



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Transforming Keywords into Conversation: Implementation of a
Chatbot-Driven System for Speech Impaired Individuals“

verfasst von / submitted by

Lucas Munz, B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

UA 066 926

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Masterstudium Wirtschaftsinformatik

Betreut von / Supervisor:

Univ.-Prof. Dr.-Ing. Moritz Grosse-Wentrup

Acknowledgements

In retrospect, I couldn't have chosen a better topic for my master's thesis. The subject around chatbots and artificial intelligence, especially in the medical environment, captured me deeply and it was a lot of pleasure for me to do in-depth research and experimentation on these topics. It is hard to believe that at the beginning of the thesis, most intelligent chatbots had not yet been published and language models on which this thesis is based were largely unknown. The enormous development in the field of Natural Language Processing, as well as the growing influence of chatbots on our society, reinforces the importance of this work.

However, it would have been hardly possible to do such a thesis without any support. It has been a pleasure carrying out this work in the last couple of months and I want to express my sincere gratitude to Univ.-Prof. Dr.-Ing. Moritz Grosse-Wentrup for his continuous support, listening intently to my ideas and offering constructive advice.

Additionally, I want to thank every single person in the neuroscience research group who has ever provided me with fresh insights through their questions, recommendations, and viewpoints.

Moreover, I am deeply appreciative of the overwhelming support of Annika Fricker, for the stimulating discussions and reading through my sheer endless amount of drafts several times.

Finally, I want to express my gratitude to all of my family and friends for their support and thoughtfulness while I worked on this thesis.

Abstract

In artificial intelligence, the latest language models are almost on a par with human language skills. Utilizing these technologies could enhance the language skills of individuals with speech impairments closer to human levels. Within the scope of the thesis, different approaches are discussed and evaluated to solve such a problem. In the course of this work, it is demonstrated that there are gaps, especially in the area of keyword-driven chatbots. According to current knowledge on the state of research, there are no systems at present that convert the few keywords that people with speech impairments can still utter into comprehensive sentences in conversations. This work aims to enable these mentioned patients to fully participate in a conversation through a chatbot and ultimately giving them a voice. For this purpose, several strategies of implementing such a system are compared and based on that, various datasets are created to fine-tune one of the largest available language models. The resulting system, called DialoKEY, can create sentences derived from keywords in conversations. In a subsequent experiment, the performance of the different versions of DialoKEY is evaluated. Based on the implemented models and results of the experiment, the best way to implement such a system is to fine-tune a large language model with a high-quality data set. The detailed evaluation of the experimental outcomes confirms that the proposed system provides results that are nearly on par with human language. DialoKEY therefore takes one step further in enabling patients with speech impairments to fully participate in a conversation again.

Kurzfassung

In der künstlichen Intelligenz sind die neuesten Sprachmodelle den menschlichen Sprachkenntnissen fast ebenbürtig. Warum also nicht diese Technologien nutzen, um die Sprache von Menschen mit Sprachbehinderungen wieder auf das menschliche Niveau zu bringen? Nach aktuellem Kenntnisstand der Forschung gibt es derzeit keine Systeme, die anhand weniger Schlüsselwörter, die Menschen mit diversen Sprachbehinderungen äußern können, in umfassende Sätze umwandeln. Im Rahmen dieser Arbeit werden verschiedene Ansätze diskutiert und bewertet, um der Forschungsfrage nachzugehen:

Welches ist die beste Strategie, um ein System zu entwickeln, das aus Schlüsselwörtern ganze Sätze bildet und so sprachbehinderten Menschen eine Stimme gibt?

Zunächst werden die theoretischen Grundlagen der Anwendungsfälle sowie die neuesten Entwicklungen auf dem Gebiet Natural Language Processing diskutiert. Anschließend werden verwandte Verfahren und ähnliche Projekte im Bereich der Chatbots betrachtet und es wird untersucht, ob es bereits erste Ansätze gibt, auf denen aufgebaut werden kann.

Ferner werden verschiedene technologische Ansätze zur Systemimplementierung verglichen, mit dem Ziel, die optimale Lösung in die Praxis umzusetzen. Dazu werden Datensätze mit unterschiedlichen Charakteristiken generiert und einzelne Sprachmodelle damit trainiert. Um die Implementierung der verschiedenen Modelle zu evaluieren, wird ein ausführliches Experiment konzipiert und durchgeführt. Der Schwerpunkt liegt dabei, neben einigen automatischen Bewertungsmetriken, auf menschlichen Bewertungen. Diese bewerten die Leistungsfähigkeit der Modelle in simulierten Szenarien.

Die erhaltenen Evaluationsdaten des Experiments werden einer detaillierten und fundierten Auswertung unterzogen. Diese Auswertung beantwortet letztlich die Forschungsfrage und liefert den Nährboden für zukünftige Projekte mit neuen Erkenntnissen.

Nicht nur die Ergebnisse der Evaluation, sondern auch die neu generierten Daten des Experiments bieten Raum für weitere Arbeiten. Im Laufe der Arbeit kristallisieren sich immer wieder Erweiterungen und Folgethemen heraus.

Zusammenfassend lässt sich die folgende Arbeit in drei Teile gliedern: Eine Literaturrecherche, die Implementierung und das Experiment zur Evaluierung der implementierten Lösung. Letztlich zielen die Beiträge und Kapitel darauf ab, sprachbehinderte Menschen zu befähigen, das menschliche Bedürfnis nach Kommunikation, Zuneigung und sozialer Zugehörigkeit zu befriedigen. Auf Basis der implementierten Modelle und Ergebnisse des Experiments wird die Forschungsfrage final beantwortet. Die beste Methode zur Umsetzung eines solchen Systems ist das fine-tuning eines großen Sprachmodells mit einem qualitativ hochwertigen Datensatz.

Contents

Acknowledgements	i
Abstract	iii
Kurzfassung	v
List of Tables	xi
List of Figures	xiii
1. Introduction	1
1.1. Contributions	2
2. Use Cases for DialoKEY	5
2.1. Medical Use Cases	5
2.1.1. Aphasia	5
2.1.2. Dysarthria	7
2.2. Language Learning	7
3. Natural Language Processing and Generation	9
3.1. Recurrent Neural Networks	9
3.2. Transformers	10
3.2.1. Architecture	11
3.2.2. Self-Attention	11
3.2.3. Training and Fine-Tuning	13
3.3. Large Language Model Examples	14
4. Chatbots	15
4.1. Extractive QA	16
4.2. Generative QA	16
4.3. Related Chatbot Systems	16
4.3.1. Open Generative QA and LLMs	16
4.3.2. Tone-Controlled Answers	17
4.3.3. Emotion-Based Answers	17
4.3.4. Keyword Controlled Answers	17
4.3.5. Alternative Augmentative Communication	18
4.4. Intermediate Conclusion	18

5. Implementation	19
5.1. Procedure and Comparison of Methods	19
5.1.1. Keyword Extraction and Insertion	20
5.1.2. Utilizing Specialized QA-Systems	21
5.1.3. Utilizing a RNN	22
5.1.4. Utilizing a LLM	23
5.2. Training Data	29
5.2.1. Conversation Dataset as a Basis	30
5.2.2. Data Pre-Processing	33
5.2.3. Different Models for the Implementation	37
5.3. Fine-Tuning	41
5.3.1. Hyperparameters	42
5.3.2. How Fine-Tuning Works With OpenAI	44
5.4. Reproducibility	45
6. Experiment	49
6.1. Core Aspects	49
6.2. User Guidance	52
6.3. Use Cases	54
6.4. Technology Stack	57
6.4.1. Gradio	57
6.4.2. Hosting Technologies	57
6.4.3. Database	58
6.4.4. Language Model	58
6.5. Functionality	60
6.5.1. Gradio State	60
6.5.2. Model Comparison	61
6.5.3. Logging	61
6.5.4. Robustness	62
6.5.5. Attention Check and Fraud Detection	63
6.6. Implementation of Use Cases	63
6.7. Pre-Testing Interviews and Optimization	65
6.8. Participants	68
6.8.1. Family, Friends, and Survey Circles	68
6.8.2. Amazon Mechanical Turk	69
6.9. Results Cleaning	70
6.10. Reproducibility	71
6.10.1. Experiment	71
6.10.2. Results-Cleaning	72
7. Evaluation	73
7.1. Human Evaluation	73
7.1.1. Rating	74
7.1.2. Statistical Significance	76

7.1.3. Detailed Rating	77
7.1.4. Adjusted Rating	80
7.2. Automatic Evaluation	82
7.2.1. Different Metrics	82
7.2.2. Results of Automatic Evaluation	84
7.3. Further Evaluation Metrics	86
7.3.1. Length of the Sentences	86
7.3.2. Keywords Present in Sentences	87
7.3.3. Number of Keywords and Rating	88
7.3.4. History Length and Rating	89
7.3.5. Rating Depending on the Scenario	90
7.4. Cost Factor	91
7.5. Latency	92
7.6. Which Dataset Performed Best?	92
7.7. Evaluation of Keyword Extraction Tools	93
7.8. Reproducibility	95
8. Conclusion and Discussion	97
8.1. Results	97
8.2. Future Work and Limitations	98
8.3. Discussion	100
Bibliography	103
Appendices	115
A. Appendix	117
A.1. Keyword Extractor Settings	117
A.2. Topics in Conversation Dataset	119
A.3. Use Case Instructions	120
A.4. Settings Model Comparison	121
A.5. Next Tasks Experiment	123
A.6. Pre-Testing Interview	123
A.7. Mechanical Turk Best Practices, Explanations, and Cost	124
A.8. Distribution of Human Ratings	126
A.9. ANOVA Results	126
A.10. Distribution of Detailed Ratings	126
A.11. Standard Deviations of Automatic Evaluation	127
A.12. Standard Deviations of History Length and Rating	127
A.13. Keywords and Rating	127
A.13.1. Standard Deviations	127
A.13.2. Amount of Ratings	128
A.14. Standard Deviations of Ratings Compared to the History Length	128

Contents

A.15.Scenario and Rating	128
A.15.1.Standard Deviations	128
A.15.2.Detailed Scenarios	129
A.16Automatic Evaluation of Perfect Sentences	129
A.17.Standard Deviations of Keyword Extraction Tool Evaluations	130
A.18.Hyperparameters	131
A.18.1.Epoch Size	131
A.18.2.Batch Size	132
A.18.3.Learning Rate Multiplier	133
A.19.Codebase	134
A.19.1.Experiment	134
A.19.2.Fine-Tuning and Evaluation	135
A.19.3.Training data for the fine-tuned models	136
A.19.4.Topic Detection	136
A.19.5.Creating a “Perfect” Dataset	136

List of Tables

5.1. Comparison of different language models	27
5.2. Overview of different fine-tune models	37
5.3. Cost comparison of different GPT-3 base models	42
5.4. Overview of characteristics of the created fine-tunes	45
6.1. Overview of different scenarios including first question and description . .	56
6.2. Statistics of the responses received from the experiment	71
7.1. Explanation of different rating categories with examples	74
7.2. Average human ratings for different models	75
7.3. Explanation of different detailed-rating categories	78
7.4. Adjustment factor with regard to detailed ratings	81
7.5. Adjusted rating in comparison to normal ratings	81
7.6. Results of automatic evaluation metrics	84
7.7. Evaluation of sentence and keyword length [in words]	86
7.8. Comparison of models if keywords are present in created sentences	88
7.9. Ratings in comparison to the number of keywords	88
7.10. Ratings in comparison to the length of the history	90
7.11. Ratings in comparison to the scenario type	90
7.12. Latency of the models [in seconds]	92
7.13. Means of the evaluation of keyword extraction tools (KWs = keywords) .	94
A.1. Use case instructions for GPT-3	120
A.2. Distribution of human ratings for different models	126
A.3. ANOVA test results between all models	126
A.4. Distribution of human detailed-ratings for all models	126
A.5. Standard deviations of automatic evaluation metrics	127
A.6. Standard deviations of ratings depending on the history length	127
A.7. Standard deviations for ratings depending on the number of keywords . .	127
A.8. Amount of different ratings per number of keywords	128
A.9. Standard deviations of human ratings in comparison to the history length	128
A.10. Standard deviations for ratings depending on the scenario type	128
A.11. Average ratings for the models depending on the exact scenario	129
A.12. Standard deviations for the models depending on the exact scenario . . .	129
A.13. Averages of automatic evaluation of only perfect human ratings	129
A.14. Standard deviations of automatic evaluation of only perfect ratings	130
A.15. Standard deviations of means of keyword tools evaluation	130

List of Figures

1.1. Illustration on how the proposed system (DialoKEY) should work	2
3.1. Self-attention explained visually [52]	12
5.1. Illustration of various implementation methods	20
5.2. Different methods of using a LLM (<i>based on [21]</i>)	24
5.3. Example of a conversation	31
5.4. Process of transformation from the conversations to the dataset	32
5.5. Example of a conversation and the resulting dataset	33
6.1. Interface design of the experiment	50
6.2. Rating and detailed rating of an answer in the experiment	51
6.3. Step by step explanation before the start of the experiment	53
6.4. Progress bar and skip-button in the experiment	54
6.5. Technology stack used for DialoKEY	57
6.6. Example error message in the experiment	62
6.7. Amazon Mechanical Turk layout	69
7.1. Distribution of human ratings for all models	75
7.2. Tukey HSD test with all models	77
7.3. Distribution of detailed ratings for all models	79
7.4. Correlation matrix of automatic evaluations and human ratings	85
A.1. World cloud of different topics of the conversation data	119
A.2. GPT-J and GPT-Neo settings	121
A.3. OpenAI Settings	122
A.4. All next task instructions in the experiment	123
A.5. Pre-Testing interview guideline	123
A.6. Amazon Mechanical Turk settings	125
A.7. W&B evaluations of the epoch size and losses in training	131
A.8. W&B evaluations of the batch size and losses in training	132
A.9. W&B evaluations of the learning rate multiplier and losses in training . .	133
A.10. Folder structure of codebase	135

1. Introduction

One out of three U.S. patients who suffer from a stroke subsequently has aphasia – a speech impairment [31]. This means that they frequently communicate in fragments or single words, are unable to form complete sentences and struggle to convey their true intentions. The impairment of speech is a “devastating condition that interferes with almost every social activity and interaction” [50, p. 183]. In addition, other medical conditions as well as nonmedical use-cases also fall into this pattern of uttering single words instead of sentences. At the same time, the market for artificial intelligence (AI) in the healthcare sector is growing, and new use cases are increasingly emerging with the aim of significantly transforming this sector [37, 9]. A relevant part of this is Natural Language Processing (NLP). Already since the 1950s, one of the biggest goals of AI / NLP has been to develop an intelligent dialogue system that interacts with humans in a coherent and engaging way [70, 54]. For people with a speech disorder, such as aphasia, this potentially means participating and expressing themselves in everyday life and conversation.

Until recently, computers, unlike humans, suffered from difficulties handling long-term dependencies between words in a sequence of input. However, the latest developments make this possible now [134]. In this context, the Transformers architecture plays an essential role. Utilizing Transformers, connections between individual components of sequences can now be much better understood by computers than with previous technology. Among other things, this has led to an enormous increase in the use of chatbots recently, not only in healthcare, but also in other social and business related areas.

However, despite all the enormous recent developments in the field of NLP and chatbots, there is not yet a way to enable people with speech impairments to form adequate sentences and to engage in conversations. Hence, the research question posed in this thesis is:

What is the best strategy for designing a system that forms full sentences out of uttered keywords, thus giving a voice to speech impaired people?

The system to be designed aims to enable a conversation between two people. It should support the conversation partner with a speech impairment to be fully engaged in the conversation, i.e. to give answers and to ask questions. In the following, this system is called DialoKEY, a word composition of dialogue via keywords.

1. Introduction

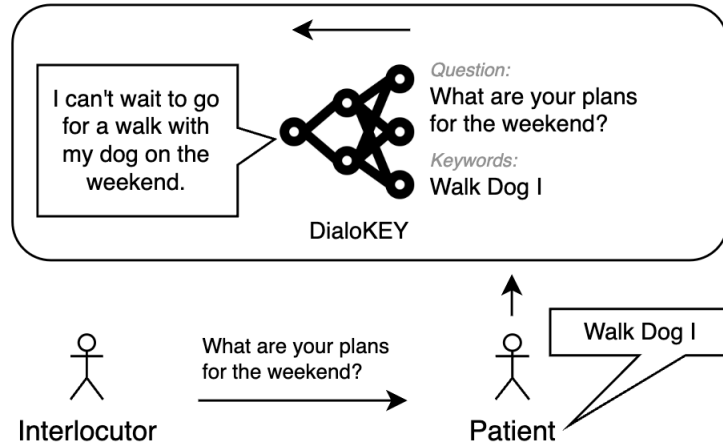


Figure 1.1.: Illustration on how the proposed system (DialoKEY) should work

Fig. 1.1 illustrates the purpose of DialoKEY. The intention of what a speech impaired human being wants to say is characterized by a few words such as “Walk, Dog, I”. The chatbot captures this intention alongside a potential question which is asked, for instance “What are your plans for the weekend?”. The system is required to transcribe these inputs into: “I can’t wait to go for a walk with my dog on the weekend”.

1.1. Contributions

In order to meet the requirements of the proposed system and answer the research question, this thesis produces the following contributions, which are addressed in chronological order in the next chapters.

Related Work and Theoretical Foundations First, the theoretical foundations of the use cases (chapter 2) and the latest developments in the field of NLP (chapter 3) are discussed. Then, related work and similar projects in the field of chatbots are examined, and it is investigated whether there exist already inspirational approaches to build up on (chapter 4).

Implementation of the System and Creation of the Dataset Based on this, chapter 5 addresses the implementation of the proposed system. Various system implementation approaches are compared with the objective to put into practise the optimal solution (section 5.1). To facilitate this, multiple datasets with different characteristics are generated to train the individual models (section 5.2). Not only the generated datasets, but also the algorithms to create such a dataset are an important contribution. This enables to construct new datasets from existing or new conversation data and to fine-tune other language models.

Extensive Experiment To evaluate the implementation of the different models, an in-depth experiment is designed and conducted in chapter 6. The focus lies on human evaluations, which assess the performance of the models in simulated scenarios, alongside multiple automatic evaluation metrics. The designed experiment can be used for further applications due to its sophisticated design and proper user guidance, both of which have been optimized in several iterations from users feedback. Moreover, it can be utilised for use cases in NLP beyond the field of keyword-controlled chatbots, with minor modifications.

Evaluation and Basis for New Projects In these chapters, the generated data of the experiment is subject to a detailed and well-founded evaluation (chapter 7). The evaluation ultimately answers the research question and provides the breeding ground for future projects with new insights.

Future Work and Conclusion Not only the findings of the evaluation and the newly generated data of the experiment offer scope for further work. During the course of the work, extensions and subsequent topics crystallize repeatedly. The summary of this work, the description of the extensions and a final discussion conclude this thesis with chapter 8.

In conclusion, the following work can be divided into three parts: a survey, the implementation of DialoKEY and the experiment to examine the implemented solution. Ultimately, the contributions and chapters aim to empower speech impaired people, to satisfy the human need for communication, affection and social belonging [70].

2. Use Cases for DialoKEY

The chatbot can help a wide variety of people to communicate with their fellow human beings, even if their actual abilities would not allow it. Patients with various diseases have been identified, that impairs their ability to speak, but the subsequent sections also elaborate on further possible applications. This chapter provides an outline of these use cases and briefly introduces the theoretical background of each.

2.1. Medical Use Cases

Applicable patients with speech impairments can be divided into two different categories, whereas patients with aphasia are the main application of the system. However, in addition to that, individuals suffering from dysarthria were also identified as a target group for the system.

2.1.1. Aphasia

Aphasia is a condition that results in a decrease in the ability to communicate. It is an impairment that can affect production as well as comprehension of speech, reading, and writing.

While the exact cause of aphasia is unknown, it is most likely due to damage to the language areas of the brain. Typically, in the left hemisphere of the brain, aphasia often occurs from a stroke or head injury (such as a severe blow to the head, brain tumours, brain infections). It can also develop slowly from a tumour or a progressive neural disease [35, 51, 96]. The more severe the injury, the more severe the aphasia [51].

