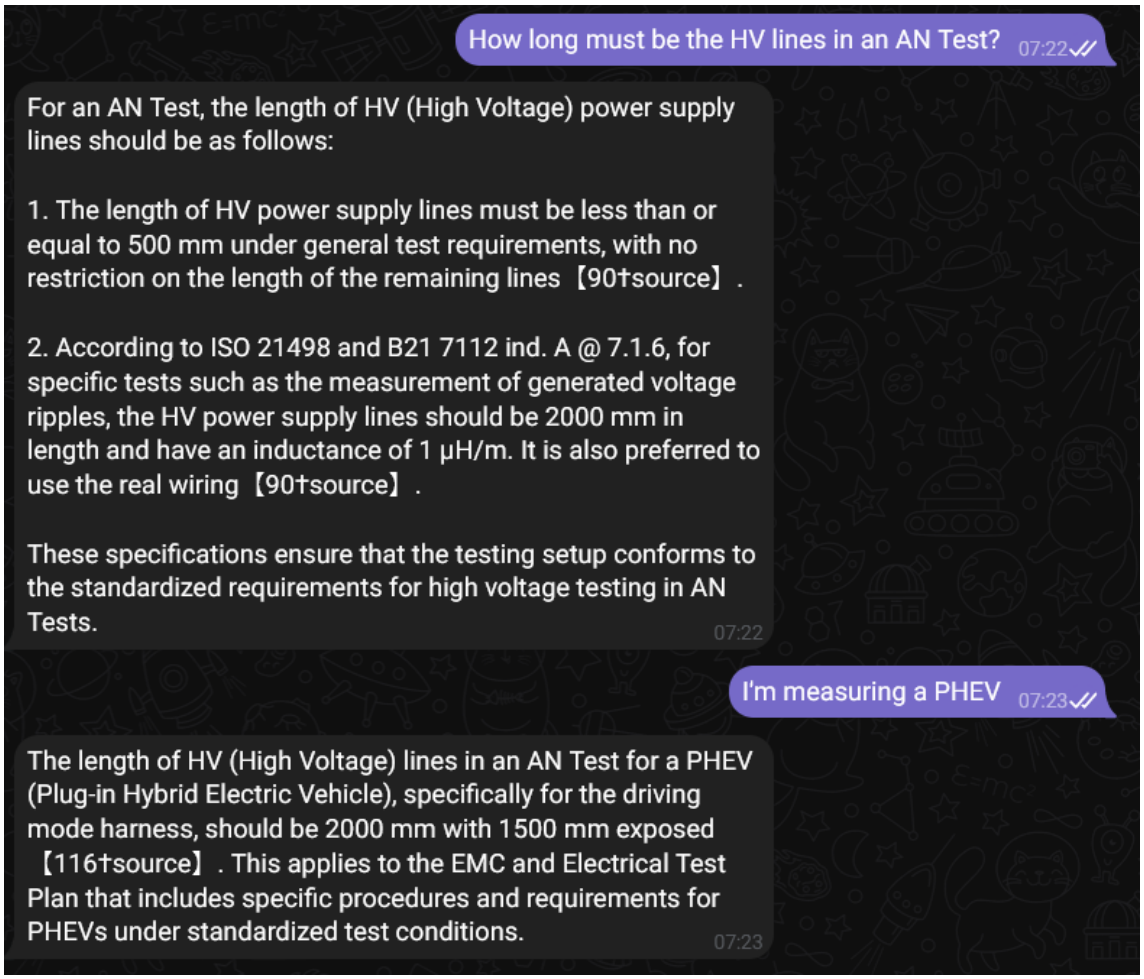


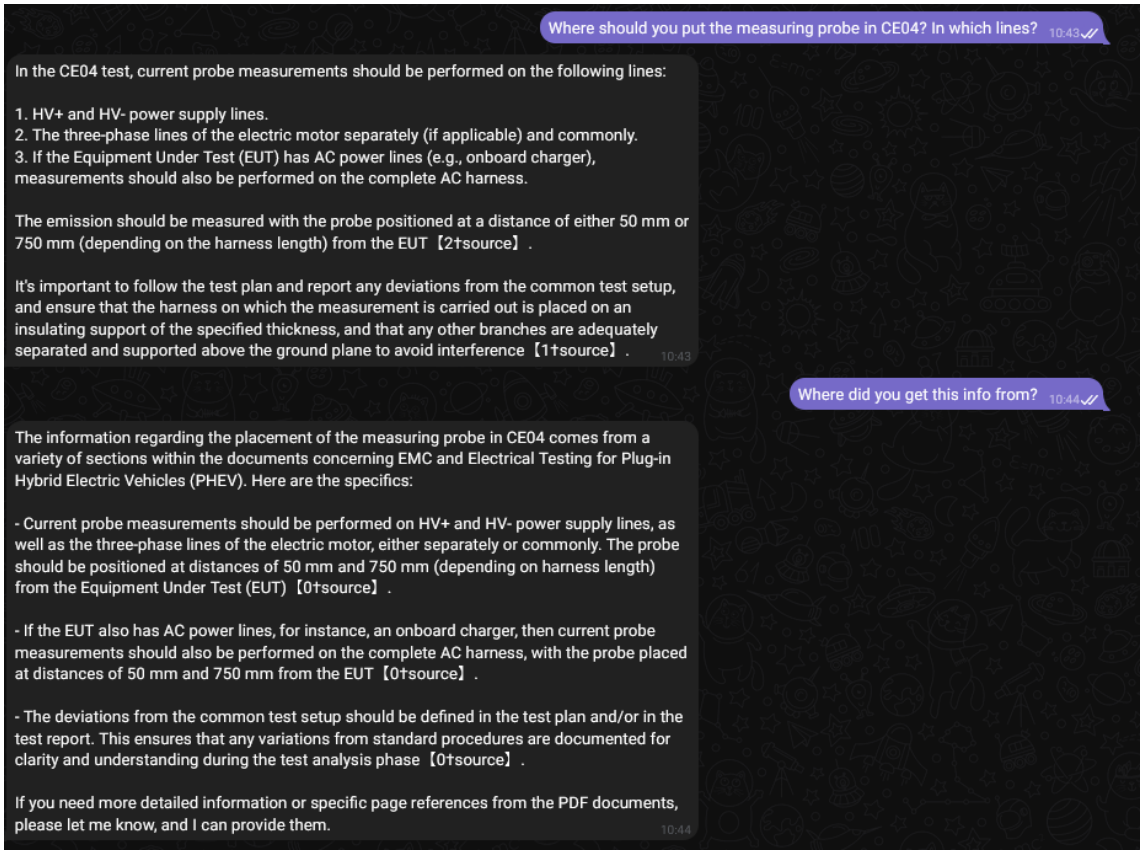


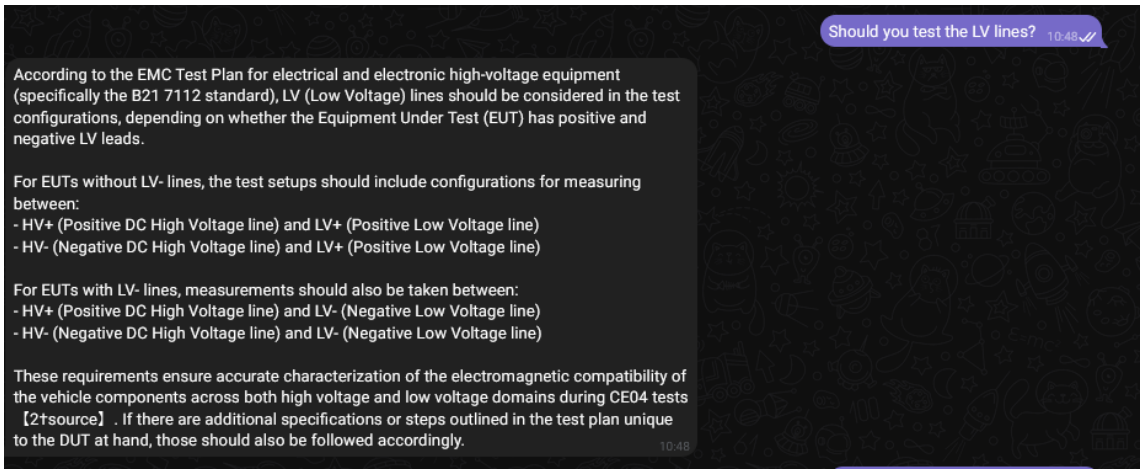

How long must be the HV lines in an AN Test? 07:11
In an AN Test, the requirements for the length of the HV (High Voltage) power supply lines may vary depending on the specific test being performed:
1. In general, the length of HV power supply lines should be less than or equal to 500 mm with no restriction on the length of the remaining lines【91†source】.
2. For specific tests, such as the measurement of generated voltage ripples as mentioned in the ELECTRICAL AND EMC TEST PLAN, the HV power supply lines should be 2000 mm in length and have an inductance of 1 µH/m; it is also recommended to use the real wiring if possible【91†source】.
These specifications are crucial to meet the standards outlined in the test plans and ensure accurate testing of high voltage components.
07:12

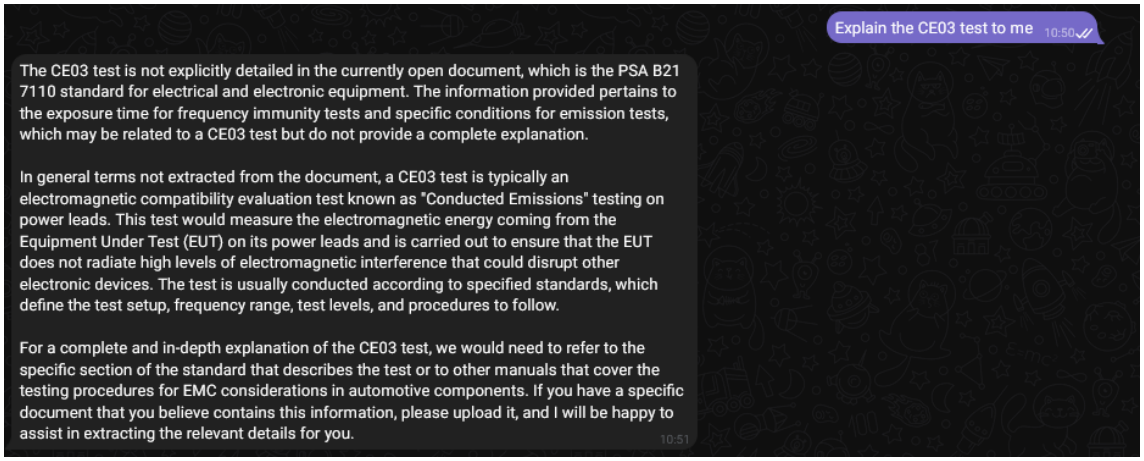

How long must be the HV lines in an AN Test? 07:13
For the AN Test, the length of HV (High Voltage) power supply lines should be as follows:
1. The length of HV power supply lines must be less than or equal to 500 mm. This is under the general test requirements where no restriction is placed on the remaining lines【90†source】.
2. According to ISO 21498 and B21 7112 ind. A @ 7.1.6, for specific tests such as the measurement of generated voltage ripples, the length of HV power supply lines should be 2000 mm (L=1µH/m), and it is preferred to use the real wiring【90†source】.
These lengths are specified to ensure compliance with testing standards and to achieve accurate results in the AN Test.
07:13