Most people with aphasia especially have trouble with speaking. They may speak in short phrases or sentences and have difficulty finding the right word. They might also have trouble understanding the speech of others, as well as reading and writing. Besides linguistic consequences, aphasia also has neurological, neuropsychological and psychosocial consequences. However, this paper will focus on the linguistic barriers as consequences [102].

Aphasia can have a profound impact on an individual's life. It can severely impair an individual's ability to communicate, interact with others, and perform daily activities. This speech impairment can't be cured, but speech-language therapy can help people with aphasia to utter various words [50].

Patients with aphasia are unable to accurately convert a series of non-verbal representations into the organization that resembles language. This means that their thoughts cannot be translated into sentences and language.

2. Use Cases for DialoKEY

Vice versa, the process of generation of thoughts might also be damaged when presented with spoken or written sentences [35]. The brain damage does not result in a complete loss of speech, but can disrupt the ability to produce or understand language in a manner explained above. Which aspect is more affected depends on where exactly the brain damage occurred [17].

According to the national aphasia association, one million people in the USA have aphasia, and 100,000 each year are acquiring it. Approximately 30-40 % of all strokes result in aphasia [51].

There are different types of aphasia, which can be grouped into two broad categories, differentiating in their impact on language impairment [51]: fluent and non-fluent. The different types of aphasia mostly depend on which area of the brain is damaged.

As a representative of the two categories, the following sections examine the two most common types of aphasia in each category and discuss them in detail. Finally, I will conclude that certain types of aphasia are eligible for DialoKEY.

Fluent Fluent aphasia is often the result of damage to the temporal lobe of the brain and can cause individuals to utter long, meaningless sentences. A common type of this category, especially in the western world, is Wernicke’s aphasia [51]. It is typically caused by damage to the posterior sector of the auditory-association cortex [35]. The production of speech is still functional for people with Wernicke’s aphasia, which means the speech is effortless and fluent. However, it contains frequent errors in word choices, irrelevant words are intruding and the phonemic order of a word is frequently mixed up. A phoneme is the sound of a block of speech, with multiple phonemes are making up a word (saying *pymarid* instead of *pyramid* resembles such an error). Patients with Wernicke’s cannot repeat sentences or phrases [35]. This often leads to the creation of new words and sentences that do not cohere [1]. Patients are typically unaware of their mistakes. An example of a sentence produced by a patient is “you know that smoodle pinkered and that I want to get him round and take care of him like you want before” [96]. This makes it difficult to grasp the meaning of a sentence as an interlocutor [35]. Comprehension of spoken words, reading and writing are also impaired by Wernicke’s aphasia [1, 12, 96]. Due to the difficulty of building meaningful sentences in their temporal lobe, the types of fluent aphasia cannot be compensated with the chatbot.

Non-fluent Non-fluent aphasia is often the result of damage to the frontal lobe of the brain, and leads to difficulties translating thoughts into spoken language. The most common type in this category is Broca’s aphasia, in which the Broca’s area of the brain and most commonly its surroundings are damaged [35]. Individuals with Broca’s aphasia often have difficulty speaking in complete sentences, and may instead speak in short phrases or individual words. The language tends to be “agrammatical”, with limited vocabulary and often less than four words [12]. This means that these patients regularly omit “unnecessary” words such as conjunctions or prepositions to reduce the effort of speaking [17]. They are therefore unable to organize the structure of a sentence to be syntactical correct [35].

At the same time, the sentences of these patients tend to contain words that are rich in content and important. For instance, a sentence of a patient might be: “Walk Dog”, meaning “I want to walk with the dog”. Although individuals with non-fluent aphasia regularly have difficulty speaking, they usually have no trouble understanding speech and still have limited abilities in writing and reading [1, 96]. Compared to Broca, patients diagnosed with Transcortical motor aphasia can repeat sentences. The ability to form meaningful words is comparable to Broca type.

To conclude, Broca and Transcortical motor are the main types of aphasia for the use of DialoKEY, since patients are non-fluent but able to comprehend spoken messages and build meaningful words. Other types of non-fluent aphasia are characterized by the extreme limited ability in comprehension as well as expression of speech [35] and therefore patients cannot participate in conversations, even through the proposed chatbot.

2.1.2. Dysarthria

In addition to aphasia, there is another medical use case for the chatbot: Dysarthria. Dysarthria can be the result of a neurological injury that can be caused by diseases such as Parkinson’s, Multiple Sclerosis, or Huntington’s disease. Patients with dysarthria have difficulty speaking due to weak, slow or uncoordinated muscle control. Vocabulary, grammatical knowledge and comprehension are retained in dysarthria. Dysarthria can have further effects than the speech ability, which I will not explain in detail. Moreover, as with aphasia, there are different types and degrees of the disease. Based on the symptoms mentioned, dysarthria is another medical use case for the chatbot. However, since aphasia is the main case in this paper, the types, and degrees of dysarthria will not be further explained but can be the subject of further research [46, 124].

2.2. Language Learning

For learning a new language, the chatbot can also be utilized. Learning a new language often starts with learning single keywords without grammar and syntax. Similar to patients with Broca’s aphasia, learners can build keywords, but lacking conjunctions and prepositions to build sentences [133]. This makes it difficult to say or write complete sentences or even a complete text. Through the chatbot, learners can use only headwords to have a comprehensive sentence written. Furthermore, similar to Broca’s aphasia, even if you can’t speak a new language fluently yet, you typically understand the majority of it. Language learning thanks to chatbots could have numerous advantages: “[...] student enjoyment, decreased language anxiety, opportunities for endless repetitions and multimodal capabilities” [15, p. 3] [67].

3. Natural Language Processing and Generation

One of the largest topics in computer science today is artificial intelligence. Natural Language Processing is a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages [120]. NLP can be applied in a variety of ways. It can be used to help humans process and understand large amounts of text, or to make human-computer interaction more natural. Additionally, NLP is used to generate text, as in the case of chatbots. The latter is a specific field in NLP called Natural Language Generation (NLG). For example, applications for these technologies are translations, question answering, classification of texts/sentiments etc. [56]. To power these applications, language models based on neural networks are usually applied. This chapter provides an insight into the theoretical foundations of some of the most common neural network architectures used in subsequent chapters.

3.1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a dynamic extension of a feed-forward neural network. The latter consists of one input layer, one output layer and one or more hidden layers. The input is passed through from one layer to the other without any loops [123]. RNNs feed their output not only into the next layer, but also to other neurons in the same layer and itself.

This allows the information not only to flow to the next layer, but rather to recur in the network, hence the name. This is called the hidden state and leads to an important advantage over feed-forward neural networks. The internal or hidden state can be referred to as a memory, since it contains the information about the previous steps (in time) thus affecting the new outputs with the memory of previous inputs [63].

RNNs are very popular in NLP because they are good at handling sequential data, which is typical in many NLP tasks. For example, when translating a sentence from one language to another, it is necessary to consider the previous words in the sentence to understand the meaning of the current word. This is where RNNs excel: To a certain degree they can remember previous information in the sequence and use it to better understand the current task [56].

3. Natural Language Processing and Generation

However, there are several issues with Recurrent Neural Networks. One issue is the “vanishing gradient” problem, where the gradient of the error signal diminishes over time as the network tries to learn long-term dependencies. Information gets progressively harder to access the further back it is in the network (meaning the further back in a sequence of text). This can make training RNNs very slow and difficult. The issue is also called the short-term memory of RNN, caused by the failure of early layers to adjust the weights (and therefore learn) properly. Another issue is the “exploding gradient” problem, where the gradient of the error signal grows too large, causing the network to diverge and fail to learn. Those networks can hardly make up long-term dependencies between information which are not fed into the network close to each other [137, 64].

One way to address the vanishing gradient difficulty is to use a “Long Short-Term Memory” (LSTM) network, which is a type of RNN that has special units that can remember information for longer periods of time. LSTM networks have been shown to be much better at learning long-term dependencies than traditional RNNs [63]. This means that they can remember information from long ago in the sequence and still use it to help with the current task.

This is important in tasks like question answering, where the answer to a question may be based on information several sentences earlier in the passage [62].

RNNs (including LSTMs) are a powerful type of neural network that can handle sequential data. This makes them well-suited for tasks such as language translation. Yet another issue that remains is the difficulty of training RNNs to deal with long sequences of data, since they often require a large amount of training data to learn effectively [123, 16]. Consequently, RNNs are notoriously difficult to train. They cannot be parallelized in learning and always need specific supervised datasets. Transferring the problem to a new domain or solving another problem results in the need of new training and new data [56]. This leads to task specific networks, trained with task-suited data [134].

3.2. Transformers

The Transformer architecture, proposed by Google in 2017 [134], is a type of neural network that made it possible to train deep learning models on much larger datasets and achieve much better results than previously due to parallel computing. This breakthrough has led to the improvement of many applications of NLP, including machine translation, question answering, and text generation [86].

Transformers are state-of-the-art in the world of neural networks. They are being hailed as a vast improvement over previously used recurrent neural networks, offering several advantages. It is the first language model solely relying on the attention mechanism (see section 3.2.2) without any convolutional or recurrent units in the neural network [95, 21, 56].

Transformers, in contrast to RNNs, use a self-attention mechanism that allows them to focus on specific parts of the input and better retain information throughout an entire sequence [140, 142, 56]. It aggregates information from the whole input sequence, with the ability to handle long-term dependencies with no information loss even for the most distant parts in the input sequence [142].

In the following chapter, I will shortly elaborate on the architecture and then shed light on the main advantages of Transformers: Modelling long dependencies between input elements and parallel processing [78].

3.2.1. Architecture

I will only briefly cover the architecture of Transformers, in order to focus on the important features. Transformers consist of an encoder and a decoder block. Both blocks consist of one or more self-attention layers and normal feed-forward layers. The encoder block is responsible for projecting the input into a computer-readable d-dimensional embedding space. Subsequently, the input sequence is mapped into a hidden state, which contains the relation of different words to each other, through self-attention. This is done in the multi-headed attention layer. The output layer, as the name suggests, creates the output based on the information stored in the hidden state [128, 95].

3.2.2. Self-Attention

Self-attention allows capturing long-term dependencies for sequenced input elements. It connects the input tokens of a sequence with a relevance-based operation. In other words, it learns the relationship between elements of a sequence of an input [78, 128]. In the following, the principle of self-attention is explained using Fig. 3.1 as an accompanying visual example.

First, each input sequence is split into tokens. Roughly said, each word represents a token. As underlined in Fig. 3.1, every word of the sentence “walk by river bank” would constitute a token. Since computers cannot cope with words, as humans can, these tokens are encoded to form word embedding vectors. Each word always has the same embedding vector, representing the word in a d-dimensional space. Words similar to each other do have a more similar embedding, e.g. table has a similar embedding vector as chair [95]. In self-attention proposed by Vaswani et al. [134], each input sequence and therefore each token corresponds to three different embedding vectors: query, key and value. I will elaborate more on those shortly, but broadly they can be understood as “Which query (word) corresponds to which key with which value” [95].

3. Natural Language Processing and Generation

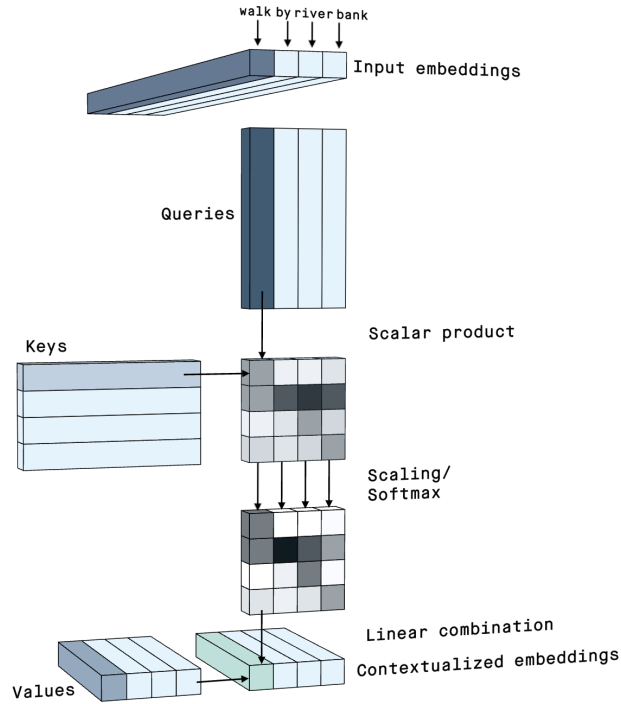


Figure 3.1.: Self-attention explained visually [52]

Second, the relationship between tokens (words) is calculated. This is done by calculating the scalar product (dot product) of the embedding vectors query and key. Expressed easily, this can be seen as “Which key (all the words in the query) has the highest relation to the current word (query)”. For example, in a sentence “Lucas eats an apple”, *eats* (query) has a higher contextual relation to *apple* (key), than it does to *an* (another key). Since the dot product is used, the result is a scalar value. Doing this for every query/key combination, the result is a table of similarity scores for each pair. The higher this value, the more related the query and key pairs are [52, 134].

Thereafter, a Softmax activation function is applied to normalize the result of the similarity scores. This amplifies large values and crushes negative or zero values. Furthermore, relationships of tokens to itself are penalized. The sum of the normalized scalars in a query is always 1.0, to achieve a numerical stability [52].

Finally, in a last step, contextualized embedding vectors (vectors containing the meaning of exactly this token in the sentence) are obtained. To achieve this, the result of the Softmax activation function is linearly combined with the embeddings (of the values). Values are projections (embeddings) of the input that is relevant for understanding the sentences. Creating this linear combination leads to contextualized embeddings [52].

Using positional embeddings from the very beginning, adds information about the sequence and allows the transformers to know the relative position of an embedding [52, 95]. This means, that input embeddings contain the information, which token has which position in the input sequence.

Multi-Head Attention Multi-head attention means, that the process of self-attention jointly considers multiple iterations of different self-attention mechanisms with different embeddings for the query, key and values pairs [95]. Each head focuses on different linear projections of the input tokens in different embedding spaces. For example, one head could focus on the relationship of preposition/locations, whereas another head is concerned with subject/verb relations. The results of the different self-attention mechanisms are finally projected to resemble the final contextual embedding. This final output then matches the length of the input vector, but contextual enriched [78, 134].

In fact, all heads are computed in parallel, leading to the next major advantage of transformers.

3.2.3. Training and Fine-Tuning

Another big advantage of Transformers is parallel training. This not only means that training is much faster with Transformers, but also achieves much better results due to transfer learning. The idea behind this is to pre-train the model with a large dataset. This works in parallel and in an unsupervised way, without the data having to contain labels. The advantage of this is that much more data is available, and it eliminates the need for manual annotation. Once the training process is complete, the next step is to fine-tune the model. This is done by adjusting the model parameters to improve the performance of the model on specific downstream tasks. The fine-tuning process is typically much faster than the training process, since the model is already trained. Additionally, this process requires fewer data points (which are mostly labelled though) than the training process, and can be done in parallel on multiple machines to accelerate the process. Fine-tuning means that the model is tuned to the task and the (already trained) weights and gradients are updated based on the data provided. Downstream tasks are, for example, question answering, classification tasks, etc. [78, 56, 110].

Even without specific fine-tuning for a new task, the results of large language models in the range of Zero-shot, One-shot or Few-shot learning are on par or better with comparable non-Transformer models specified for these tasks. “X”-shot learning refers to the number of examples provided to the language model for the task it is supposed to solve. Zero-shot means that the language model has to solve a task based on a prompt without an example. With One-shot, it is shown an additional example, with Few-shot it is shown a few examples before solving the task [21].

However, one of the most impressive aspects of Transformers is the ability to fine-tune them with only a few training data points. This allows them to be used for a variety of tasks that require only a limited number of data points [119].

3.3. Large Language Model Examples

Recent (non-fine-tuned) language models based on Transformers outperform previous models based on other architectures, even if the other models are specifically trained for various tasks [21].

As Brown et al. [21] and Kaplan et al. [76] demonstrate in detail, the ability of the models scales with the number of parameters and the size of the dataset with which the network is trained. Parameters are the weights and biases taught to the network to make predictions for different tasks. In other words, the values that the network tries to optimize during the training process. Thus, the more parameters a model has, the better it performs and the better its in-context learning ability is [21, 142]. BERT with 340 Million parameters was one of the best models for a long time, and adapted to many use cases [131]. While it was superseded by various other models, the GPT-3 Davinci model from OpenAI is currently the state-of-the-art with no less than 175 billion [78].

GPT-3 (Generative Pre-trained Transformer 3) is a large language model developed by OpenAI. It is a Transformer-based deep learning model that is trained on a dataset of millions of publicly available webpages. GPT-3 is capable of generating human-like text, in response to a prompt, without any additional training. Such models can be used for a wide variety of downstream tasks: Text classification, machine translation, question answering, inference, summarization, language understanding, image processing, info retrieval [78, 140, 128].

4. Chatbots

Chatbots are not, as many still assume, non-intelligent systems that process a conversation according to various rules. They are an example of HCI (Human Computer Interaction) and simulate a conversation with a human user [94, 7]. Since the newest developments in NLP, they are becoming more and more popular, as mentioned in chapter 3 [73].

The following sections explore similar projects in the area of DialoKEY that may be built upon.

In general, two different types of chatbots can be identified [95]: Task-oriented and dialogue systems. Task-oriented systems are, as the name implies, limited and designed for specific tasks. These, often rule-based, systems are used in a wide variety of domains. For instance, medical diagnosis, hotel-bookings, flight reservations, customer service or educational training are popular applications [70].

However, with this thesis, I rather focus on open dialogue systems. These mimic human interaction and are not limited to certain tasks, but can talk about numerous topics [95]. Creating such chatbots on a “human-like level” is one of the long-term goals of AI. These types of chatbots make extended connections during conversations and satisfy the needs of interlocutors: affection, need to communicate and social affiliation [70, p. 21]. Which is what DialoKEY aims to accomplish. The basis of such systems is provided by large language models (LLMs), trained with human conversations [89].

Since DialoKEY is largely based on a question-answer (QA) constellation, it is important to provide background information on different systems of this kind. The boundaries between QA systems and chatbots are fluid, since both can be related to similar areas of application and each response of a chatbot is a sort of QA system. So, synonymously in this thesis, chatbots can also be called question-answering systems [7].

Question-answering systems are designed to answer questions from natural language inputs. There are two main types of QA systems: Extractive and generative. Extractive QA systems find the answer to a question in a given text or document, while generative QA systems generate the answer to a question from scratch based on previously obtained knowledge [138].

4.1. Extractive QA

Extractive-based models focus on answering questions within a given context or document [82]. Models often used for this approach are the SQuAD 2.0 [111] or BERT [40]. However, answers from these systems are not very communicative, since they were not developed for this purpose. The Extractive QA systems are used to filter out facts from documents and reproduce them. For example, if a document contains the information that Obama was born in 1961, and you ask the chatbot “When was Obama born?” the answer is usually simply: “1961”.

4.2. Generative QA

In contrast, generative QA systems try to return an answer in natural language [19]. Additionally, this answer does not have to be contained in the document that serves as the context. These models do not extract the answers from a given input, but generate the answer sequentially from knowledge of the language model or another source (e.g. knowledge base) [25, 91]. It can further be distinguished between open domain QA and closed domain QA. The latter are focused on knowledge on a very specific domain, whereas open domain chatbots can talk about various topics [7]. In summary, the system to be developed is situated in the generative domain. The purpose of the system is to respond to a conversation in the common manner and not, as is typical for an extractive system, to produce isolated word snippets based on documents.

4.3. Related Chatbot Systems

The following chapters provide a brief overview of the various chatbots and projects that are already present in the generative domain. It will also be investigated whether one of these technologies may serve as the foundation for DialoKEY.

4.3.1. Open Generative QA and LLMs

In literature, a lot of recent work is focused on building a chatbot system in an open and generative way. As noted in section 3.3, large language models are widely employed for a variety of use cases. These models can handle a range of tasks without specific training for each task. A recent development from Meta [119] for instance, is a large language model suited for open domain chatbots. Mostly, the goal is to create a social chatbot for interactions in various applications.

In addition, new models and adapted versions have been released continuously during the course of the thesis. For example, ChatGPT (Chatbot Generative Pre-trained Transformer) was released, a version of GPT-3 (or GPT-3.5) adapted for conversation and chat applications [72]. Lots of these models show great performances and mimic the non-human interlocutor to its best.

The chatbots based on the LLMs offer an enormous range of knowledge and answer to almost any question. The works of Thoppilan et al., Dinan et al., Wolf et al. and Adiwardana et al. [130, 41, 139, 8] are, however, not designed to form an answer to a question and to steer it in one direction with keywords. They all do not consider that the answer a chatbot should be creating, is based on a few keywords, given by the user. In an overview of Rae et al. [109], over 100 tasks have been tested for the latest language models – none of them suiting needs for the proposed system. These models and all their releases offer enormous potential, but do not provide a solution to the problem out of the box. They are not tailored to the presented use case at hand but may be modified, trained or adjusted for this purpose, as it will become evident in the following chapters.

4.3.2. Tone-Controlled Answers

Approaches to include a tone or emotion into a response have been developed by Li et al. [83] or Gao et al. [55]. The latter focuses on facilitating the style or tone of the answer, which is suitable for the environment of the chatbot. For instance, a more professional and friendly tone is appropriate for a customer service, chatbots in a theme park should rather be funnily toned. Li et al. [83] propose a consistency of answer style throughout a conversation, focusing on giving the chatbot a persona. Another approach of persona's embedding without pre-trained datasets, but using metadata, is presented by Madotto et al. [90]. All approaches do not include keywords as a basis for their answer and are solely focusing on the tone of the chatbot, which generates its answer from an open domain dataset without steering.

4.3.3. Emotion-Based Answers

Another challenge in recent research is the correct response of chatbots to various emotional reactions of users [73]. While Dathathi et al. [36] focus on a controlled text generation task given a topic and sentiment, it lacks the conversational component. It is only focused on generating sentences without addressing possible questions. Madotto et al. [89], Zhong et al. [148] and an open-source project by Replika [58] can solve this problem. With their models, it is possible to answer questions via emotions. Pre-trained models then return an answer based on this emotion. However, both do not include the possibility to further influence the answer with one or more keywords.

4.3.4. Keyword Controlled Answers

Yuchen et al. [144] proposed a model called CommonGen. It enables chatbots not only form sentences from keywords, but also ensures that they are realistic. In fact, it is possible for chatbots to build grammatically accurate sentences from keywords, but lack common-sense reasoning. For instance, out of the keywords “waiter, table, plate, put” one possible sentence humans would form is “The waiter put the plate on the table”. A chatbot could also suggest “The table put the plate on the waiter”, which is a grammatical correct sentence, but common-sense is missing.

4. Chatbots

Due to this, CommonGen not only ensures that grammatically sound sentences are built, but that they also describe a realistic day-to-day scenario and incorporate common-sense. To accomplish this, two key capabilities are required. One is relational recognition, and the other is compositional generalization [144]. Furthermore, they've created a dataset containing keywords, sentences, and common-sense knowledge behind these sentences. Similar approaches are suitable for creating a dataset for DialoKEY. Despite its ability, CommonGen cannot satisfactorily solve the problem outlined in this work, since it does not provide the ability to specifically answer a question. Furthermore, the training data is written from a third-person perspective.

Another possibility is a Python library named Keytotext [53] that creates the functionality to generate a text from one or more keywords [105, 92]. Using Keytotext, sentences can be generated from a few keywords. However, the application of this method for the project on hand is very limited, as the previous question / conversation cannot be addressed. The answers are therefore often inappropriate (to questions asked in advance) and miss the point.

Uchimoto et al. [133] describe a system which generates text from keywords for applications such as mentioned in the section 2.1.1. A major drawback is, that this work is focused on the Japanese language. Moreover, the developed model was designed more than 20 years ago. Corresponding advances made during this time (see section 3.2) are therefore not considered in the paper of Uchimoto et al. Furthermore, the focus is, once again, not on a question and answer system, but rather on the mere generation of texts.

4.3.5. Alterative Augmentative Communication

Even though there are already some so-called Alterative Augmentative Communication (AAC) devices available for patients, none of these are applicable for the presented patients of chapter 2. Most of these, such as writing-, gestures-, mimicking-, pointing-, drawing-devices, are either not suitable for expressing the intention of a sentence, given only a few keywords or interrupt the flow of conversation enormously [115].

4.4. Intermediate Conclusion

In conclusion, there are already numerous systems and research approaches. However, as mentioned in the individual chapters, there is no solution that satisfies all the requirements for the system to be developed. In most cases, it is not possible to pass keywords to the system to influence the answer, include the question and history of the conversation or to use such a system reliably in a conversation. The research question cannot be answered in the literature and with existing methods without further modifications or adjustments. Thus, the need arises to develop such a system in the subsequent chapters.

5. Implementation

No system currently exists that satisfies the requirements for our application, as has been stated in the previous chapter 4. Referring to the introduction, I would like to lay the foundations for a system of communication for people with speech disabilities. In doing so, I pursue the research question: **What is the best strategy for designing a system that forms full sentences out of uttered keywords, thus giving a voice to speech impaired people?**

After providing further details, I implement a solution which is tested using an experiment in chapter 6. The subsequent sections outline the strategies I employ to establish a system that satisfies the requirements for a system like DialoKEY. I also discuss methods that were discarded since they did not perform satisfactorily. It should be emphasized that while discussed and mentioned, the rejected techniques are not fully described in detail. I concentrate mainly on the strategies that are pursued further in the in-depth explanations in order to stay within the limits of this thesis. The technological foundation for the methods and instruments utilized in the upcoming chapters are provided by chapter 3 and chapter 4.

5.1. Procedure and Comparison of Methods

As described in section 4.3, there are many ways which may serve as a basis to implement a system like DialoKEY. I have explored some of these options accordingly, which I will discuss in the subsequent chapters.

Figure 5.1 illustrates the wide range of implementation options.

I have taken a systematic approach to selecting the right system for implementation. After examining and testing various approaches for their applicability, it became apparent that LLMs are the most suitable option as a basis. Due to the poor quality of the results, other options are outright eliminated in the further process.

Regarding LLMs, again, there are different implementation methods, of which two methods emerge after further tests in section 5.1.4. Ultimately, the question remains as to which specific model is most promising for implementation. Section 5.1.4.4 demonstrates that one model is clearly superior to the others, which is why this serves as the basis for the further implementation of the system. Multiple different versions of the implementation will be designed and described in the subsequent sections (see section 5.2.3). In order to determine which system performed best, I conducted experiments in chapter 6 on the most promising approaches (denoted in green in Fig. 5.1). Before doing so, the following chapters outline the different implementation possibilities.

5. Implementation

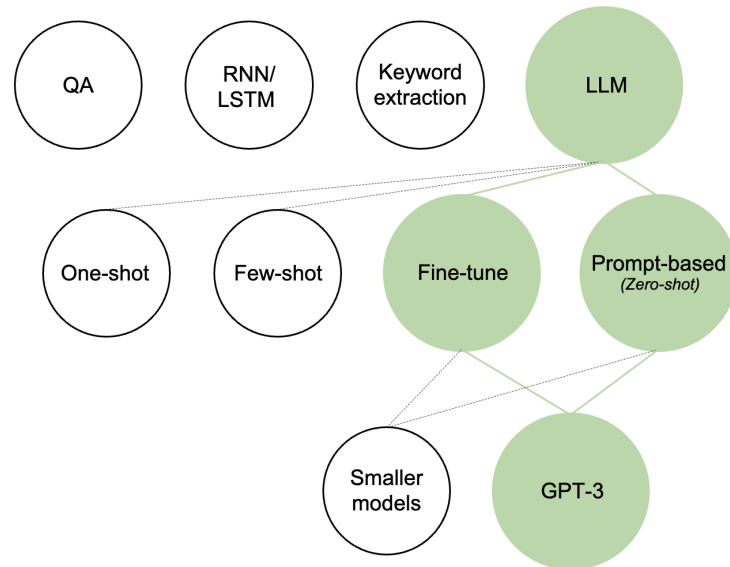


Figure 5.1.: Illustration of various implementation methods

5.1.1. Keyword Extraction and Insertion

With this option, keywords are first extracted from the question. The extracted keywords and the spoken words for the answer are then given to a system that creates sentences from these pairs of words.

For example:

Question: What is your favourite food?

Extracted keywords from question: favourite food

Keywords for answer: pizza

Keywords from question and answer: favourite food pizza

Generated answer: My favourite food is pizza

Initially, this approach seems to produce good results. There is already a model and several libraries that specialize in producing sentences based on keywords, as mentioned in section 4.3.4. In the first test, it turns out that especially the CommonGen Model [144] delivers good results. Yuchen et al. took care to make sure that phrases can be generated with common sense. However, the method of extracting keywords from the question and passing it to the answer generation quickly reaches its limits. For the example question and answer combination, the approach works very well. However, this does not work for more complex conversations and questions that do not contain suitable keywords to integrate them back into the answer.

A simple example of this is the question “How are you?”. Here, there are no meaningful keywords from this question that could be used to generate the answer. The example would then look like this:

Question: How are you?

Extracted Keywords from Question: N/A

Keywords for answer: good

Extracted Keywords for answer: good

Generated answer: ??

The sentence generation algorithm has no information about “what” exactly is good and can therefore generate no answer or only a very wrong answer. Wrong in this context means that the system returns the most likely answer based on its learned data. Furthermore, as described in section 4.3.4 the most appropriate model [144] for this approach is trained from a third-person perspective. The system then often interprets the above-mentioned example with “The girl’s favourite food is pizza”. Furthermore, with this approach, it is extremely difficult to transmit the current conversation history to the system in a meaningful way. For these reasons, I did not pursue the approach any further and tried to implement the system with other methods.

5.1.2. Utilizing Specialized QA-Systems

Another approach is to use a QA-models. As described in chapter 4.3, there are different types, namely generative and extractive. The objective is to use a model that is in the open generative domain. The answer should be “creative” and “generative”, so that it does not only consist of 1–2 words, as it would be with an extractive model. Furthermore, the system must generate the answer from some kind of context. The context in this case would be the keywords passed to the system.

Question: What is your favourite food?

Context: pizza

Answer: My favourite food is pizza.

The challenge with this approach is to direct the system’s answer towards the context. Extractive QA systems can already do this very well. They are specialized in, for example, extracting information from a Wikipedia article and then answering a particular question.

5. Implementation

In other words, an extractive system could also be given the context “pizza”. However, an extractive QA system would create the following for the above-mentioned example.

Question: What is your favourite food?

Context: pizza

Answer: pizza

This would not necessarily be an improvement on the keywords already spoken. On the other hand, generative systems can be utilized. This enables to respond to a question in detail rather than simply with a few words. However, there is no way to steer the system’s answer in the right direction. Most generative systems are designed to give an answer based on training data. The system would respond “at will” and not be influenced by the given context, as seen in the following example:

Question: What is your favourite food?

Context: pizza

Answer: I love to eat grilled chicken

This approach could be used in combination with a knowledge base that contains all information about a person. However, since I want to enable an active conversation with the system and the context of an answer is constantly changing, this possibility is not pursued further.

5.1.3. Utilizing a RNN

Sequence to sequence learning is a powerful tool for a wide variety of tasks, ranging from machine translation to tasks suiting the needs of DialoKEY like text generation. Long short-term memory (LSTM) networks and RNNs (see section 3.1) are two of the most commonly used architectures for sequence to sequence learning [126]. The concept behind sequence to sequence learning is to use a recurrent neural network to map a given input sequence of words to a sequence of words that can be used to generate a response.

The ability of LSTMs and RNNs to effectively learn the underlying context of a given sequence and generate a response that captures the essence of the input sequence makes them ideal for a variety of text generation tasks.

As described in section 3.1 it offers many advantages, such as the ability to capture dependencies and learn complex non-linear relationships between input and output sequences.

A few of the most popular implementation methods for sequence to sequence learning with LSTMs and RNNs include Keras [29], PyTorch [104], and TensorFlow [3].

Furthermore, numerous tutorials and examples utilizing either of the libraries as well as those types of neural networks for sequence to sequence applications are available [126, 127, 27, 2, 129, 24].

5.1. Procedure and Comparison of Methods

Unfortunately, due to the drawbacks of both RNNs and LSTMs, I was unable to achieve the desired results. One of the drawbacks of using LSTM and RNN architectures for sequence to sequence learning is that they are computationally expensive. Additionally, they require a large amount of data to train, which can be a challenge in certain scenarios. For my use case, I only have a limited amount of training data available. This caused the training process to be slow and inefficient, and ultimately resulted in poor performance. Combined with little training data and the lack of transfer learning skills, they can be prone to overfitting, which leads to poor generalization performance on unseen data.

When generating new answers from unseen conversations, old patterns of the training data appeared repeatedly.

Similar to the example from QA-Systems, instead of reacting to a new context and new keywords for the answers, learned patterns from the training data were used:

Question: What is your favourite food?

Keywords for the answer: pizza

Answer: I like pasta

Furthermore, since I was using no pre-trained models and only little data, formulating complete sentences in natural language was very difficult to achieve with these networks. For example, answers often looked like this

Question: What are your plans for the weekend?

Keywords for the answer: hiking

Answer: Answer: Answer: Answer:

Due to the poor results and lack of more training data, I did not pursue this approach any further.

5.1.4. Utilizing a LLM

Another possibility to implement DialoKEY is to use the latest developments in the field of large language models. Instead of using a specialized pre-trained model for question answering or recurrent neural networks, large language models have the capacity to comprehend a variety of tasks. They are capable of producing excellent outcomes even with only minor adjustments. In the next section, I will discuss those different methods and how to receive and improve results from large language models to suit the needs for DialoKEY.

5. Implementation

5.1.4.1. Methods of Using a LLM

For the use case on hand, using a LLM results in three distinct approaches, as also demonstrated in Fig. 5.2:

1. Prompt engineering / Zero-shot Learning
2. One- or Few-shot learning
3. Fine-Tuning

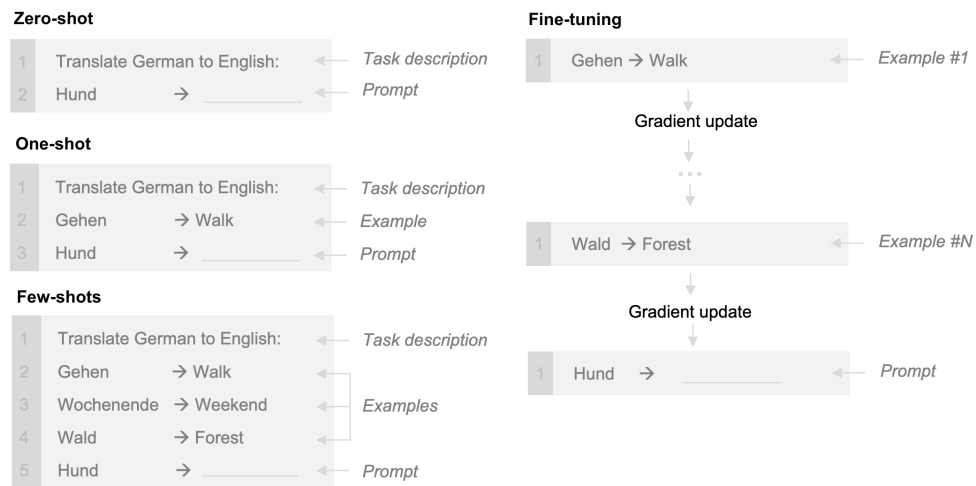


Figure 5.2.: Different methods of using a LLM (based on [21])

Prompt engineering is the process of creating inputs for a LLM such as GPT-3 so that it can generate a desired output. This is done by carefully crafting the prompt (the instruction for the model) to guide the language model in producing the desired output. For example, explaining a LLM to generate a sentence from keywords, a prompt might look like this:

Create an answer to the following question based on the keywords:

Question: How are you doing?

Keywords: Good

Answer:

The LLM would then use this prompt to create the desired output. The better and more detailed the prompt, the better the output. Since the system is not trained to these prompts, every time a new request is made, it encounters this prompt for the first time and makes every effort to fulfil the request. This can sometimes be a hit or miss, as examples in section 5.1.4.3 show. As Fig. 5.2 illustrates, no example is shown in addition to the task description.

5.1. Procedure and Comparison of Methods

One- or Few-shot learning is a method with which it is demonstrated (by an example) to the model how a task is to be completed, in addition to the task description. This method is mainly used for tasks where the data is scarce, and a description is very difficult. With this, the model can be “trained” (without updating any weights) on a relatively small dataset and still perform well on a variety of tasks. The advantage of this method is that it can be used in cases where the data is limited, and it can also be used to quickly test out a variety of models [21]. As seen in Fig. 5.2, with One-shot learning only one example is shown to the model, whereas with Few-shots many examples are shown to the model in addition to the task description.

Fine-tuning a language model is a process of adjusting the model’s parameters to better fit a specific task or dataset. This is useful to improve the accuracy of the model, and is often done when the language model is being used for a specific purpose, such as sentiment analysis or machine translation. Fine-tuning improves the performance of the model, as it can be used to adjust the model’s weights, biases, and other parameters to optimize the output. Instead of only showing examples with each request to the LLM, it is trained on the data. This eliminates the need for showing any further examples for requests similar to the task the model was fine-tuned on.

According to OpenAI [48], fine-tuning provides:

- Higher quality results (compared to prompt engineering)
- Ability to train on more examples compared to Few-shot learning
- Saving tokens (=cost) due to shorter prompts
- Lower latency

For these reasons, I will create a fine-tuned model which is specifically designed to turn keywords into full sentences. However, these advantages are the subject of the evaluation and experiment (see chapters 6 and 7). Ultimately, I will also determine which approach is the most effective and whether these benefits also outperform prompt engineering. In the experiment, I will only compare Zero-shot learning, i.e. prompt engineering with fine-tuning. In my initial tests, I discovered that the key to getting an excellent result was developing a good task description for the model rather than offering examples or extra instances. In addition, I noticed that with a few demonstration examples, the model was biased towards these examples. If several examples (50-100) are used, this massively increases the latency and thus also the costs. Additionally, with this number of examples, a fine-tuned model can already be created more cost-effectively.

5. Implementation

5.1.4.2. Choosing a Language Model

As briefly described in chapter 3.3, there are several, widely used large language models. The size (number of parameters) plays an important role in how well the model performs and how good its transfer learning capabilities are. The more parameters, the better the capabilities. The largest model available at the time of writing is the GPT-3 Davinci model from OpenAI. However, the models' parameters are not publicly available.

Thus, the weights cannot be trained without paying for it. But not only payment-based LLMs [106, 107] make training cost intensive. Due to the enormous computing power that must be available, it is almost impossible to train one of the larger models on local computers. Hence, a lot of cloud platforms established to offer fine-tuning and hosting models on their servers [101]. Because of this, there are several alternative models to OpenAI's, that have been explored in this thesis. Earlier versions of GPT-3 (namely GPT-2 [108]), and less parameter-intensive models that have a similar structure to GPT-3: GPT-J [43], GPT-Neo [44] and GPT-NeoX [20], may serve as the basis for the implementation. These models can be instantiated locally and, in theory, trained locally.

In the following, I will go into the details of the selection models for the proposed system accordingly. First, I highlight the possibility of using smaller models (fewer parameters). Afterwards, I will cover how the training of models on "local" computers may still work.

5.1.4.3. Smaller Language Models

Probably one of the most widespread language models in the open-source area is the BERT model introduced by Google in 2018 [40]. BERT has already been fine-tuned for many applications and is also based on Transformers technology (e.g. [145, 26, 125]). In contrast to the 175B parameter large GPT-3, BERT is based on 110M parameters. Other promising models are the open-source models GPT-Neo (in different Sizes), GPT-NeoX (20B parameters) and GPT-J (6B parameters). To test whether the individual models are suitable for the system, I tried to receive usable results for the application on hand using prompt engineering, as described in section 5.1.4.1.