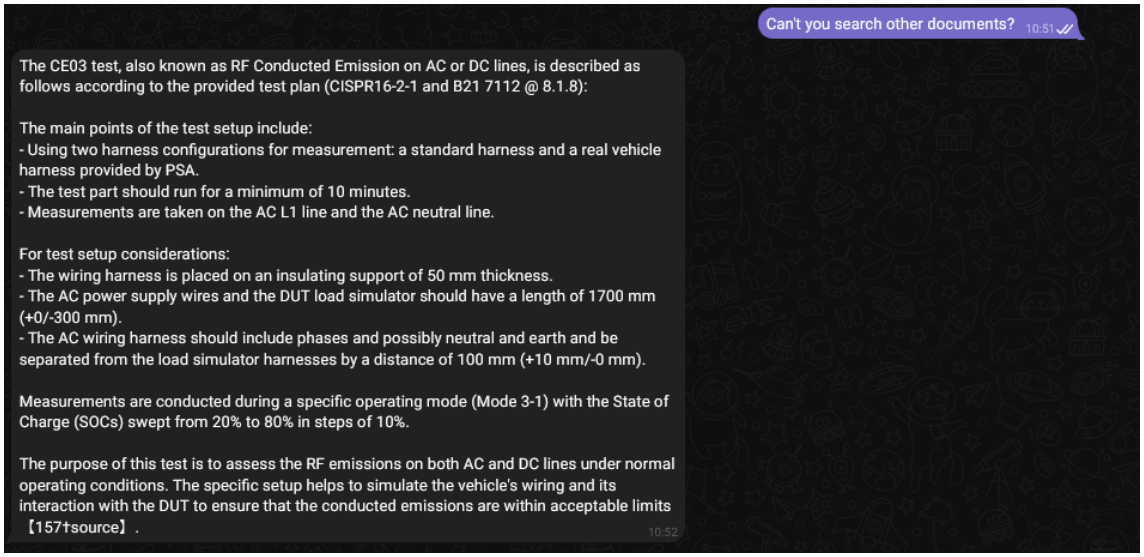

What is the definition of God according to cristianity? 07:19
I am unable to locate any references in the uploaded PDF documents regarding the definition of God according to Christianity. This topic is more theological and philosophical in nature and would typically be found in religious, historical, or philosophical texts rather than in technical documents related to electronic components or game rulebooks, which are the subjects of the available PDFs.
If you're looking for detailed information on the definition of God in Christianity, I recommend consulting theological texts, encyclopedias of religion, or perhaps specific doctrinal documents from Christian denominations. If you have such a document in PDF format, feel free to upload it, and I can then extract the information you're seeking from there. 07:19