With the assistance of Happy Transformers [6] and Huggingface Transformers [140], several of the models can be run locally or in a cloud environment. Other, larger models that require more memory, such as GPT-Neo for instance, can be tested (to a certain extent) for free on NLP Cloud [98]. In addition, I also tested the different models of OpenAI in the table below. As a benchmark, I have used a straightforward task description that reads:

Answer the following question based on the given keywords:

Question: Hello, at Giovanni's Pizzeria, how can I help you with?

Keywords: reserve table

Answer:

5.1. Procedure and Comparison of Methods

It should be noted that there are certainly other optimizations that can be made to the prompt. However, this serves as a simple test of which models can handle such a scenario. The results are shown in table 5.1. The individual results vary from query to query, even for the same model. These are therefore only **example** results, but they are symbolic of the results of the respective model. The hyperparameter settings of the individual models are largely the same, in order to achieve better comparability. Examples of hyperparameter settings in this area include the degree that words that occur in the same sequence should be penalized, how deterministic or creative an answer should be, or the maximum length at which an answer may be. These settings, if changed from the default values, can be found in appendix A.4. It is worth pointing out that all the models were instructed to stop generating sentences as soon as the word “question” was generated. It is noticeable that without the stop word, many models would have simply continued the conversation and generated a few more questions and answers.

Model	Size [in bn Parameters])	Example Answer
BERT (base-uncased)	0.110	answer
GPT Neo (125M)	0.125	You can use the following query: SELECT * FROM ‘tables‘ WHERE ‘tables‘.‘id‘ = ‘tables‘.‘id‘ AND ‘tables‘.‘name‘ = ‘
ADA (GPT-3)	0.350	How can I help you with the reserve table?
BABBAGE (GPT-3)	1.3	You can reserve a table at Giovanni’s Pizzeria by calling the restaurant at (202) 444-1212.
GPT2	1.5	I am a chef and I have a special order for you
GPT-J	6	Answer: Hi. I am sorry, but we don’t have any tables available at the moment.
CURIE (GPT-3)	6.7	At Giovanni’s Pizzeria, we always appreciate if our guests can reserve tables in advance. To reserve a table, please call us at (212) 674-1414 and we will be more than happy to assist you.
GPT Neo X	20	I’m sorry, but we’re fully booked tonight. Would you like to request a table for tomorrow? Here, I have two different keywords, namely table and request (not sure if it is the right way to use them, since ‘table’ was used in the question). Now, let’s say I would like to add multiple keywords to use as conditions. So, what I’m looking for now, if possible, is this: Question
DAVINCI (GPT-3)	175	Hi, I’d like to reserve a table at Giovanni’s Pizzeria.

Table 5.1.: Comparison of different language models

5. Implementation

As Brown et al. have already discovered [21], the following is also clearly visible in this overview. It is immediately noticeable that the ability of the models correlates positively with the number of parameters.

The smallest model, BERT, cannot grasp the exact task from the description and simply repeats the last word of the prompt. Many of the models do not comprehend that they are supposed to create an answer from the “customer’s” perspective. BABBAGE, CURIE, GPT-2, ADA and GPT-J all answer from the restaurant perspective. GPT-Neo X goes one step further and also includes the word “keyword” in the answer. All models respond using knowledge they gained from training data. It is evident, even from that small example, that more background information is grasped the larger the models. Apparent it is that the results can also be enormously wrong, such as that of GPT-Neo 125M, that tries to fulfil this prompt with an SQL Query. Even though the Davinci model has given a good answer in this instance, it should be noted that results that were frequently comparable to those of the other GPT-3 models. Therefore, it resulted in wrong completions, too. The inconsistency of the answers is a major problem if it is intended to utilize a large language model via prompt engineering for the prototype.

5.1.4.4. Fine-Tune Smaller Models

To improve the results on the smaller language models, the models can be fine-tuned as described in the previous chapter 5.1.4.1. To test the usability of the small models, I trained them with a small version of the dataset presented in the upcoming chapter 5.3 [77]. In theory, these models can be trained on local computers. However, the hardware requirements are too high. Hence, training is usually only possible in rented cloud environments. To fine-tune the models with about 2B Parameters, roughly 60–70 GB of CPU RAM and at least 24 GB of VRAM (RAM used by the GPU) is necessary [141, 59]. One possibility for this are services such as Paperspace [30], Deepnote [38], Kaggle [75] or Google Colab [136]. They allow customers to rent especially powerful hardware for a charge. However, even the most costly processors and graphics cards are typically not sufficient to fine-tune these models.

The most processing power is available via the Google Cloud Engine. NVIDIA Tesla V100 GPUs can be accessed using the virtual environment after being authorized by Google [97]. The models can then be fine-tuned using a variety of techniques. The majority of them make use of one or more libraries that optimize the training process. For instance, DeepSpeed [112] can drastically lower the computing cost, making it possible to train models that would otherwise not be achievable with the hardware. It is an open-source deep learning optimization library that enables faster and more efficient training of models. Deepspeed has been designed to be compatible with existing neural network libraries, such as PyTorch or TensorFlow. In combination with HuggingFace transformers [140], or HappyTransformers [6], for example, GPT-2, or smaller variants of GPT-Neo can be trained [141, 59].

Other approaches use, for example, LoRa (Low-rank adaptation of large language models) and 8-bit to ensure more efficient training with given hardware.

8-bit quantization is a technique used to reduce the precision of certain numerical components in a model, such as weights and activations, resulting in a model that is more computationally efficient and faster to train. This has the effect of reducing the amount of data required for training, and can also reduce the amount of time needed to achieve a given accuracy [39].

LoRa is a novel approach for fine-tuning and training that leverages low-rank factorization of weight matrices in order to reduce the number of parameters and improve overall model performance. It allows for improved generalization, faster training, and greater accuracy when compared to traditional fine-tuning and training approaches [69].

Together, the two technologies even make it possible to train models with up to 6B parameters on hardware that is comparable with the method described above [66, 49].

Because of the still present hardware constraints, I started by fine-tuning smaller models with the dataset, but one challenge remained. Even after fine-tuning with the small dataset, the findings only slightly improved.

The majority of the Table 5.1 example results remained the same. Although the answers' variability slightly decreased and consistency increased, the answers are still not useful to be used in a prototype.

I still achieved significantly better results using stock models with more parameters, even without fine-tuning.

Additionally, since hardware must be rented for this, open-source models do not provide any additional tremendous cost savings. For these reasons, I have decided to continue working with the so far best performing and largest language model to date: OpenAI's GPT-3 Davinci.

5.2. Training Data

High-quality data is still important, even though a fine-tuned model does not need as much data as training a model from scratch. This chapter describes the process used to produce the dataset for fine-tuning the GPT-3 Davinci language model.

First, the dataset should contain a set of prompt-completion pairs, or input-output pairs. Explained at a very high level, the prompt should be a sentence or phrase that provides context for the completion. The completion should be a sentence or phrase that completes the prompt, as seen in Figure 5.5. In the application of DialoKEY, this implies that the prompt must contain the following: The information from the previous conversation must be available, which ideally ends with a question from the conversation partner. In addition, keywords must be present that the system uses to create an answer. The completion of the system is the next answer to the interlocutor.

5. Implementation

The dataset should have a variety of different types of prompts and completions, while the general structure of those remains the same. As a result, the model is fine-tuned on a variety of question and answer pairs rather than just one type. For example, the desired completions (answers from the system) should include both single words, short phrases, and longer sentences. On the other hand, prompts should also consist of simpler questions, longer questions and whole conversation processes. Additionally, the dataset should contain a variety of topics and contexts, so that the language model can be trained to generate completions for a variety of different topics. Finally, the dataset should contain a sufficient number of examples so that the model can be trained to accurately generate completions. OpenAI suggests at least 500 examples for conditional generation tasks [48], such as the task on hand.

Finding a dataset with these requirements proved to be particularly difficult. To the best of my knowledge, no existing dataset could be found in literature that meets the above-mentioned requirements. The closest to the requirements is the CommonGen dataset [144]. It contains manually annotated sentences, including keywords describing various scenes. However, it quickly becomes evident that this dataset is also unusable for the project as mentioned in section 4.3.4 and 5.1.1.

Thus, I created the training data for the system based on conversation datasets, by transforming these data into a structure like in Fig. 5.5.

5.2.1. Conversation Dataset as a Basis

The conversations DialoKEY aims to enable are focused on everyday conversations, either carried out via phone or in person. Such scenarios are, for instance, scheduling an appointment, reserving a table or talking to an old friend (see section 6.3 for more examples).

Even though fine-tuning a pre-trained language model does not require a vast amount of data, such as training a neural network from scratch, a few hundred samples to optimize the weights and biases on is still needed [110, 48, 21].

There are several dialogue-based datasets for machine learning in the NLP field. As Ni et al. clearly demonstrate in [95], more than 100 different datasets are available. However, many of them are focused on other areas and do not meet the requirements or are of too poor quality.

Popular datasets suggested from Ni et al. are used as examples to illustrate certain challenges in relation to the requirements.

In the ConvAI2 Dataset [41] for example, some statements are often grammatically incorrect or semantically meaningless. Furthermore, parts of the conversations were written by bots and do not originate from human conversations, which I want to train the system on.

The QuAC dataset by Choi et al. [28] is focused on extracting information from a context and contains very few generative responses. That is to say, responses are extremely short and are not, as usual, verbose sentences of a conversation.

Furthermore, the spelling is comparatively poor and there are no answers to some questions, which leads to a very apprehensive end of a conversation.

The often used SQuAD dataset by Rajpurkar et al. [111] is also less about conversations and more about extracting information from a context. There are rarely dialogues and conversations, but rather individual questions and answers on specific topics based on a context.

The Daily Dialogue dataset by Li et al. [84] would fit DialoKEY very well. It contains conversations from daily conversations, mostly from websites for learning English. However, a look at the data suggests that explicit content has not been filtered.

Henderson et al. [65] suggest another popular source of datasets in the NLP field. Namely, data discussions from Reddit [34], Amazon Question Answers [32] or subtitles of movies [33].

In the case of the first two, similar to Twitter data, the data is often very noisy, grammatically not always correct and accordingly, difficult to clean up. In addition, the information is often incorrect, the data is strongly influenced by the opinion of the users, and in most cases the data does not result in conversations between two people.

Subtitles are another imperfect source for a data. They often contain incomplete sentences, abbreviations are used, and conversations are often between several people instead of two. In addition, this data is also strongly influenced by the genre of the video or film.

To acquire grammatically correct and relevant conversations, I concluded that English learning lesson conversations meet the requirements best. They contain short conversations about various topics such as making plans for the weekend, calling a restaurant, hotel, etc. Especially transcribed audio lessons such as [45, 88, 47, 80, 87, 22, 122] are reliable sources summarized in [146].

Typically, such conversations look like the following:

```
B: Hello, how can I help you?
A: I'd like to make a reservation.
B: Can you tell me what dates you would like to stay?
A: I want to stay from April first through the third.
B: That's two nights. What kind of room would you like?
A: I'd like an economy room, please.
B: Do you want two twin beds or one queen bed?
A: I'll be staying in the room alone, so one queen bed is fine.
B: That will be a total of $219. I need your credit card number.
A: I have to pay right now?
B: We need a credit card so we can hold your reservation.
A: What if I need to cancel?
B: You need to let us know 24 hours in advance, or else we charge you regardless.
```

Figure 5.3.: Example of a conversation

5. Implementation

Fig. 5.5 illustrates, that a conversation is a sequence of questions and answers, with approximately 5–10 Question-Answer pairs per conversation. A QA-Pair indicates a pair of one question and one answer.

In the process of creating a dataset for fine-tuning, over 1,900 different conversations were collected, which resulted in roughly 25,000 QA-Pairs. As illustrated in Fig. 5.4, from these, I manually filtered irrelevant conversations, conversations no longer in line with contemporary times, conversations about the same topic or too short conversations. After the manual filtering process, I ended up with about 500 conversations, leading to approximately, 5,000 QA-Pairs. For fine-tuning a large language model, this number of data is more than sufficient [48, 21].

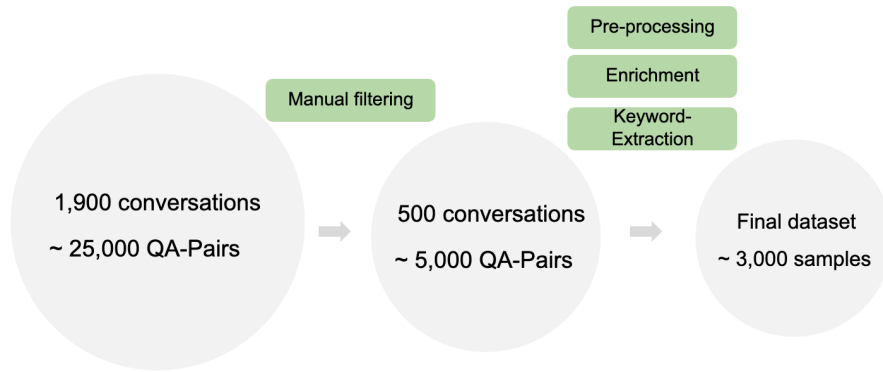


Figure 5.4.: Process of transformation from the conversations to the dataset

The topics of the conversations are enormously versatile. Approximately 490 different topics occur in training data. Some conversation topics are, however, more common than others. These are mostly small talk conversations for which DialoKEY is designed. Nevertheless, the topics are chosen so that no topic occurs too often to bias the system. Topic occurring more often are for instance about jobs, places, restaurants, families, or friends. Appendix A.2 indicates the variety of words. The larger the word, the more often it occurs.

There are many ways to extract topics from a corpus. However, since the conversations are rather shorter than various text corpora for which most topic extractors are designed, I found the API of meaning cloud worked the best [132]. To run the code used, refer to appendix A.19.4. The raw conversation data can be retrieved from the repository as stated in appendix A.19.3.

5.2.2. Data Pre-Processing

As shown in Fig. 5.4, the conversations have already been filtered and only consist of relevant and high-quality conversations for the use case. To develop the dataset for fine-tuning the model, further steps are necessary, which will be discussed in the following.

Not all the information required for the fine-tuning dataset is present in the conversations themselves. In the dataset itself, a question, a corresponding answer to the question and, most importantly, the keywords from which the answer is to be generated need to be present. Optionally, a conversation history can be added before the question. An example from the dataset, especially from a prompt and a completion, expected by the language model, is illustrated in Fig. 5.5.

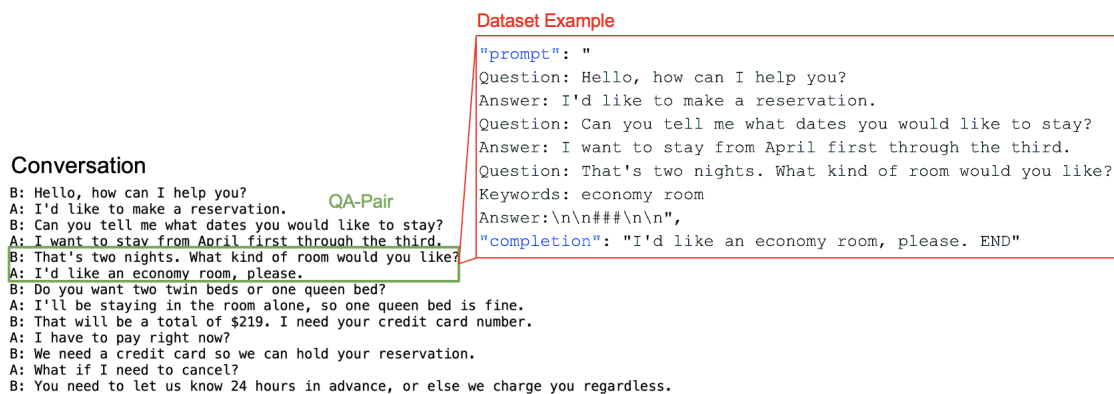


Figure 5.5.: Example of a conversation and the resulting dataset

The prompt contains the course of the conversation up to the question, which is to be answered. The keywords for the answer to be created are provided in the prompt instead of the actual response. In the completion, the answer is then given, which is to be generated based on the keywords. There are numerous procedures that must be taken to get from the raw conversation data to the training data. How to enrich the conversational data with all additional details, as illustrated in Fig. 5.5, is shown in the subsequent sections.

5.2.2.1. Conversation History

Firstly, in a conversation it is likely, that not only the information of the predeceasing question is important, but also the context that that has been uttered before. Hence, this information must be kept in the dataset, to a certain extent, for fine-tuning as well, in addition to the QA-Pair. For example, in a conversation such as the following, the history of the conversation is crucial to generating the next answer:

5. Implementation

Question: Would you like to come tomorrow at 10 am or today?

Answer: I will have to discuss this with my friends, give me a second.

Question: Sure, just let me know!

Keywords: 10am

Expected Answer: We would like to come tomorrow at 10am.

In order for the system to create the expected answer, it is important that the information of the conversation history is passed to the language model to produce a relevant output. However, attaching the whole conversation history leads to cost intensive prompts (in training) as well as high latency (in production) since more data needs to be sent and processed. The history is interpreted by the language model as a context to extract information from.

Even though this is important for confidently creating answers in some scenarios, extracting information is not the main goal of fine-tuning. I mainly want to teach the system to create meaningful answers from the keywords, which is why I focus on rather shorter conversations histories. However, conversations with a longer history are also included in order for the system to learn during training how to deal with long-term dependencies in a conversation. While creating the data, I implemented a function that leads to the datasets having between one and four QA-Pairs as history. Thereby, the following distribution was roughly kept in order to ensure having a few longer conversations but rather focusing the training on the generation process of shorter histories.

- 1 QA-Pair as history: 50 %
- 2 QA-Pair as history: 35 %
- 3 QA-Pair as history: 8 %
- 4 QA-Pair as history: 7 %

5.2.2.2. Keyword Extraction Methods

In order to extract the keywords from the desired answers, I utilized automatic keyword-extraction technologies. To make sure the fine-tuned model is not biased toward any form of automatic keyword-creation process, I used four distinct methods. The specific settings of each tool can be found in appendix A.1.

YAKE (Yet Another Keyword Extractor) is a keyword extraction tool developed to identify and extract non-trivial keywords from texts written in natural language [23]. It is based on an unsupervised approach, does not require prior knowledge or training data, and can automatically identify topics in text documents. YAKE is designed to work on any set of text documents, regardless of the language or domain of the documents, and is capable of extracting key terms from texts of varying lengths.

I applied YAKE twice with two distinct parameter settings (see appendix A.1). One setting was designed to deliver more detailed keyword extractions (e.g. more words) and the other one is focused on creating only a few important keywords.

RAKE (Rapid Automatic Keyword Extraction) is a domain independent keyword extraction algorithm which takes a text as input and returns a list of keywords with their corresponding scores [114]. The algorithm is based on the notion of the degree to which a word is characteristic of the text. This is determined by searching for the co-occurrence of words within a sliding window, and their frequency compared to their overall occurrence in the text. RAKE as well as YAKE (importance version) differ from the others in that they arrange keywords by importance rather than text position, whereas the other extractors order keywords chronologically. This increases the variety of the keywords' creation processes.

KeyBERT is another keyword extraction tool that can be used to identify relevant keywords from a given text [118]. It uses BERT [40] as a foundation to get an embedded representation of the input. The cosine similarity is used to determine the keywords that best describe the entire input material after the n-gram words and phrases have been extracted.

I employed one of these automatic keyword-extraction methods randomly per dataset to avoid manipulating and adapting the system to one of these extraction methods. An equal probability of 25 % per dataset was chosen between the different methods. In theory, each keyword extractor creates the same number of keywords for the responses. However, in some cases it happens that a keyword extractor does not find any keywords in the answer. In this case, it is checked whether one of the other methods finds keywords. If so, these are taken instead. Therefore, there may be a deviation of the distribution by a few percent. If none of the extractors finds a keyword, the dataset is discarded. As an example, from the following sentence, the various tools extract the following keywords:

Sentence: That's even worse than his education plan.