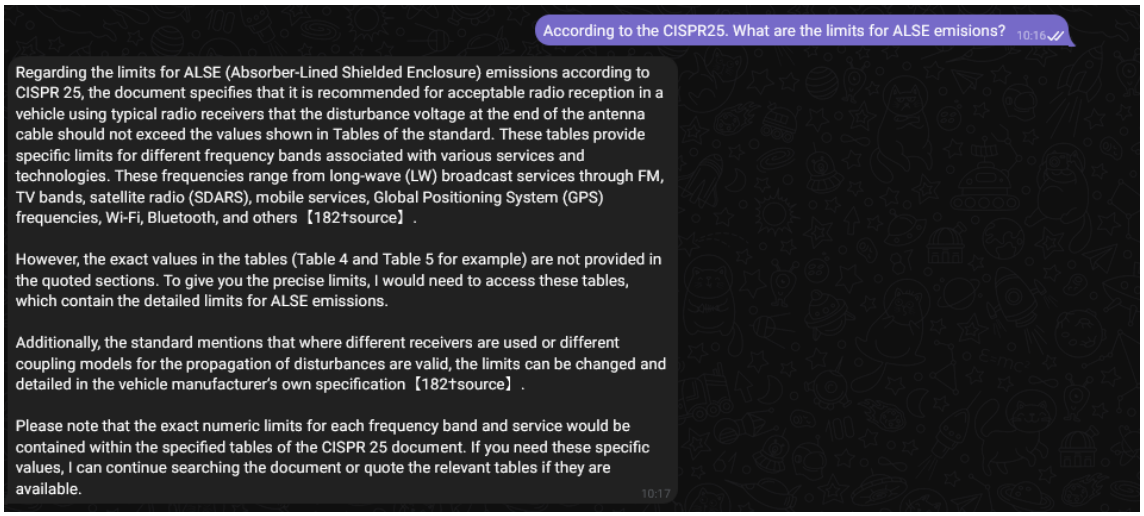

How long must be the HV lines in an AN Test? 07:22
For an AN Test, the length of HV (High Voltage) power supply lines should be as follows:
1. The length of HV power supply lines must be less than or equal to 500 mm under general test requirements, with no restriction on the length of the remaining lines【90†source】.
2. According to ISO 21498 and B21 7112 ind. A @ 7.1.6, for specific tests such as the measurement of generated voltage ripples, the HV power supply lines should be 2000 mm in length and have an inductance of 1 µH/m. It is also preferred to use the real wiring【90†source】.
These specifications ensure that the testing setup conforms to the standardized requirements for high voltage testing in AN Tests. 07:22
I'm measuring a PHEV 07:23
The length of HV (High Voltage) lines in an AN Test for a PHEV (Plug-in Hybrid Electric Vehicle), specifically for the driving mode harness, should be 2000 mm with 1500 mm exposed【116†source】. This applies to the EMC and Electrical Test Plan that includes specific procedures and requirements for PHEVs under standardized test conditions. 07:23