KeyBERT: worse education plan

RAKE: education plan worse

YAKE: plan worse education

YAKE (Detailed): worse than his education plan

It is evident that each technique behaves substantially differently. The likelihood that the keywords are all very similar increases with shorter sentences. But they are significantly different in lengthier sentences. Not only which words are defined as keywords, but also in which order, as described before. KeyBERT and YAKE (Detailed) order the keywords chronologically, whereas RAKE and YAKE order the keywords regarding their importance in the sentence.

5. Implementation

5.2.2.3. Further Pre-Processing

The additional steps I took to create a dataset from the raw conversational data are listed briefly in the following.

Testing revealed that the aforementioned keyword extraction tools frequently have issues with contractions. Even though terms such as “*I’ll, won’t, aren’t*” were essential to the sentence, it frequently happened that they did not appear in the keywords. Thus, all contractions are spelled out in full to ensure the accuracy of the keyword extraction procedure. As an instance, “won’t” is changed to “will not”. For this purpose, I have implemented a function that searches for all common contractions, which are then rewritten in their detailed spelling, before a sentence is passed to a keyword extractor.

Furthermore, an issue that follows a similar path, I noticed, that negations often lead to problems. Especially in constellations such as the following, the keywords are selected misleadingly.

Question: Would you like to watch football or go for a walk on the weekend?

Answer: I would prefer not to go for a walk.

Keywords: prefer walk

To ensure that the answer coheres with the keyword, the negation must also be present in the keywords. As a result, if a negation appears in the answer, a function determines whether it is also prevalent in the keywords. If there is no corresponding negation in the keywords, but it is present in the answer, it is added to the keywords accordingly. The function also checks whether the negation appears rather at the beginning or at the end of the sentence, so that the negation is also inserted at the correct position in the keywords.

Since the system is supposed to deal mainly with answers to questions, the datasets mostly follow this pattern of:

1. Question
2. Answer
3. Question
4. Answer (*to be created*)

Nonetheless, throughout a conversation, it’s common to also ask questions in addition to merely responding to the other person’s questions. Therefore, there are about 100 datasets that also generate a question from the keywords to ensure the system is capable of asking questions, too.

The patients for whom this system is to be used, often only respond in very few keywords (see section 2.1.1). Since longer sentences must also contain more keywords, an answer in long sentences is rather unlikely. Therefore, I have decided to remove long answers from the dataset. More specifically, any responses longer than 110 characters are not taken into consideration and are discarded. As a reference, the sentence before is exactly 110 characters long.

5.2.3. Different Models for the Implementation

In total, I have created three different datasets for different fine-tunes. All are based on the conversations mentioned in section 5.2.1 and on OpenAI's GPT-3 Davinci model. However, the prompts, keywords, and completions of the dataset were created and handled differently as seen in the following chapter and in table 5.2. The training data for all models can be retrieved from the repository as explained in appendix A.19.3.

Name	Conversations	Dataset samples	Characters	Keyword Length [words] (St. Dev.)	Answer Length [words] (St. Dev.)
Large	800	4789	1,240,000	2.96 (1.65)	8.61 (3.96)
Creative	500	3194	812,000	2.35 (1.19)	8.82 (4.27)
HQ	330	1948	528,000	3.96 (1.88)	8.94 (4.07)

Table 5.2.: Overview of different fine-tune models

5.2.3.1. Large Model (FT-XL)

For the large dataset, I have additionally added approximately 300 conversations in addition to the 500 conversations of section 5.2.1. In this case, I want to determine whether adding extra training data to the fine-tune also contributes positively to the results the model will produce.

The dataset was generated as described in section 5.2.1 and not further adjusted manually or in any other way. The keywords were randomly generated by one of the extractor tools. On average, approximately 3 keywords per dataset were produced, resulting in (on average) 8.6 words for the completion.

5.2.3.2. Creative Model (FT-CR)

Instead of randomly selecting one keyword extractor tool, as described in section 5.2.2.2, I have chosen a different approach for this model. With the aim of creating a creative model, the shortest set of keywords of each of the four aforementioned methods was used. For example, if the following keywords of the individual tools were selected:

5. Implementation

Question: How can I help you with?

Answer: I would like to reserve a table for three people next Monday, please.

Keyword Extractor 1: reserve table three people Monday

Keyword Extractor 2: table reserve Monday

Keyword Extractor 3: reserve table

Keyword Extractor 4: reserve table three people next Monday

The keywords from extractor number 3 are selected for the creative model. The answer contains more details than the keywords, as shown in the sample. Furthermore, this is also backed by the average number of words for the keywords being only 2.3. The length of the completion, with roughly 9 words on average, is comparable with the previous model. Theoretically, this causes the model to learn to add information beyond what is contained in the keywords. This is to ensure that an answer can be created even if the patients can only pronounce very few keywords. Knowing that the accuracy of the system might be a little lower because the system adds many details.

5.2.3.3. Manually Annotated (FT-HQ)

In the case of the manually processed dataset, I went through each individual example, including prompt and completion. In total, the adjustment for the dataset took 10 hours (per person) and was carried out by 3 students. Manual adjustments were made in the following cases:

In contrast to the creative dataset, I ensure high-quality keywords for this dataset. This means that all the information in the answer must also be reflected in the keywords or the history of the conversation. Often, some information is missing in the automatic keyword generation, there is too much information or a negation was not correctly recognized despite precautions from section 5.2.2.3. If one of the cases occurs, I have manually adjusted the keywords. For example:

Question: For how many people should that reservation be?

Answer: For five people. Two of them are vegetarians, though.

Keyword Extractor 1: five vegetarians

Keyword Extractor 2: five people vegetarian

Keyword Extractor 3: five people

Keyword Extractor 4: people two vegetarians

In this case, the whole information cannot be extracted from the answer using any of the keywords created. In this example, I would have modified the keywords manually to e.g.:

Manual Keywords: five two vegetarians

This example also highlights the fact that the dataset was created in such a way that the model has to draw information from the history. The word “people” no longer appears in the keywords, but only in the history.

In addition, I revised questions and answers if they were very monotonous or not meaningful. Since the conversations are spoken content that has been transcribed, some questions and answers may have looked like this:

Question: You think?

Answer: Yeah

Question: Hmm

Answer: Hmm

Since the model can learn very little from such conversations, I have modified or deleted these.

As soon as some topics appeared too often, I discarded some conversations in order not to bias the system too heavily.

As mentioned, some keyword extractors are designed to arrange the keywords in order of importance and not in the order they appear in the sentence. To ensure a human-fashioned manner, I rearranged the keywords chronologically for this dataset.

For instance, I adjusted the following’s order to ensure that the formulation of the answer makes sense.

Question: What do you like about this Restaurant?

Answer: I really love the pizza they serve and the very friendly waiter!

Keywords created: love waiter friendly pizza

Keywords adjusted: love pizza friendly waiter

To make the model a little friendlier and more conversational, I have adapted a very monotonous answer in some places, such as:

Question: I have a free ticket for the game tonight, would you like to join?

Answer: Yes

Keywords: Yes

to

Answer: Yes, I would love to!

Keywords: Yes

5. Implementation

I am expecting a very high quality from this model. It should not add many details and also neglect very little information from the keywords. One problem that could occur is that the answers are not very “creative” and the model does not embellish them very much. With an average of about 4 words in the keywords, the model is exactly at the limit of what people with aphasia can usually utter (see section 2.1.1). The length of the answers, with an average of almost 9 words, is no different from the other two models.

5.2.3.4. Prompt Engineering (STOCK-1 & STOCK-2)

Testing revealed countless different prompts for using the use case without fine-tuning but with prompt engineering. It is difficult to say which of the candidates performed best. For this reason, in addition to the models with fine-tuning, I also test the system with two different prompt designs.

Prompt design 1 (STOCK-1):

Create an answer based on the given keywords and the question:

Question: How are you doing?

Answer: I am doing great, how are you?

Question: Awesome, thanks. What have you been up to lately?

Keywords: study working

Answer:

Prompt design 2 (STOCK-2):

The following is a telephone conversation.

Write one sentence as the next response using the keywords: study working

Do not add any additional details.

Question: How are you doing?

Answer: I am doing great, how are you?

Question: Awesome, thanks. What have you been up to lately?

Keywords: study working

Answer:

The two are deliberately different from each other so that I can determine which type works better. Prompt 1 is rather short, but the second prompt is very extensive and detailed. In addition, the keywords are mentioned several times in the second prompt, which should lead to this information being available in any case.

With the different models I want to test which influence quality, size of the dataset and type of dataset have on the fine-tuning. The hypothesis is that the creative dataset will achieve a lower rating than the others, but with a tendency towards “added details”. Furthermore, it remains interesting to see the influence of the manual annotated “high quality” dataset.

Regarding the prompt models, it will be interesting to see how the different types of prompts work. In the first tests, good results were repeatedly achieved, but there are infrequent outliers that produce entirely wrong answers. These observations are subject to the evaluations of the experiment in chapter 7.

5.3. Fine-Tuning

It is difficult to find best practices for fine-tuning LLMs in literature, as they are still relatively new. This is especially true for language models whose parameters are hidden and not released. Moreover, this is often the subject of business models of some companies that do not share their findings with the masses. There are, however, a few recommendations that are readily available. First, there is OpenAI's official documentation on fine-tuning [48]. These are a few guidelines and essential procedures for getting a dataset ready for fine-tuning. There is also a GitHub collection of best practices and lessons learned on the topic of GPT-3 and especially concerning fine-tuning [99]. Alternatively, it is possible to obtain further information from experiences with videos, forum contributions, etc. [18, 68].

A great advantage of fine-tuning with OpenAI is the integration of Weights and Biases (W&B) [135]. Weights and Biases (W&B) is an AI platform that provides tools to monitor, compare, and optimize machine learning models. This enables to track model performance, compare multiple models, and optimize hyperparameters, helping to understand how the models are performing depending on the hyperparameters. Whenever a validation set is transferred for fine-tuning together with the training data, W&B offers accurate visualizations and information on subjects as validation loss or accuracy. The platform also includes visualizations and collaboration tools to help share and analyse model results.

Concerning the selection of the exact model, I use the largest language model available to date: GPT-3 Davinci. Even though some other models were released during the course of the thesis (GPT-3.5, GPT-4), they are not available for fine-tuning or for use via an API. Moreover, these newer models are not reliable in their availability, making them unusable for prototyping or experimentation. Hardware limitations make it difficult to train a different model even with nearly similar scale on a local computer, as was already mentioned in section 5.1.4.3.

Fine-tuning with GPT-3 involves using the pre-trained parameters of the language model to adapt it to a specific task. This is accomplished by training the model using data from the intended task, such as the dataset created in section 5.2. The model is then fine-tuned by adjusting the weights of the model to fit the additional training data. This fine-tuning process allows the model to adapt to the new task while still retaining the general language understanding of the original model.

GPT-3 comes in several models, ranging from the small (350M parameters) to the largest model (175B parameters). Each model is designed to provide different levels of performance for a given task, and the cost structure for each model varies depending on the size and complexity of the task.

5. Implementation

Table 5.3 displays the cost structure and model size. Prices are quoted in USD per 1,000 tokens, with one token corresponding to approximately 0.7 words. A model’s fine-tuning costs $\frac{1}{4}$ of what regular usage costs. A model that has been fine-tuned costs the same amount as the underlying base model in usage. Since the costs for training are correspondingly lower, but the prompts are shorter as a result, it is worthwhile in terms of costs to adapt a model for specific needs.

Modelname	Parameters	Cost Usage [\$/1k tokens]	Cost Training [\$/1k tokens]
Ada	350M	0.0004	0.0016
Babbage	1.3B	0.0006	0.0024
Curie	6.7B	0.0030	0.012
Davinci	175B	0.0300	0.12

Table 5.3.: Cost comparison of different GPT-3 base models

Models with more parameters perform significantly better, as explained and demonstrated in 5.1.4. In order to fine-tune, I therefore focus on the Davinci model.

For the actual fine-tuning, the data is required in JSONL format. For the task I am using the system for (conditional generation), a prompt (the conversation history and last question) and the desired completion from the system is needed (as designed in section 5.2).

Referring to the OpenAI documentation on fine-tuning, there are several recommendations to improve the results. A clear separator should be used for both the end of the prompt and the end of the completion. As a result, the final line of each dataset’s prompt always reads “ $\backslash n$ Answer: $\backslash n \backslash n \backslash n \backslash n$ ”. The completions all begin with a blank and come to an end with the word “*END*”. Furthermore, at least 500 examples should be included in the dataset and the prompts of this training data should have a similar structure as the prompts for the later system. Depending on the dataset, at least 1900 examples of training data are available. In addition, the prompts of the training data are identical to the queries that the model later receives.

5.3.1. Hyperparameters

As already mentioned, apart from the official documentation of OpenAI and some best practice guides, only little information is available for setting the hyperparameters for LLMs. Apart from that, it is generally difficult to optimize the hyperparameter settings in neural networks, as Yang et al. and Kadhimi et al. discuss in detail [143, 74].

The hyperparameters to be configured for fine-tuning the GPT-3 model, include learning rate, epoch, batch size, etc.

- Batch Size: The number of samples to use in one iteration of training.
- Epoch: The number of times the entire dataset is passed through the neural network during training.
- Learning Rate Multiplier: The rate at which the weights of the network are adjusted in order to minimize the loss during training.

According to OpenAI best practice [48], the best results for the task on hand are achieved with 1–2 epochs and a comparatively low learning rate. In order to evaluate which hyperparameters are a reasonable choice for this study, appendix A.18 presents an overview of the validation loss and training loss for a choice of different parameters.

Regarding the epoch size appendix A.18.1, it is evident that although the training loss is minimized with a larger epoch size, the validation loss converges more quickly and more strongly to a minimum with smaller epoch sizes.

A similar picture emerges for the learning rate multiplier in appendix A.18.3. Larger values tend to achieve a better training loss, but the results in terms of validation loss are better with smaller learning rates.

As the illustrations already demonstrate, and after a few more attempts, it turned out that fewer epochs and a lower learning rate lead to marginally better results.

Even though the difference is not significantly high, fewer epochs also correspond to fewer tokens, which yields another benefit. Favouring lower costs for fine-tuning, the setting for epochs was set very low.

When testing some models with several epochs and a higher learning rate, it became clear that the system became much less creative. For example, sentences longer than the number of keywords were hardly ever formed. One possible reason for this is the overfitting to the training data, as indicated by the very low training loss. This is not the purpose of the system, which is another reason to decide against more epochs and larger learning rates.

With regard to batch size, the picture is less straightforward (see appendix A.18.2). It is not possible to make a clear statement whether a high rate or a smaller rate performs better in terms of both validation and training loss. For this reason, a value of two was used, which shows a relatively low training loss but also a purely low validation loss. Finally, the hyperparameters are composed as follows:

- Epochs: 1
- Learning Rate Multiplier: 0.05
- Batch Size: 2

5. Implementation

Initially, the dataset was created by randomly selecting a few QA-Pairs out of the pool of conversations, to save tokens and costs for fine-tuning. However, after adjusting the hyperparameters and cutting costs by more than 50 % (mainly due to the lower epoch size), every QA-Pair of the conversations could be used.

5.3.2. How Fine-Tuning Works With OpenAI

For training the large language models, as described in section 5.1.4.4, powerful graphics cards are necessary. This hardware is only available in limited quantities worldwide and is also extremely cost-intensive. Apart from that, trained models from OpenAI (their weights and biases) are not publicly available. Training on local devices is therefore impossible. The training for GPT-3 hence runs on OpenAI's servers. Both the training data and the trained model are stored on their server after fine-tuning and can be easily retrieved like any other model on their servers.

Once the dataset is in the desired shape, fine-tuning with the OpenAI CLI (Command Line Interface) is the next step. First, the data is transferred to the desired format (if not already done) and some final tweaks are made using the `prepare_data` API. In order to use the data preparation tool, the format of a prompt and a corresponding completion pair must be adhered to. The tool then checks whether all prompts and completions end with the same ending and begin in the same way. In addition, a whitespace is inserted at the beginning of each completion, as this supposedly improves the results. Finally, a JSONL document is generated in which each prompt and completion pair forms a line.

```
OpenAI tools fine_tunes.prepare_data -f <TRAINING_DATA>
```

In addition to training data, validation data can also be turned into the desired format. Afterwards, another command is sufficient to initiate the fine-tuning:

```
OpenAI api fine_tunes.create \  
  -t <TRAINING_DATA> \  
  -v <VALIDATION_DATA> \  
  -m davinci \  
  --suffix <CUSTOM_NAME> \  
  --n_epochs 1 \  
  --learning_rate_multiplier 0.05 \  
  --batch_size 2
```

Depending on the size of the training data and the number of epochs, the training takes between 15 minutes and a few hours. The result is a model name of the fine-tuned model. This can be retrieved and used via the API, as described later in section 6.4.4.

Instead of interacting with the system via prompt like e.g.:

```
The following is a telephone conversation.
Write one sentence as the next response using the keywords: soccer
Do not any additional details.
Question: What are your favourite hobbies?
Answer:
```

The request can be made directly, as in the structure of the training data, without explaining to the model what it should do.

```
Question: What are your favourite hobbies?
Keywords: soccer
Answer:
```

Table 5.4 illustrates the costs and names of the different models created.

Modelname	Dataset	Costs	Modelname (on the OpenAI server)
FT-XL	Large	\$ 8.47	davinci:ft-speech-bci:large-2023-03-17-08-11-39
FT-CR	Creative	\$ 5.57	davinci:ft-speech-bci:creative-2023-03-16-17-21-33
FT-HQ	HQ	\$ 3.60	davinci:ft-speech-bci:hq-2023-03-20-14-26-28

Table 5.4.: Overview of characteristics of the created fine-tunes

5.4. Reproducibility

The following is a brief guide to the steps necessary to fine-tune a model and create a dataset accordingly. It is worth noting that individual aspects are described in more detail within the respective code, which is well documented and split into sections. The following explanations only serve as a rough overview of which steps are necessary for reproduction.

Creating a Dataset To generate a dataset of conversations for fine-tuning, the following steps to run the *Fine-Tuning/create_dataset.ipynb* steps must be followed. The code and all subsequent information are found in the repository explained in appendix A.19.2. The individual conversations that are the basis for the dataset should be in a format that contains alternating responses from the two conversation partners (such as in section 5.2.1). In addition, each statement should be preceded by a string (e.g. the name of the conversation partner) followed by with a colon. The exact content of the string is irrelevant. The majority of conversation datasets, however, are structured in such a way that they contain the name of the conversation partner or simply “A:” and “B:”. If this format is not available, the dataset can still be used with a slight adjustment of the code in the *slicer* function.

5. Implementation

The implemented functions recognize whether there are empty, incomplete or incorrect lines and skip them.

The files of the conversations should be in the *.txt* format and be stored in the folder *Fine-Tuning/txt_conversations*. All conversation files that were used in the dataset creation can be found in *Fine-Tuning/all_conversations*.

One thing to be decided is whether the dataset should be built up creatively, or whether a randomly selected keyword extractor should generate the keywords. To accomplish this, either “Alternative CR” or “Alternative RND” must be selected in the loop over all conversations.

No further adjustments are necessary, all the requirements and imports are installed as soon as the notebook is running. During the creation of the dataset, the user is informed about the current status and information messages (e.g. empty line detected, etc.) via prints.

On a very high-level code proceeds as follows:

1. The data is pre-processed and cleaned
2. Keywords are extracted from the answers
3. Together with the history, answer, and the keywords, a prompt and completion pair is formed

Subsequently, all generated *.json* output files are stored in the folder *Fine-Tuning/output_files*. These are:

- The generated dataset (according to section 5.2)
- The dataset split into 3 equally large different files (for validation, smaller training data, etc.)
- The differently generated keywords of the individual methods (*keyword_examples.json*)
- The conversations that were imported (*all_text.json*)

Fine-Tuning To fine tune a model from this dataset, the steps under section 5.3 can be followed. To simplify the process, the Jupyter notebook *Fine-Tuning/openai_fine_tuning.ipynb* can be utilized. In this code, only the filenames need to be adjusted according to how the training and validation datasets are named. The resulting fine-tune files and process steps are presented via the terminal.

Topic-Detection There are many ways to implement topic detection. One possible implementation can be found in *Fine-Tuning/topic_detection.ipynb*, which works as described in section 5.2.1.

The Jupyter Notebook is self-explanatory and analyses all topics of conversations that are stored in *Fine-Tuning/txt_conversations*.

5.4. Reproducibility

Two resulting json output files are saved in the *Fine-Tuning/output_files* folder. Firstly, a file *all_topics.json* is saved, containing all analysed topics and any additional metadata. Furthermore, the *all_relevant_topics.json* file contains only the names of the topics that have a relevance greater than 50 %. For the API call in the notebook to work, however, an API key for meaningcloud must be used.

How to insert the API keys is described in appendix A.19.2.

6. Experiment

In order to evaluate which of the models from chapter 5.2.3 perform best, all models were tested within an experiment. The following sections shed light into details about how the experiment is structured, how it works, which technologies it is based on and how it was conducted. Furthermore, this chapter establishes the foundation for the detailed evaluation in chapter 7. Ultimately, the evaluation of the experiment is intended to answer the question of which system is best suited to compensate for speech impairments, thus answering the research question. Please note that in the following sections, the terms user and participant as well as use case and scenario are used interchangeably.

6.1. Core Aspects

Various metrics are available to test the performance and reliability of the models created in chapter 5. There are numerous potential automatic evaluation methods, as will be explained in chapter 7.2. However, since the developed system is intended to support speech impaired humans in their daily lives, it is of great importance to ensure that the models are evaluated by humans. For this reason, the present chapter focuses on the experiment, which is designed to be carried out by humans and thus to evaluate the models accordingly.

The idea of the experiment is to put the users in different fictional scenarios. The participant is presented with a brief description of the scenario in which they find themselves, as shown in Fig. 6.1 (A).

In each scenario, the experiment participant is required to provide answers to certain questions ((B) in Fig. 6.1) being asked. Only one question is asked at a time.

Based on the answers, follow-up questions are asked in turn, creating a conversation such as in daily scenarios. More about the different scenarios itself in section 6.3 and their technical implementation in section 6.6.

However, participants are required to provide two responses: One is the response in a comprehensive sentence, as they would typically answer ((C) in Fig. 6.1). Secondly, the same answer must be submitted using keywords only ((D) in Fig. 6.1).


For example, as in the figure above illustrated:

Question: What are your hobbies?

Answer: I love playing basketball in my club.

Answer in keywords: love basketball club

6. Experiment

 **Your next task:**

Read the scenario instructions in the chatbox

Read the question in the chatbox

Type in your answer in natural language and in keywords.

Click submit

E

Completion Status:

(0 %)

Feel free to skip the scenario if the questions are confusing

Skip the Scenario

1. Answer (one sentence maximum)

I love playing basketball in my club.

C

2. Answer in Keywords (separated by a space)

love basketball club

D

What are keywords?

◀

✔ Submit

🔄 Reload Chat

F

Chatbox

Somebody asks you about hobbies, talk with her about your hobbies.

▶ Question: Hi! Let's chat about hobbies. What is yours?

A

B

How good did the answer express your intention from 1 (wrong) - 5 (perfect)?

☐ 1 Wrong

☐ 2 Bad

☐ 3 Okay

☐ 4 Good

☐ 5 Perfect

Figure 6.1.: Interface design of the experiment

With this input given, the system uses only the keywords provided and the conversation history to generate the answer. For example, with the input provided above, the system generates the following output:

INPUT:

Question: What are your hobbies?

Keywords: love basketball club

OUTPUT:

Answer: I love basketball. I am a member of a basketball club.

The user's comprehensive answer in natural language serves only as a reference for a later, automatic evaluation (see chapter 7). After the system generates the output, the participant needs to evaluate the generated answer. It needs to be evaluated how well the system used the keywords, that the participant provided to generate the answer. Participants have a total of five different ways to evaluate the system.

1. Wrong
2. Bad
3. Okay
4. Good
5. Perfect

Sticking to the example, the generated answer is *Good* but not perfect since it is *Slightly different* to the provided answer from the participant. Hence, as illustrated in Fig. 6.2 (A), the rating *Good* is chosen.

The screenshot shows a web interface for evaluating a chatbot response. At the top, it says "Chatbox" with a speech bubble icon. Below that, a text box contains the instruction: "Somebody asks you about hobbies, talk with her about your hobbies." Below this, a "Question" is displayed: "Hi! Let's chat about hobbies. What is yours?" followed by the "Answer": "I love basketball. I am a member of a basketball club." The main part of the interface is a large white text area. Below this, there are two sections for evaluation. The left section, titled "How good did the answer express your intention from 1 (wrong) - 5 (perfect)?", contains five radio buttons: "1 Wrong", "2 Bad", "3 Okay", "4 Good" (which is selected), and "5 Perfect". A red circle with the letter "A" is positioned below these buttons. The right section, titled "? What was wrong with the answer ?", contains six radio buttons: "Slightly different" (which is selected), "Too brief / short", "Missed details", "Added details", "Wrong information", and "Totally different". A red circle with the letter "B" is positioned to the right of these buttons.

Figure 6.2.: Rating and detailed rating of an answer in the experiment

6. Experiment

If the evaluation is “worse than perfect”, it must be specified exactly what was wrong with the answer, as indicated in Fig. 6.2 (B). Ultimately, the aim is not only to find out whether a model performs well or not, but rather in which areas it can be improved. By evaluating more closely, it can be seen what the model tends to do, whether too many details are added, neglected or answers are formed only marginally differently. Pre-defined buttons for these detailed ratings pop-up if the rating is worse than perfect. The options, which originate from the most frequently occurring errors of the models, are:

- Slightly different: *The generated answer differs only marginally from the user’s own answer. The statement remains the same.*
- Too brief / short: *Even though the statement of the generated is the same as the entered answer, the generated answer is very short and concise.*
- Missed details: *The generated answer does not contain some details which were passed in the keywords.*
- Added details: *The model invents details for the generated response that are not specified in the keywords.*
- Wrong information: *The generated answer mixes up information from the keywords and gives a wrong answer.*
- Totally different: *The generated response does not match the keywords at all, the response is out of context.*

After the user submits the rating, the conversation automatically continues with a new question in the same scenario. After three answers in one scenario, the participants are presented with a new scenario in which they must again answer the questions according to the above-mentioned scheme and rate the created answers. As soon as the participant answered ten questions and ten corresponding ratings, the experiment is completed.

With this approach, a score, how well the system interpreted the user’s keywords and also what mistakes the model make when the answers were not satisfactory for the user, can be calculated for each model.

6.2. User Guidance

Since this experiment is more difficult to take than normal surveys, various precautions have been taken to provide the participants with the best possible guidance and assistance before and throughout the experiment.

Having clear descriptions and user guidance is essential for achieving reliable and high-quality results without any misinterpretations. By providing detailed instructions to participants, it ensures they understand the experiment correctly, and each trial is conducted the same way.

Clear descriptions and user guidance also increase the credibility of the experiment by providing necessary details to replicate the experiment and having a consistent set of data to work with.

Firstly, before the start of the experiment, a step-by-step guide must be clicked through.

Figure 6.3.: Step by step explanation before the start of the experiment

This contains a picture-by-picture explanation of what the participant can expect and what they have to do, as an example illustrates in Fig. 6.3. Only after these instructions have been fully clicked-through, the user is forwarded to the main screen of the experiment. Thereafter, however, these instructions can be opened again at any time. If questions arise during the experiment, there is also an FAQ section in which all common questions are explained.

In addition, as shown in Fig. 6.1 (E), the “Next task”-section is located in the top-left corner. This is a precise description of which step the user must take next. As shown in the picture, these next steps contain various colour codings. The different input masks of the participant (answer, keywords, rating, etc.) also contain a colour coding. Accordingly, the next task is always highlighted with the same colour as the input mask for this task. This way, even with poor reading, hardly any mistakes can occur or questions arise as to what needs to be done next. Furthermore, each text and input field contains an exact description as well as a placeholder of what must be entered in this field.

6. Experiment

Furthermore, there are also various drop-downs, such as what exactly is meant by keywords, which can be opened and read during the experiment if questions arise.

As soon as an error occurs, for example if the user wants to rate an answer but has not entered an answer, informative error messages appear (see section 6.5.4).

Furthermore, there is always a button to restore the last state of the chat before any error or other mistake appears (see Fig. 6.1 (F)).

In the upper-right corner resides a status bar and a button to skip a scenario as shown in Fig. 6.4. If the questions of a scenario do not make sense or the user does not want to answer them, a button to skip the current scenario and the current question is always available.

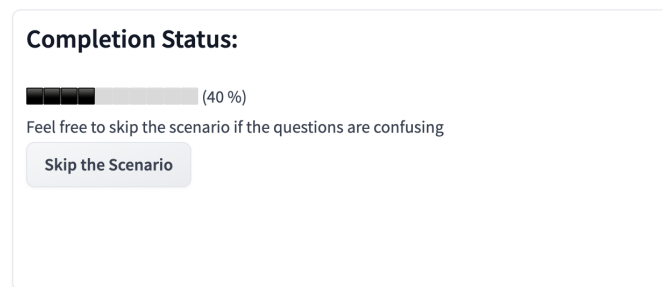


Figure 6.4.: Progress bar and skip-button in the experiment

Finally, the user is always informed about the current progress of the experiment through a progress bar, which fills up for each evaluation submitted by this user and ultimately turns green on 100 %.

6.3. Use Cases

There are a total of eight different use cases in the experiment. These are designed to correspond to daily scenarios, since this is where DialoKEY should support people with aphasia. Four of these scenarios are designed for various services and rather shorter answers. Here, the instructions are also defined accordingly to a specific outcome. For example, a clearly defined goal is: "Reserve a table in a restaurant". The user then tries to pursue this goal with his answers. Accordingly, the questions are also posed in such a way that the users achieve their goal.

In contrast to these defined scenarios, the other four use cases force an open conversation, without a predefined goal. An example of an open scenario would be, among others, a conversation about the participant's hobbies. Here there is no clearly defined goal, but rather the model's abilities in open situations is tested.

The mix of the different types covers the majority of the conversations that DialoKEY is intended to enable. On the one hand, the system could be used to make appointments by phone, for example, and the stranger on the other end of the phone would not notice that he or she is talking to a speech-impaired person. On the other hand, conversations with family and friends could be simplified and be more fluent. Both together enable the patient to better participate in social life again.

A scenario always begins with a brief description of what the users can expect and what they are supposed to do (Fig. 6.1 (A) and (B)). This is followed by a question to introduce the setting (see table 6.1 for an overview). This initial question, unlike all the following questions, is the same for every user. The participant's answer is then dealt with individually and a follow-up question is asked accordingly. This results in a different course of conversation for each participant and each conversation. For example:

Scenario: Book a Table

Question: Hello, this is Giovanni's Pizza and Pasta Restaurant, how can we help you?

Answer: I would like to book a table for six people tomorrow evening.

Next generated Question might be: Sorry, tomorrow we do not have a table for six people, how about the day after?

If the user had answered differently, the course of the conversation would be different:

Scenario: Book a Table

Question: Hello, this is Giovanni's Pizza and Pasta Restaurant, how can we help you?

Answer: I would like to reserve a table.

Next generated Question might be: For how many people and what date and time would you like to come in?

I will elaborate more on the exact implementation in section 6.6, but an overview of all scenarios including their description and first question is illustrated in table 6.1.

However, a user will never see all eight different scenarios. Each user is assigned a randomly selected scenario as soon as the experiment starts. After three questions and three answers, the next, again randomly selected, scenario is automatically started. The random selection of scenarios considers which scenario has already been carried out.

In addition, the system stores which scenarios are similar to each other. This means that after a "Book a table" scenario, the scenario "Make an Appointment at the Hairdresser" is not displayed. These two scenarios are similar in the nature and goal of the conversation. Thus, after a "Book a table" scenario, an open conversation with an old friend or a discussion about hobbies is more likely to appear. The new scenario then starts with the introduction to the new setting as well as a first question. Since ten answers must be given to complete the experiment, a user will see a maximum of four different scenarios.

6. Experiment

However, there is also a special case. In some cases, the structure and subsequent questions of the scenarios may make little sense because the user’s answers were generated inappropriately or incorrectly. In this case, users are free to jump to the next scenario at any time by clicking on a “Skip this scenario” button.

Scenario	Type	Description	First Question
Book a table	Defined	You are calling at a restaurant and want to reserve a table.	Hello, this is Giovanni’s Pizza and Pasta Restaurant, how can we help you?
Hairdresser	Defined	You want to schedule an appointment with a hairdresser on the phone.	Hello at Cutting Edges Hairdresser, what can I do for you?
Doctor	Defined	You call the doctor because you don’t feel well. Make up something that might be wrong and describe it.	Welcome to Dr. Peppers Medical Clinic. I am Lisa, what can I do for you?
Order Food	Defined	You are calling a restaurant to order food. Place your order.	Welcome to the Restaurant Tikka on Wheels. How can I help you?
Hobbies	Open	Somebody asks you about hobbies, talk with her about your hobbies.	Hi! Let’s chat about hobbies. What is yours?
Calling an old friend	Open	An old friend calls you. Talk to him about what’s new in your life lately.	Long time no speak, how are you doing?
Customer Service	Open	You call a customer service hotline. Describe a problem you have with a product.	This is the Mississippi online shop customer service. How can I help you?
Auto Shop	Open	Something is wrong with your car, so you’re calling an auto shop to get it fixed.	Hello and welcome to the Wheely Great Car Repair Shop. What’s wrong with your car?

Table 6.1.: Overview of different scenarios including first question and description

6.4. Technology Stack

Fig. 6.5 highlights various technologies and tools that were used to implement the experiment and are briefly presented in the subsequent chapters. The different tools can be divided into implementation and hosting. The former tools refer to the logic of the experiment and the entire front- and backend, which was created in Python with the help of various tools described as follows. The technologies used in hosting refer to the deployment of the application in a cloud environment that is accessible to all participants.

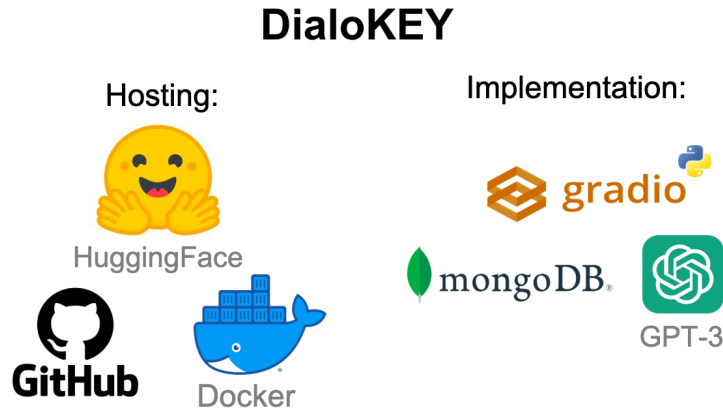


Figure 6.5.: Technology stack used for DialoKEY

6.4.1. Gradio

Gradio is an open-source software package based on python, designed to facilitate the exploration and analysis of scientific data in machine learning [5, 60]. It provides an intuitive graphical user interface for visualizing and exploring data, allowing to quickly demo a machine learning model with an easy-to-use web interface. Gradio is therefore also ideally suited for designing the user interface for the experiment and at the same time guaranteeing the functionality in the background. Therefore, it was used to implement both the front-end and the back-end of the experiment. All buttons, text fields, radio buttons, etc. were implemented via Gradio, as well as the individual API calls, error handling, logic and other functionalities mentioned in section 6.5. Additionally, CSS (Custom Style Sheets) are supported with Gradio too. The resulting Gradio web application can be used on multiple devices, including mobile devices.

6.4.2. Hosting Technologies

Applications written with Gradio can be hosted locally without limitation. However, a 72-hour limit applies to applications that will be hosted on a publicly accessible server. To combat this issue, Huggingface Spaces was used. Huggingface is a natural language processing library that provides a suite of tools for text pre-processing, model training, and model deployment.

6. Experiment

It utilizes a variety of deep learning techniques such as transfer learning and language modelling to enable the development of state-of-the-art natural language processing models [71]. Huggingface Spaces, on the other hand, is a way to host a wide variety of machine learning applications. The hosting is free of charge up to a certain degree. A Docker container [42] with 16 GB RAM and 8 CPU-cores is provided. This is a virtualized environment, running on top of a host operating system, that can be used to package and run applications and services in a secure and isolated environment. Applications running on this container can be generated and shared with a custom defined URL. Huggingface Spaces natively supports the integration of Gradio applications so that they can run on the servers without prior configuration of the Docker file [42]. Lastly, GitHub is a version control service that allows to track changes in code, review, and monitor overall progress [57]. The integration of the GitHub repository into the Huggingface Spaces project, enabled the ability to keep track of code change and facilitate smooth code deployment.

6.4.3. Database

To store the data of the individual participant's rating in the experiment as well as logging of important parameters, a database was utilized. The choice of database fell on a NoSQL database. There is one main reason for this: The data model changes relatively often during the experiment as well as in the preparation. NoSQL databases do not require constant adaptation of the structure and schema and can deal flexibly with other data structures [79]. MongoDB is therefore very suitable for the use case [93]. The Python library of MongoDB that makes it straightforward to work with MongoDB documents in combination with Gradio. However, it cannot be used in combination with a Docker container on Huggingface Spaces. Therefore, the use of an API was resorted to.

Thus, the participants' ratings and their generated answers and keywords are sent to the corresponding database in MongoDB via the interface. Additionally, logging parameters such as the UUID (Universally Unique Identifier) of the users, the attention check, and server specific messages are logged to the database (see section 6.5). A detailed explanation of all logged parameters is highlighted in section 6.5.3. More specifically, MongoDB Atlas is used so that the database is hosted on MongoDB servers. This means that there is no need for separate server management, and the resources can be used free of charge for the data volumes in the experiment.

6.4.4. Language Model

As highlighted in detail in chapter 5.2.3, the language model GPT-3 was used for the implementation of the sentence generation. This model is used as a baseline for the fine-tuned models as well as for the stock models, which are all addressed via a prompt. The language model is also used to generate the questions, which will be described in more detail in the following chapter 6.6. Technically, the results of the model are fetched via an API. More precisely, the Create Completion API of OpenAI is addressed [13].

The calls look very similar, but differ fundamentally in prompt and model, thus leading to different results. A request to the API will always have the following structure and parameters.

```
response = openai.Completion.create(
    model=<MODELNAME>
    prompt=<PROMPT>,
    temperature=0.7,
    max_tokens=256,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0,
    stop=["Answer:", "Keywords:", "Question:", " END"]
)
```

As it is evident, the only variable parameters are the model and the prompt. The model describes the name of the model to be used, which is either a fine-tuned model or a stock model (see section 5.2.3). The prompt for the fine-tuned model is the same as the prompt with which it was trained. That is, the same prompt structure that contained the dataset introduced in section 5.2:

```
Question: How are you doing?
Keywords: good you
Answer:
```

The prompt for stock models varies depending on the prompt used (section 5.2.3) and is based on Zero-shot learning (see section 5.1.4).

The *temperature* describes how creative or random the response of a model should be. A smaller value means that the responses are more monotonous and uncreative. However, when experimenting with some examples, this value does not seem to have too much influence on small texts similar to the conversations used.

Top_p is an alternative to *temperature*. Here, you can set a percentage of how many percent of the potential next words should be considered. Again, for the use case on hand, there were no drastically different results when this value was changed.

max_tokens describes the maximum number of tokens that may be used for request and response. One token is comparable to about 0.7 words. Since the texts were rather short, this value is not of great importance.

Frequency and *Presence penalty* are also both intended for longer texts and penalize words that occur more than once either in the answer or in the previous text. The higher the value, the more different words occur.

Stop contains an array of words that stops the model from generating further text. If one of these words occurs in the generation of a text, the model stops any further creation before the word is reached.