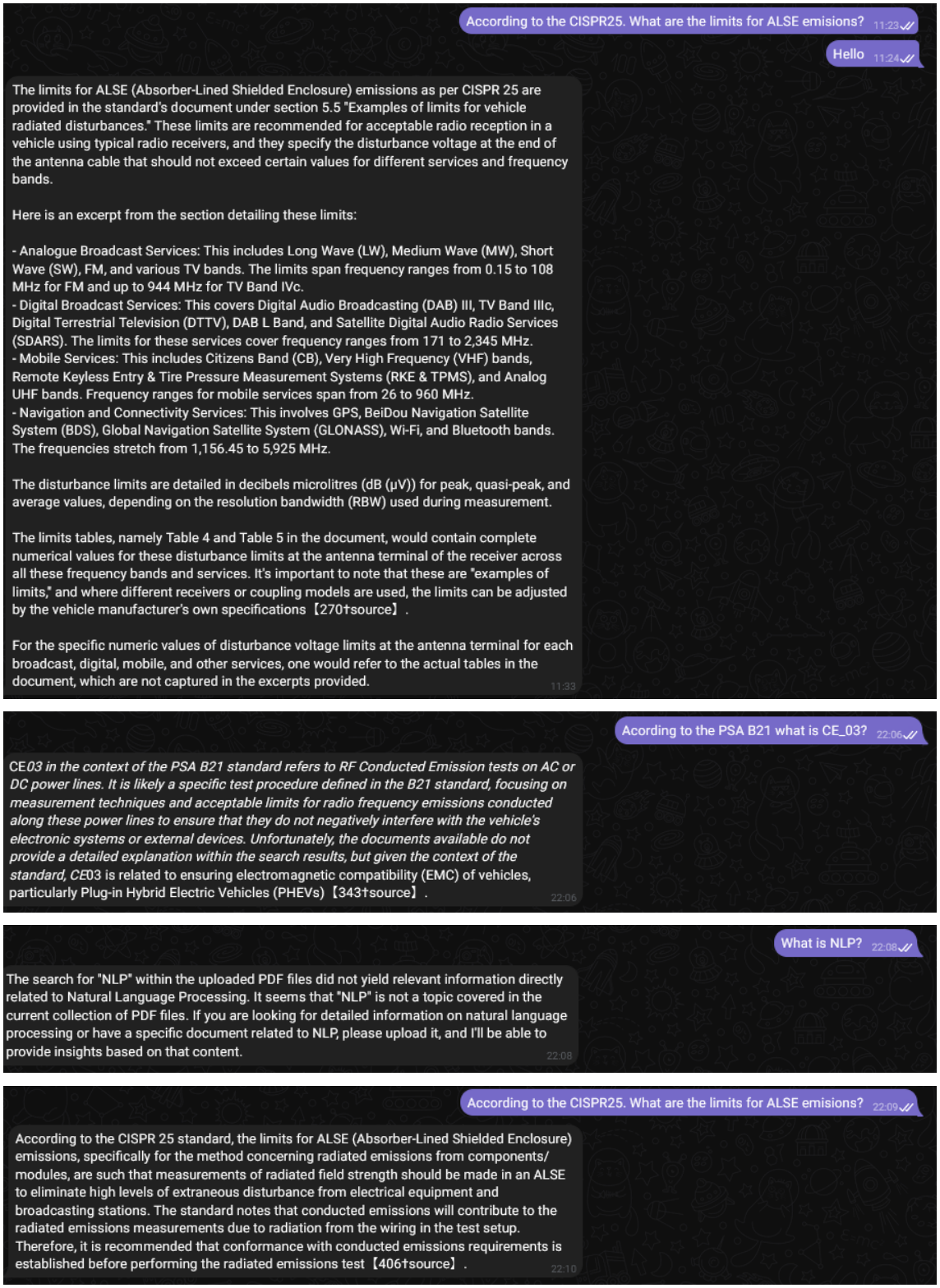

Where should you put the measuring probe in CE04? In which lines? 10:43
In the CE04 test, current probe measurements should be performed on the following lines:
1. HV+ and HV- power supply lines.
2. The three-phase lines of the electric motor separately (if applicable) and commonly.
3. If the Equipment Under Test (EUT) has AC power lines (e.g., onboard charger), measurements should also be performed on the complete AC harness.
The emission should be measured with the probe positioned at a distance of either 50 mm or 750 mm (depending on the harness length) from the EUT【2†source】.
It's important to follow the test plan and report any deviations from the common test setup, and ensure that the harness on which the measurement is carried out is placed on an insulating support of the specified thickness, and that any other branches are adequately separated and supported above the ground plane to avoid interference【1†source】. 10:43
Where did you get this info from? 10:44
The information regarding the placement of the measuring probe in CE04 comes from a variety of sections within the documents concerning EMC and Electrical Testing for Plug-in Hybrid Electric Vehicles (PHEV). Here are the specifics:
- Current probe measurements should be performed on HV+ and HV- power supply lines, as well as the three-phase lines of the electric motor, either separately or commonly. The probe should be positioned at distances of 50 mm and 750 mm (depending on harness length) from the Equipment Under Test (EUT)【0†source】.