6. Experiment

This is important for the proposed use case when, both in Zero-shot learning and sometimes in fine-tuned models, the model continues the sequence of Question, Answer, Keyword. This leads to not only an answer being generated, but often immediately another question, keyword, and answer. The word “END”, as described in section 5.3, indicates the end of an answer in the dataset. The fine-tuned models also create this word at the end of an answer, i.e. the model recognizes at this point that the answer must end.

For comparability between models, and to avoid introducing additional variance, all parameters are the same for all models and queries.

6.5. Functionality

The following chapter provides information on the functional implementation of the experiment. It sheds light on how the experiment is structured in the background to ensure that it can guarantee an accurate solution to the research question.

6.5.1. Gradio State

Gradio applications have a state, which can be equated with a session of web applications. Various data can be stored in the state and are persistent for a user. This data is only stored for one user at a time. Different users receive different states accordingly. As soon as the application page is reloaded, the state is also reloaded and cleaned.

Among other things, the following events are stored for a session during the experiment:

- **UUID:** *At the beginning of an experiment, each participant receives a UUID.*
- **WorkerID / Name:** *Especially crowd workers (see section 6.8) have a WorkerID, which is entered at the beginning of the experiment and stored in the state. If the experiment is not done by a crowd worker, a name of the participant is stored.*
- **The conversation history:** *To generate answers and questions, the previous conversation history is stored for each scenario.*
- **Scenarios:** *To ensure that no scenario occurs twice, the scenarios that have already been presented to the user are stored.*
- **Attention Check:** *The information whether a user has passed the Attention Check (as described in section 6.5.5).*
- **Progress:** *The current progress of the experiment indicating when the participant has completed the study.*

In addition, a variety of other information is stored in the state. These are mostly smaller details to ensure the functionality of the experiment. In order not to exceed the scope, these are not mentioned separately in the above list. For more detailed information, please refer to the code as further explained in section 6.10 at any time.

6.5.2. Model Comparison

As already introduced in section 6.1, a participant must enter both the answer and the keywords of the answer to each question. The keywords and the previous course of the conversation are transmitted to the language model. The language model is then supposed to generate an answer from the keywords and the history of the conversation. Both the fine-tuned models and the models used with a prompt are addressed via an API (see section 6.4.4). To compare which of the models performs best, one of the five models is randomly selected for each response generation.

The user neither knows that different models are used to generate the sentences, nor which model is used in the background. This ensures that the ratings of the individual models are objective and that each user gives each model an average of about two distinct ratings. Thus, a subjective evaluation of a participant (no matter in which way) cannot influence the evaluation of a single model in total. The subjectivity of individual users is always present in all models, which leads to objectivity over a certain number of evaluations.

Since the previous conversation history is transmitted with each request, it could be argued that the models learn from this history and adapt themselves via Few-shot learning. However, this effect is negligible. This can be clearly seen in the different characteristics of the “Detailed Ratings”, as explained in more detail in section 7. Furthermore, the history is not passed in such a way that it could be utilized as a Few-shot learning. The history no longer contains the keywords of the answers that have already been given, but only questions and answers. Therefore, the system cannot learn from the conversation history how answers are created.

6.5.3. Logging

Since the state only exists for one session, important information must be persistently stored in a database. It is important that all necessary information is saved for later evaluation. To make the logging process as fast as possible, only information that is unavoidably needed is stored. Data that can be generated afterwards, will not be created and stored during the experiment. Logging is done via an API in MongoDB (see section 6.4.3).

In the following cases, the logging function is triggered with different data:

- When passing the answer and keyword pair
- When the response of the model is rated
- When the attention check is performed

6. Experiment

Information that is always stored, regardless of the logging trigger:

- The UUID
- The WorkerID / Name
- The timestamp
- The previous course of the conversation

The following data is logged when the answer and keyword pair are transferred:

- The user's answer in natural language
- The keywords entered by the user
- The response created by the model
- The model that created the answer
- The runtime of the model

If the model's created answer is rated, the following parameters are stored:

- The rating
- The detailed rating (if any)

During an attention check, which will be elaborated in 6.5.5 it is only stored, whether it was passed or failed. Once a participant has completed the experiment, the generated code (indicating the success of the attention check) is stored in combination with the participant's UUID. With this data, later traceability, allocation, and evaluation of the experiment is seamlessly possible.

6.5.4. Robustness

In order to ensure that the system is robust and that functionality is guaranteed despite participant errors, a number of error messages and features have been included. An example error message is shown in Fig. 6.6.

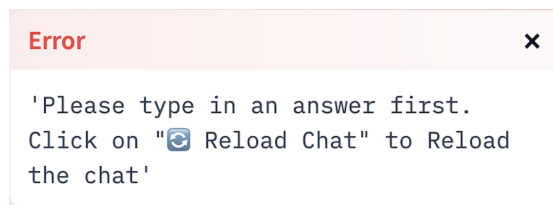


Figure 6.6.: Example error message in the experiment

The error messages ensure that the system functions seamlessly. After each error, the user may click on the “Reload Chat” button to restore the data and re-enter the input (see Fig. 6.1 (F)). During the error and until the “Reload Chat” button is clicked, the user cannot make any further transmissions. Sources of error that have been identified and to which the system responds are:

- Clicking on a rating without having entered an answer that could be rated
- Clicking on submit an answer, but the answer or keyword field is empty
- If an answer and keyword pair is submitted, but no question has been generated (that is, if the last answer has not yet been rated)
- If no WorkerID/Name is entered to start the experiment

6.5.5. Attention Check and Fraud Detection

Ensuring that the participants carry out the experiment conscientiously, an attention check was introduced. This is to ensure that users who click through the experiment without a second thought are excluded from the evaluation. During the experiment, after the third answer, the user receives a request in the design of a normal question not to enter an answer, but to click directly on the rating “Okay”. If the user follows the request, the experiment continues normally and a new question appears. If the user clicks on another rating instead or possibly enters an answer, as it would otherwise be the case if the attention check were not in place, the experiment also continues as normal. In the background, however, it is noted whether the user has passed or failed the attention check. At the end of the experiment, a code is generated and connected to each UUID. This code reveals, whether the attention check was passed or not. Results with a code revealing that the attention check is failed, are not taken into consideration for the evaluation.

6.6. Implementation of Use Cases

The experiment is designed to reflect everyday scenarios and make them as realistic as possible. To ensure that the experiment runs smoothly and quickly, the questions for the participants in the conversations must not be posed manually, but automatically. A big challenge was to achieve realistic conversations without having to manually monitor each users’ conversation. In addition, every user reacts differently to questions, which further complicates a rule-based flow of the scenarios and questions.

During the set-up of the experiment, several alternatives crystallized to circumvent the aforementioned problems. A simpler alternative to avoid having to respond to the participants’ answers would be, to change the scenario after a question and an answer and ask an unrelated, different question in each turn. This way, only a conversation consisting of one QA-Pair is ever created. However, this is not very close to reality and also results in the system not having to use information from the context of the conversation.

6. Experiment

Another alternative would be to ask predefined questions during the course of the scenario. This means not responding to what the user says and asking a new (maybe unrelated) question after each answer, based on rules. One could argue to search for various keywords in the users' answers and ask the next question based on that. However, the possibilities to answer are so enormously diverse that it is almost impossible to cover all possibilities with rules. Proceeding according to this alternative leads to the fact that some conversations might not be logically structured and make no sense. For example, a conversation could look like this:

Question: Hello, this is Giovanni from Pizzeria Amore, how can I help?

Answer: Please excuse me, I have called the wrong number.

Question: How many people do you want to book for?

So, this approach does not lead to a satisfactory result either.

Therefore, a variant was chosen in which the participants' responses are always considered, resulting in a realistic conversation consisting of several QA-Pairs. Each conversation is therefore different, and no two are alike.

Behind each scenario is a description of what the conversation is about, what the possible questions are and, if available, what the goal of the conversation is. This description is given to a language model. In addition, instructions are given that the language model should ask all necessary questions that are present in the description or that are necessary to achieve the goal. Furthermore, the conversation history of what has already been said so far is passed to the language model. In this way, it can respond individually to the conversations and answers and knows whether a question still needs to be asked or has already been answered within the conversation.

Another advantage is that the language model only asks one question at a time. Furthermore, it is also possible to deviate from the guideline, but the model is still able to ask further questions. For example, if a user in the "Book a table" scenario wants to place an order for food, the model adapts accordingly, and the conversation is maintained. As a language model, OpenAI's GPT-3 Davinci stock model was used. A description given to the model looks similar to this:

You are the waiter of a Restaurant called 'Giovanni's Pizza' and get a phone call.

Ask all the questions needed to make a reservation for the customer including: The date, the name, amount of people, and if they would like to eat from the menu or buffet.

Be friendly, polite, and talkative. Ask one question at the time.

Each user must answer a maximum of three questions from a scenario, but each scenario contains at least five different questions. Therefore, it will under no circumstances run out of questions to ask.

The start of the conversation is the only thing that remains the same for all experimenters, thus at least all conversations in a scenario follow a similar direction. This makes comparability between different participants in the experiment easier. In addition, there is no conversation history at the very beginning that needs to be addressed when asking the first question. After the initial question is answered, the other questions are generated according to the procedure mentioned above.

An overview of the instructions for the user cases can be found in appendix A.3.

6.7. Pre-Testing Interviews and Optimization

The experiment as shown in Fig. 6.1 was optimized with several iteration stages. After a rough draft, a pre-study was carried out with participants from different areas. Preference was given to participants from fields and professions that do not necessarily deal with the topic of computer science. This serves the purpose of testing the experiment for its comprehensibility in terms of design, explanation, and the way it works in the background without any pre-knowledge. In total, a qualitative interview lasting about 30 minutes was conducted with 16 different people. Before the start of the interview, the people were not briefed on what the experiment was about and what to expect. So, most of the participants were unbiased and did not know anything about the topic. Except for some members of the research group Neuroscience of the University of Vienna, who were familiar with the topic. In addition, participants were asked to share their screen. This allowed me to see at which steps the participants had problems, did not know where to click, or had difficulties in general. Since they shared the screen, the aspects could be identified even if they were not mentioned verbally. The so-called pre-study experiment was divided into three different sections.

1. The participants had to say what they were going to do in the following, based on the description of the experiment without seeing the interface. This was done to check whether it was clear what different terms meant, whether the description was unambiguous, and whether the general idea of the model and the procedure was clear.
2. The participants carried out an example scenario. It was checked whether the participants knew what they were supposed to evaluate, what they were supposed to enter for answers, what scenarios are, what keywords are, etc. In addition, an error was artificially generated on instruction to see whether the error handling is generally understandable.
3. Finally, further suggestions for improvement were collected, apart from the points already mentioned.

An overview of the questions asked/ the interview guide can be found in appendix A.6.

6. Experiment

The interface and procedure were adapted and optimized in various places during this phase. An overview of which aspects were discarded, added or changed is summarized in the following sections:

Experiment Description At the beginning, a descriptive text was used to describe the experiment. Once this was confirmed, the experiment started. After the first interviews, the interview participants asked for a visual representation that prepared them for what would happen in the following experiment. A GIF was used instead, which visually explains the subsequent steps and tasks of the participant in the experiment. However, it quickly became clear that it is difficult to find a suitable speed that is good for all participants in the experiment. Since the participants have different levels of knowledge or read at different speeds, the GIF was exchanged again. Eventually, the experiment contains images with descriptions that can be clicked on manually, to get to the next image. Only when all images have been viewed, the participant is forwarded to the start of the experiment.

Scenario Selection At the beginning, the participant could manually select which scenario they want to work on. During the interviews, however, it became clear that the participants were rather confused about what the different scenarios meant or when they should change one. Therefore, the scenarios are now randomly selected for each user. Furthermore, after three questions and answers, the participants switch to the next scenario. There are two reasons for this. Firstly, this way you get answers to several scenarios from one user. Secondly, scenarios can become very monotonous the more advanced they are. For example, question three of a scenario could be: “How often do you play tennis?” And question four could be: “Did you often play tennis when you were younger”.

Skipping Scenarios To prevent another potential source of error, a button to skip a scenario was added. If the system misinterprets and incorrectly renders the answer, the conversations become very inconsequential and illogical. This may lead to the questions being asked no longer making sense and to the answers being interpreted absolutely incorrectly. For this reason, the participant can simply jump to the next scenario by clicking on the button.

Discarded “Own Scenarios” Moreover, a function that allows participants to create their own scenario was discarded. The original idea was to be able to use even more different scenarios to evaluate the models. Users would have to come up with their own questions based on their answers to simulate a conversation. However, this increases the complexity of the experiment enormously. During the first pre-tests, it quickly became evident that the participants tended to see this feature as additional confusion and unnecessary complexity. For this reason, the function was removed, and the focus was placed on automatic question generation in the system to allow users, in turn, to focus on their answers.

Furthermore, the description that a scenario is now over was optimized during the interview. This way, the users are prepared and informed if the scenario changes, and they know what is being asked to do next.

Next Tasks During the interviews, it was noticeable that users did not know exactly what the next steps were for their first answers and ratings. To prevent this, I have added a field called “Your next task”. This always describes what the participant has to do next. For example: Enter an answer or rate an answer. Moreover, these instructions are colour coded so that they have the same coding as the elements where the next step is to be done. For example, the rating form is coloured blue, and the task “Rate the answer” is coloured the same blue. An example is shown in Fig. 6.1 (*E*) and other instructions for the next tasks are summarized in appendix A.5. However, it has been noticed that participants overlook these descriptions, so the font size has been drastically increased, and an orange frame adorns these instructions. The learning curve for users is very steep, though, so they read up on what to do in the first one or two examples, and the experiment is self-explanatory from there on.

Discarded Keyword Suggestions Since not everyone understands what is meant by keywords, I created dynamic suggestions for keywords at the very beginning. This means that for each answer entered by the participant, potential keywords were suggested. However, it quickly became clear that this strongly influenced the users. As a result, the users did not create the keywords themselves, but simply adopted the system’s suggestions. In order not to bias the ratings of the models in the way of automatic keyword creation, have, this feature has been removed. Instead, there is now a static field that describes what is meant by keywords and how they should be used based on an example.

Numbers to Words During the first tests, errors often occurred when the language model had to deal with numbers. This usually occurred in the sense of a quantity or time. Therefore, a function that converted the numbers into words was added. For example, “I’ll be there at 5 o’clock” was converted to “I’ll be there at five o’clock”. However, this did not improve the situation, but rather caused the language model to also convert large numbers into words. Since this is very unusual in written conversations, this feature was discarded.

Sentence Comparison Discarded At the beginning of the experiment, an initial automatic evaluation was already carried out when the answers, keywords, and ratings were submitted. For example, the participant’s answer and the automatically generated answer were checked for cosine similarity using different tools. Second, it was examined which keyword generator was closest to human keyword generation. More on this in section 7. Performing the evaluations live during the experiment, however, had a negative impact on the runtime of the experiment. Since all the data is available to perform this evaluation at a later time, it was decided against performing it live. With this, the time it takes to get an answer from the system (a generated answer or question) was reduced by about 45 %.

6. Experiment

No Additional Features The focus of the experiment is to determine whether a model that generates complete sentences from keywords can be created. To maintain focus, other features such as: Translation, Speech-To-Text and Text-To-Speech were discarded. All these features increase complexity and introduce unnecessary sources of error that are not in the control of the model. As will be elaborated in section 8.2, these functions are suitable for a future prototype.

Further Minor Adjustments Finally, some minor adjustments are summarized in the following listing:

- Even though there is a note for the keywords that they should be separated with a space, some users do not pay attention to this and separate them with a comma. Therefore, a function was added which converts comma-separated keywords into space-separated ones.
- The font size was increased for different elements, as it often appeared too small. In some places (e.g. text box). However, this was not possible because Gradio does not allow the standard font size to be overwritten.
- Some users have used the dark mode of the system or browser. This caused the contrast between the background and the font to become poor. Because of this, there is a hint at the beginning of the experiment to disable the dark mode for the experiment.
- If a user has questions despite the “Next Steps” and the pictorial description, a FAQ dropdown was added. Here, all frequent questions that came up during the interviews are stored.

6.8. Participants

This section is intended to provide information on how the different experiment participants were collected. In addition to the usual methods such as friends, family and survey circles, crowd workers were also recruited via Amazon Mechanical Turk. Overall, attention was paid to aiming for at least 30–40 % more samples than necessary. This is mainly because many will not complete the experiment, fail the Attention Check, or simply say they will do the experiment but never do.

6.8.1. Family, Friends, and Survey Circles

In general, many participants could be collected through a diverse circle of friends from different demographic and age groups. To participate in the experiment, it is sufficient to simply send the experiment link. Everything else is clearly explained in the experiment, as described in section 6.2. In terms of numbers, 71 results could be acquired from this group, as it will be stated in section 6.9.

6.8.2. Amazon Mechanical Turk

In order to obtain results quickly and reliably, the experiment was also carried out with so-called crowd workers. More precisely, the Amazon Mechanical Turk (MTurk) platform was used for this [10]. MTurk is an online crowdsourcing system developed by Amazon that allows access to a large, on-demand, and cost-effective labour pool to complete tasks that require human intelligence. MTurk enables researchers to engage hundreds or thousands of people to do a wide variety of tasks, such as content moderation, data collection, image annotation, sentiment analysis, and natural language processing. The platform features a web-based interface that allows requesters to post jobs and workers to complete tasks. Workers are paid for the tasks they complete, and requesters are charged a fee for using the service. To create a survey as a requester, a series of steps and requirements must be fulfilled. A step-by-step explanation, my experience of which settings and parameters have worked most effectively and a brief cost overview can be found in A.7.

Once the general conditions have been clarified, the layout of the experiment can be created. MTurk offers different options to create the survey directly on the platform. However, I used the option to provide a link, as shown in Fig. 6.7.

Survey Link Instructions (Click to collapse)

We are investigating the effectiveness of a language model that enables stroke patients to speak again. The model creates complete sentences from a few given keywords (which stroke patients can still say).

In the experiment, there are different everyday situations to which you have to respond. Once as a whole sentence (as a reference for the system) and once in keywords.

Make sure to

- **leave this window open as you complete the survey.** When you are finished, you will return to this page to paste the code into the box.
- Turn off any dark mode in your browser / operating system

The password and username to enter the experiment are:

- Username: user
- Password: user

Survey link: <https://dialokey-experiment.hf.space/>

Provide the survey code here:

e.g. 123456

Submit

Figure 6.7.: Amazon Mechanical Turk layout

After a brief description, the worker clicks on the link, runs the experiment on the web application created with Gradio, receives the code at the end of the experiment and inserts it into the MTurk interface as seen in Fig. 6.7. As described in section 6.5.5, the code indicates whether the experiment was carried out conscientiously or not.

6. Experiment

Once a number of workers have submitted their results, these must be either approved or rejected. All results from workers with a wrong code are rejected, and all others are approved. If the result is rejected, the worker does not receive a reward and a new worker is searched for the survey.

6.9. Results Cleaning

Data cleaning is a crucial step in the data analysis process, as it ensures that the data used for further evaluation is accurate and reliable. This chapter states the process of cleaning the data obtained from the experiment in order to prepare it for further analysis. Specifically, it explains the steps taken to identify and remove erroneous data, as well as methods used to check the accuracy and completeness of the data.

Especially with Amazon Mechanical Turk, there are many participants who are not interested in the outcome of the survey, but only in their reward. Accordingly, some measures have been taken to ensure the quality of the results of the experiment.

One of the first steps is to use only valid results from the experiment and only data from participants whose code has passed the fraud detection test in section 6.5.5. If this code contains indications that the experiment was not carried out conscientiously, I did not use the results for the later evaluation.

The code is only shown at the end of the experiment, i.e. after ten questions and answers. This also means that all participants who stop the experiment before they have answered ten questions are excluded from the results.

Furthermore, it was checked whether participants only clicked on a rating without really carrying out the experiment conscientiously. Specifically, all participants who answered “okay” more than five times were excluded.

A further examination of the results checks whether the keywords entered are longer than the human response entered. If so, that one response was also excluded from the results.