- If the EUT also has AC power lines, for instance, an onboard charger, then current probe measurements should also be performed on the complete AC harness, with the probe placed at distances of 50 mm and 750 mm from the EUT【0†source】.
- The deviations from the common test setup should be defined in the test plan and/or in the test report. This ensures that any variations from standard procedures are documented for clarity and understanding during the test analysis phase【0†source】.
If you need more detailed information or specific page references from the PDF documents, please let me know, and I can provide them. 10:44

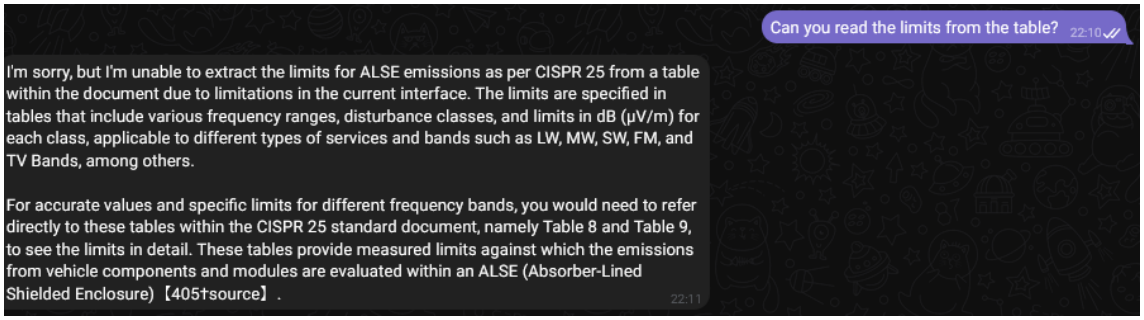

Should you test the LV lines? 10:48
According to the EMC Test Plan for electrical and electronic high-voltage equipment (specifically the B21 7112 standard), LV (Low Voltage) lines should be considered in the test configurations, depending on whether the Equipment Under Test (EUT) has positive and negative LV leads.
For EUTs without LV- lines, the test setups should include configurations for measuring between:
- HV+ (Positive DC High Voltage line) and LV+ (Positive Low Voltage line)
- HV- (Negative DC High Voltage line) and LV+ (Positive Low Voltage line)
For EUTs with LV- lines, measurements should also be taken between:
- HV+ (Positive DC High Voltage line) and LV- (Negative Low Voltage line)
- HV- (Negative DC High Voltage line) and LV- (Negative Low Voltage line)
These requirements ensure accurate characterization of the electromagnetic compatibility of the vehicle components across both high voltage and low voltage domains during CE04 tests【2†source】. If there are additional specifications or steps outlined in the test plan unique to the DUT at hand, those should also be followed accordingly. 10:48


Explain the CE03 test to me 10:50
The CE03 test is not explicitly detailed in the currently open document, which is the PSA B21 7110 standard for electrical and electronic equipment. The information provided pertains to the exposure time for frequency immunity tests and specific conditions for emission tests, which may be related to a CE03 test but do not provide a complete explanation.
In general terms not extracted from the document, a CE03 test is typically an electromagnetic compatibility evaluation test known as "Conducted Emissions" testing on power leads. This test would measure the electromagnetic energy coming from the Equipment Under Test (EUT) on its power leads and is carried out to ensure that the EUT does not radiate high levels of electromagnetic interference that could disrupt other electronic devices. The test is usually conducted according to specified standards, which define the test setup, frequency range, test levels, and procedures to follow.
For a complete and in-depth explanation of the CE03 test, we would need to refer to the specific section of the standard that describes the test or to other manuals that cover the testing procedures for EMC considerations in automotive components. If you have a specific document that you believe contains this information, please upload it, and I will be happy to assist in extracting the relevant details for you. 10:51


Can't you search other documents? 10:51
The CE03 test, also known as RF Conducted Emission on AC or DC lines, is described as follows according to the provided test plan (CISPR16-2-1 and B21 7112 @ 8.1.8):
The main points of the test setup include:
- Using two harness configurations for measurement: a standard harness and a real vehicle harness provided by PSA.
- The test part should run for a minimum of 10 minutes.
- Measurements are taken on the AC L1 line and the AC neutral line.
For test setup considerations:
- The wiring harness is placed on an insulating support of 50 mm thickness.
- The AC power supply wires and the DUT load simulator should have a length of 1700 mm (+0/-300 mm).
- The AC wiring harness should include phases and possibly neutral and earth and be separated from the load simulator harnesses by a distance of 100 mm (+10 mm/-0 mm).
Measurements are conducted during a specific operating mode (Mode 3-1) with the State of Charge (SOCs) swept from 20% to 80% in steps of 10%.
The purpose of this test is to assess the RF emissions on both AC and DC lines under normal operating conditions. The specific setup helps to simulate the vehicle's wiring and its interaction with the DUT to ensure that the conducted emissions are within acceptable limits 【157†source】.
10:52


According to the CISPR25. What are the limits for ALSE emisions? 10:16
Regarding the limits for ALSE (Absorber-Lined Shielded Enclosure) emissions according to CISPR 25, the document specifies that it is recommended for acceptable radio reception in a vehicle using typical radio receivers that the disturbance voltage at the end of the antenna cable should not exceed the values shown in Tables of the standard. These tables provide specific limits for different frequency bands associated with various services and technologies. These frequencies range from long-wave (LW) broadcast services through FM, TV bands, satellite radio (SDARS), mobile services, Global Positioning System (GPS) frequencies, Wi-Fi, Bluetooth, and others 【182†source】.
However, the exact values in the tables (Table 4 and Table 5 for example) are not provided in the quoted sections. To give you the precise limits, I would need to access these tables, which contain the detailed limits for ALSE emissions.
Additionally, the standard mentions that where different receivers are used or different coupling models for the propagation of disturbances are valid, the limits can be changed and detailed in the vehicle manufacturer's own specification 【182†source】.
Please note that the exact numeric limits for each frequency band and service would be contained within the specified tables of the CISPR 25 document. If you need these specific values, I can continue searching the document or quote the relevant tables if they are available.
10:17


According to the CISPR25. What are the limits for ALSE emisions? 11:23
Hello 11:24
The limits for ALSE (Absorber-Lined Shielded Enclosure) emissions as per CISPR 25 are provided in the standard's document under section 5.5 "Examples of limits for vehicle radiated disturbances." These limits are recommended for acceptable radio reception in a vehicle using typical radio receivers, and they specify the disturbance voltage at the end of the antenna cable that should not exceed certain values for different services and frequency bands.
Here is an excerpt from the section detailing these limits:
- Analogue Broadcast Services: This includes Long Wave (LW), Medium Wave (MW), Short Wave (SW), FM, and various TV bands. The limits span frequency ranges from 0.15 to 108 MHz for FM and up to 944 MHz for TV Band IVc.
- Digital Broadcast Services: This covers Digital Audio Broadcasting (DAB) III, TV Band IIIc, Digital Terrestrial Television (DTTV), DAB L Band, and Satellite Digital Audio Radio Services (SDARS). The limits for these services cover frequency ranges from 171 to 2,345 MHz.
- Mobile Services: This includes Citizens Band (CB), Very High Frequency (VHF) bands, Remote Keyless Entry & Tire Pressure Measurement Systems (RKE & TPMS), and Analog UHF bands. Frequency ranges for mobile services span from 26 to 960 MHz.
- Navigation and Connectivity Services: This involves GPS, BeiDou Navigation Satellite System (BDS), Global Navigation Satellite System (GLONASS), Wi-Fi, and Bluetooth bands. The frequencies stretch from 1,156.45 to 5,925 MHz.
The disturbance limits are detailed in decibels microlitres (dB (µV)) for peak, quasi-peak, and average values, depending on the resolution bandwidth (RBW) used during measurement.
The limits tables, namely Table 4 and Table 5 in the document, would contain complete numerical values for these disturbance limits at the antenna terminal of the receiver across all these frequency bands and services. It's important to note that these are "examples of limits," and where different receivers or coupling models are used, the limits can be adjusted by the vehicle manufacturer's own specifications【270†source】.
For the specific numeric values of disturbance voltage limits at the antenna terminal for each broadcast, digital, mobile, and other services, one would refer to the actual tables in the document, which are not captured in the excerpts provided. 11:33
Acording to the PSA B21 what is CE_03? 22:06
CE03 in the context of the PSA B21 standard refers to RF Conducted Emission tests on AC or DC power lines. It is likely a specific test procedure defined in the B21 standard, focusing on measurement techniques and acceptable limits for radio frequency emissions conducted along these power lines to ensure that they do not negatively interfere with the vehicle's electronic systems or external devices. Unfortunately, the documents available do not provide a detailed explanation within the search results, but given the context of the standard, CE03 is related to ensuring electromagnetic compatibility (EMC) of vehicles, particularly Plug-in Hybrid Electric Vehicles (PHEVs)【343†source】. 22:06
What is NLP? 22:08
The search for "NLP" within the uploaded PDF files did not yield relevant information directly related to Natural Language Processing. It seems that "NLP" is not a topic covered in the current collection of PDF files. If you are looking for detailed information on natural language processing or have a specific document related to NLP, please upload it, and I'll be able to provide insights based on that content. 22:08
According to the CISPR25. What are the limits for ALSE emisions? 22:09
According to the CISPR 25 standard, the limits for ALSE (Absorber-Lined Shielded Enclosure) emissions, specifically for the method concerning radiated emissions from components/modules, are such that measurements of radiated field strength should be made in an ALSE to eliminate high levels of extraneous disturbance from electrical equipment and broadcasting stations. The standard notes that conducted emissions will contribute to the radiated emissions measurements due to radiation from the wiring in the test setup. Therefore, it is recommended that conformance with conducted emissions requirements is established before performing the radiated emissions test【406†source】. 22:10


Can you read the limits from the table? 22:10
I'm sorry, but I'm unable to extract the limits for ALSE emissions as per CISPR 25 from a table within the document due to limitations in the current interface. The limits are specified in tables that include various frequency ranges, disturbance classes, and limits in dB (µV/m) for each class, applicable to different types of services and bands such as LW, MW, SW, FM, and TV Bands, among others.
For accurate values and specific limits for different frequency bands, you would need to refer directly to these tables within the CISPR 25 standard document, namely Table 8 and Table 9, to see the limits in detail. These tables provide measured limits against which the emissions from vehicle components and modules are evaluated within an ALSE (Absorber-Lined Shielded Enclosure)【405†source】.
22:11