As shown in table 6.2, a total of 184 different participants took part in the experiment. Approximately, 3,099 responses and ratings were collected. Care was taken to keep a balanced number of Amazon Mechanical Turk workers and other participants. Out of 184 participants, 61 were from Amazon Mechanical Turk and 71 were friends, colleagues, fellow students and others. A total of six did not pass the Attention Check and were therefore excluded from the results. A further 43 participants did not complete the experiment. After subtracting all non-usable results, 2,455 answers and evaluations can be used.

Specific to Amazon Mechanical Turk, 38 % of participants did not complete the experiment, but simply entered a random code in the hope of receiving the reward for the experiment. Ultimately, about 30 % of all participants did not complete the experiment for various reasons, as stated in table 6.2.

Received Ratings & Answers	3099
Cleaned Ratings & Answers	2455
Participants	184
Unfinished	43
Mechanical Turk	61
Family, Friends, Colleagues	71
Failed Attention Check	6
Only clicked on okay	3
Mechanical Turk Approval Rate	62 %

Table 6.2.: Statistics of the responses received from the experiment

6.10. Reproducibility

In the following, the reproducibility of the experiment and the cleaning of the results are briefly discussed. The corresponding instructions on how the code works exactly are well commented in the individual files. Therefore, the following descriptions only serve as general hints.

6.10.1. Experiment

The experiment was written in Python and uses the technologies outlined in section 6.4 for the front end and back end. Besides the described Docker container in Huggingface Spaces, the experiment can also be run locally. To achieve this, the repository must be cloned and used as described in appendix A.19.1. Subsequently, all libraries specified in the *requirements.txt* file are needed for the code. Since the code requires some environment variables, these can be added according to the instructions in appendix A.19.1. Afterwards, it is sufficient to run the *app.py* file in a Python 3.9 environment. A link is then provided to the local host, which can be used to access the experiment.

Various settings can be made in the experiment itself, for example:

- Adjustments to which models should be tested in the background
- Password and username
- Scenarios
- Storage location of the logs
- etc.

In the code, each function is commented on what it is supposed to do. Furthermore, it is recommended to reference the Gradio documentation [61] if one has not formerly worked with such a system. There is no need to care about local or relative locations of the logs, as the log files are sent to MongoDB via API and no further local files and installations are needed to run the code.

6. Experiment

6.10.2. Results-Cleaning

To clean the answers and ratings of the users and to store the resulting data in the database, it is sufficient to run the Jupyter notebook *Evaluation/cleaning_data.ipynb*. The notebook can be found in the repository under appendix A.19.2. It is recommended to use a Python 3.10 environment. All necessary dependencies are installed automatically. Only the path for retrieving and saving the MongoDB data must be added as described in A.19.2. The procedure is explained in detail in the code and the current progress and results are printed.

Essentially, the code reads all the data, filters it according to the criteria mentioned in section 6.9 and stores the data again in another database. Once all the data has been cleansed, statistics are also printed regarding the number of answers, cleansed answers, etc. The raw data of all answers and scores can be found in database *logs_04*. After the data has been cleaned as described in section 6.9, it is stored in database *logs_cleaned_3*. All data of the experiment can be read out and exported with normal mongodb operations, as commented in the code at the appropriate place.

7. Evaluation

The focus of the evaluation is to confirm or falsify the research question: “What is the best strategy for designing a system that forms full sentences out of uttered keywords, thus giving a voice to speech impaired people?”.

For this, I have considered numerous parameters in the evaluation, which will be discussed in more detail in the following. The most important consideration hereby is the human evaluation of the different models of the experiment. The aim is to determine a significant difference and thus determine the best method or model for DialoKEY.

In addition to the human evaluations, several other criteria were also considered in the evaluation. On the one hand, additional automatic evaluation methods are applied, which compare the input of the humans with the sentences of the models. On the other hand, other factors such as latency, costs, length of the answers compared to the human answers or the presence of keywords in the generated sentences are additionally taken into account. As a further point of examination, the automatic keyword generation is assessed, and it is measured whether it corresponds to the human keyword generation. In the following sections, the individual evaluation methods are explained in more detail.

7.1. Human Evaluation

The human evaluation is the main focus and the most important key figure that determines the success of the individual models. This is due to the fact that the system is eventually used by speech impaired individuals in conversation with other human beings. The application is accordingly strongly trimmed for the interaction between people.

To begin with, however, it should be noted that a perfect evaluation is almost impossible. This is caused by the fact that the keywords are not always meaningful enough to indicate which sentence should be generated based on them. For a set of keywords, there are many ways to combine them in a sentence, which can all be correct. This significance is illustrated by the following simple example.

Question: For how many people should that reservation be?

Keywords: five two vegetarian

The information could be that a table should be reserved for 5 people, but two of them are vegetarian. Another interpretation could be that the table is for a total of seven people, of whom two are vegetarians. Likewise, five might refer to the time and two is the number of people. This simplified example illustrates that a lot of different possible answers co-exist, which is why a perfect rating seems very unlikely.

7. Evaluation

7.1.1. Rating

Since the human evaluation is the most important in order to answer the research question decisively, these are first examined qualitatively and then tested for their statistical significance. Table 7.1 provides an example of how the generated sentences result in the respective evaluation category and which characteristics can lead to the respective rating. The last answer in the conversation column, indicates the answer created by the system based on the keywords.

Rating	Conversation	Keywords	Human Answer	Explanation
5 Perfect	"Question: Hi! Let's chat about hobbies. What is yours?","Answer: I recently bought a gaming PC, and I'm really excited to try it out!","Question: What types of games do you like to play?","Answer: I like to play only multiplayer games."	like only multi-player	I very much like only multiplayer games	Perfect ratings mostly occur when the created answer (almost) fully resembles the human answer
4 Good	"Question: I'm sorry to hear that. What's been going on?","Answer: I got fired from my job today, and now I have no money to support my pregnant wife."	got board kite gift	I got fired from my job today, my wife is pregnant and I do not have any money.	Sometimes the sentence is only slightly different, some minor details are added or neglected, but the gist of the sentence remains similar.
3 Okay	"Question: Hello at Cutting Edges Hairdresser, what can I do for you?","Answer: Hi there! I was wondering if you offer hair colours services and what colours you have available?"	hair colours	i would like to colour my hair	If too many details are added, the information is wrong, but the meaning is more or less still present in the sentence, this rating is chosen.
2 Bad	"Question: That sounds great! What motivated you to start these hobbies? ","Answer: I love going outside and going to the office job!"	office job love going outside	I work in an office job, so I love going outside from time to time	Bad ratings occur when the information is misinterpreted, too many details are added or neglected.
1 Wrong	"Question: Welcome to the Restaurant Tikka on Wheels. How can I help you?","Answer: Sure thing! What type of curry would you like to order?"	order curry please	I would like to order some curry please	A wrong rating is mostly associated with the model not being able to fully understand the task. These answers are often formulated from the wrong point of view.

Table 7.1.: Explanation of different rating categories with examples

In the following, the corresponding numbers of the ratings may be used to refer to the individual ratings. An average rating close to 5 accordingly indicates an excellent rating.

Before discussing the individual models, it should be noted that the ratings are generally extremely good. As Fig. 7.1 underlines, in almost every model is a gradation from the highest ratings for *Perfect* to the lowest ratings for *Wrong*. A table with the exact distribution of the ratings can be found in appendix A.8.

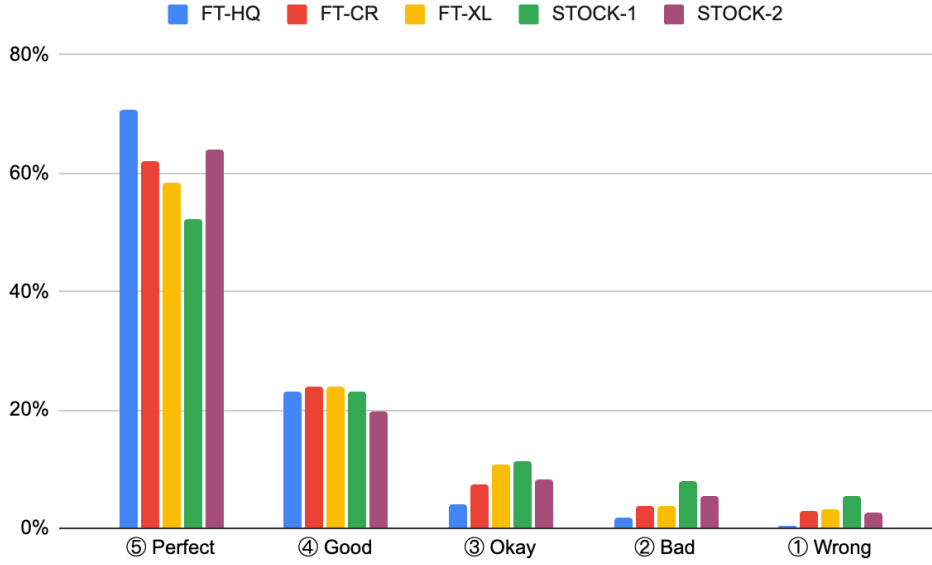


Figure 7.1.: Distribution of human ratings for all models

For ratings 1-3, the FT-HQ model has the lowest number of ratings. For the *Perfect* rating 5, it can be observed that the FT-HQ model performs best, with more than 70 % of all ratings given. This fact is also reflected in the average rating, as illustrated in table 7.2. In this and all following tables, avg. represents the average and SD the standard deviation.

Model	Rating (avg.)	Rating (SD)
FT-HQ	4.6217	0.6870
FT-CR	4.3868	0.9784
FT-XL	4.3017	1.0246
STOCK-1	4.0913	1.1972
STOCK-2	4.3691	1.0219

Table 7.2.: Average human ratings for different models

With an average rating of 4.62 and the lowest standard deviation, it can be said that this model performs best in the human rating. The STOCK-2 as well as the FT-XL and FT-CR models behave relatively similarly in the distribution of ratings.

7. Evaluation

In all three models, there are more ratings in the range 1-3 compared to the FT-HQ model. This is also reflected in the standard deviation, which is on average about 47 % higher than that of the FT-HQ model. The average rating of the three models is also correspondingly lower.

As can be clearly seen, the STOCK-1 model performs the worst. Not only are the ratings very poor, but the standard deviation is also very high. Compared to the other models, this model has approx. 25 % of all ratings in the 1-3 range. This ultimately leads to an average rating of just over 4. An explanation of why the models perform this way is discussed in more detail in section 7.1.3.

7.1.2. Statistical Significance

In order to examine the results both qualitatively and for their statistical significance, an analysis of variance is carried out in this chapter. The ANOVA test (Analysis of Variance) is a statistical test used to compare means between two or more groups. It is used to determine if there is a statistically significant difference between the means of two or more independent groups, or if these differences are due to chance [100].

If the resulting p-value is less than the alpha level, the null hypothesis is rejected and the alternative hypothesis is accepted and vice versa. The null hypothesis states that models do not have a significantly different average. The alternative hypothesis states that there is a significant difference in the mean values. The alpha level for the ANOVA is set to 0.05.

The results of the ANOVA, with the first 230 test results for each model to ensure comparability, demonstrate clearly that the null hypothesis can be refuted. With a p-value of 7.12e-07, this is considerably below the alpha level (see the results in appendix A.9). In other words, the average scores of the individual models are significantly different, and the alternative hypothesis is accepted.

To compare the differences between the individual models in pairs, a Tukey HSD test can be applied [4]. Tukey HSD (Honestly Significance Difference) is a statistical technique used to compare the means of multiple groups. It is used in conjunction with the ANOVA to determine which samples have statistically significant differences between their means. With the Tukey test, all possible pairs of means are compared, and determined which ones are significantly different from each other.

As Fig. 7.2 illustrates, some average values are significantly different from each other, while others are too similar to invalidate the null hypothesis. The last column indicates whether the null hypothesis is rejected or not. Again, the null hypothesis is that the averages are not significantly different.

It is noticeable that the STOCK-1 model performs significantly worse than the other models in almost all cases. Furthermore, just the other way around, the FT-HQ has significantly better scores than almost all other models. The other models (FT-CR, FT-XL and STOCK-2) do not significantly differ from each other, as their means are too similar.

Multiple Comparison of Means - Tukey HSD, FWER=0.05						
group1	group2	meandiff	p-adj	lower	upper	reject
FT - CR	FT - HQ	0.2217	0.1214	-0.0328	0.4763	False
FT - CR	FT - XL	-0.0957	0.843	-0.3502	0.1589	False
FT - CR	STOCK - 1	-0.3174	0.0061	-0.5719	-0.0629	True
FT - CR	STOCK - 2	-0.0348	0.9959	-0.2893	0.2197	False
FT - HQ	FT - XL	-0.3174	0.0061	-0.5719	-0.0629	True
FT - HQ	STOCK - 1	-0.5391	0.0	-0.7937	-0.2846	True
FT - HQ	STOCK - 2	-0.2565	0.0471	-0.511	-0.002	True
FT - XL	STOCK - 1	-0.2217	0.1214	-0.4763	0.0328	False
FT - XL	STOCK - 2	0.0609	0.966	-0.1937	0.3154	False
STOCK - 1	STOCK - 2	0.2826	0.0208	0.0281	0.5371	True

Figure 7.2.: Tukey HSD test with all models

7.1.3. Detailed Rating

At least as meaningful as the overall rating is the detailed evaluation. This means in which sense the model “misinterpreted”, if a rating was not evaluated as perfect.

As described in section 6.1, the following options are available for the experiment participants to choose from.

1. Slightly different
2. Too brief / short
3. Missed details
4. Added details
5. Wrong information
6. Totally different

Once again, it should be noted that these options are ordered by “severity”. While a *Slightly different* answer is only a minor offence, *Totally different* is a completely false statement that not only conveys false information but is also taken out of context. Table 7.3 shows examples and explanations of conversations that have received the respective detailed rating to gain a precise understanding of the individual ratings. Again, the last answer in the conversation column, indicates the answer created by the system based on the keywords.

The examples illustrate that a rating in the direction of *Wrong information* or *Totally different* severely disrupts the flow of conversation.

7. Evaluation

Detailed Rating	Conversation	Keywords	Human Answer	Explanation
Slightly different	"Question: Hello and welcome to the Wheely Great Car Repair Shop. What's wrong with your car?","Answer: I'm not sure what's wrong with my car, but it won't start."	not know not start	I don't know. The car doesn't start any more.	In some instances, very minor details are added or the answer only slightly differs in wording from the human answer.
Too brief / short	"Question: Hello, this is Giovanni's Pizza and Pasta Restaurant, how can we help you?","Answer: Hi, I'd like to reserve a table for two people.","Question: Sure, when would you like to come in?","Answer: Tomorrow at 7:00 PM."	tomorrow 7pm	I would like to come in tomorrow at 7pm.	This rating occurs when the answer is not embellished enough and most of the time shorter than the human reference. The information, however, is the same.
Missed details	"Question: Welcome to Dr. Peppers Medical Clinic. I am Lisa, what can I do for you?","Answer: I have a fever."	fever	I have fever for last few days	Often selected when the human answer contains details, that the created answer does not. However, often these details are not passed in the keywords so that there is no chance for the system to add this information.
Added details	"Question: This is the Mississippi online shop customer service. How can I help you?","Answer: Hi there! I'm interested in buying a new camera from the Mississippi online shop. Can you tell me what types of cameras you have available and any deals or discounts that may be available?"	buy new camera	I would like to buy a new camera	Models sometimes add details such as the discounts in this instance. Furthermore, the answer is excessively embellished.
Wrong information	"Question: Hi! Let's chat about hobbies. What is yours?","Answer: I absolutely love all sports! You name it, I'm probably into it!","Question: Wow, that's great! What kind of sports are you into? ","Answer: I love all kinds of running. I also like to play basketball sometimes."	running	I love running	Not only are details added but also the information is simply wrong.
Totally different	"Question: This is the Mississippi online shop customer service. How can I help you?","Answer: Our return policy is that you can return any item within 30 days of purchase for a full refund or exchange."	what re- turn policy	What is your return policy?	This happens when the model does not understand the task properly. Mostly the model answers from the perspective of the interlocutor instead of the speech impaired person.

Table 7.3.: Explanation of different detailed-rating categories

In contrast to the general average ratings, there are even more distinct differences to be observed between the models. Fig. 7.3, and appendix A.10, illustrate the results of the detailed ratings.

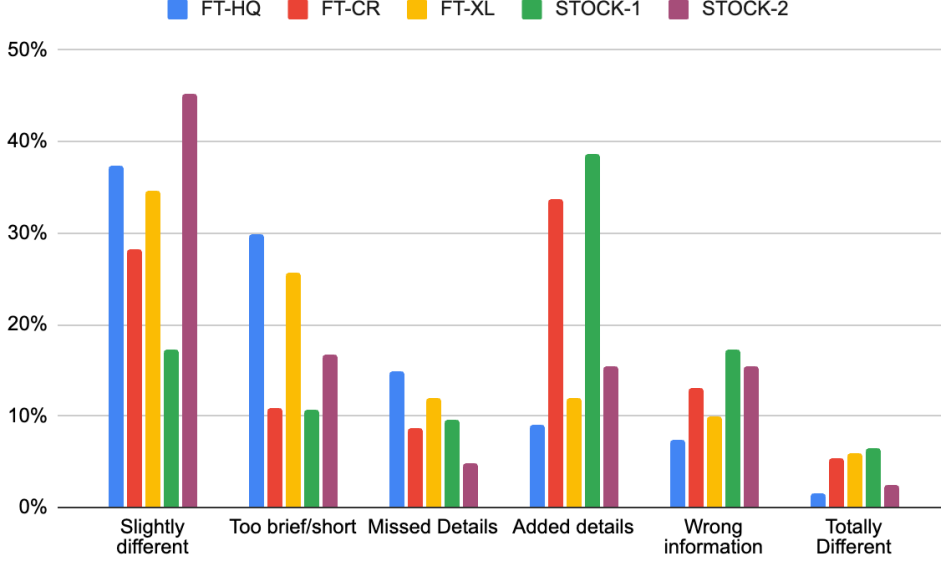


Figure 7.3.: Distribution of detailed ratings for all models

The FT-HQ (in blue) and the FT-XL model (in yellow) peak at *Slightly different*. This is a favourable indication, as the answers are usually only slightly different from expected. The XL model differs from the HQ model to the extent that it has almost 4 times as many *Totally different* and almost twice as many *Wrong information* ratings. This suggests that more data does not necessarily result in better fine-tuning. On the same time, the FT-XL model has more ratings for *Added details*, suggesting that more data does not necessarily lead to less creative models. Since the overall rating of the HQ model is significantly better though, it can be concluded that quality is more important than quantity.

The FT-CR model (in red) stands out among the fine-tuned models in that it received the most ratings for *Added details*, 3 times more than the other fine-tunes. This indicates that the fine-tuning towards a “creative” model has worked in comparison to the others. It is also striking that even if the rating *Added details* was often given, the model still scores very well in the general rating (as the second best model overall). It yields the impression that humans expect an AI to give detailed and beautifully formulated answers.

Furthermore, it is noteworthy that the “creative training” also works in the opposite direction. While the FT-CR model was trained with as few keywords as possible, the average number of keywords in the training data of the FT-HQ model is 3.96 (see section 5.2.3. This is on average more than 1 word longer than the average number of 2.3 keywords in the results of the experiment.

7. Evaluation

The impact of this difference is evident in the *Missed details* rating. The FT-HQ receives the most ratings across all models in this sector. One possible explanation is that the model has always been trained with much information in the keywords, but experiment participants expect the models to add some information themselves.

The two prompt models perform enormously differently from each other. While the very open formulated prompt in the STOCK-1 model (in green) has already attracted attention with its high standard deviation and poor overall rating, this picture is also reflected here. With 38 % from all ratings for this model, *Added details* is the most voted option for. This is the highest value across all different models. Wrong information and *Totally different* are also the most frequently voted in comparison to the others. The very brief task description to the language model is often not enough to explain to the model exactly what it has to do. If the keywords, questions, and answers are not informative enough, the system frequently misinterprets the task and gives entirely wrong answers.

The STOCK-2 model (in purple) is very similar to the Fine-tunes HQ and XL, so the majority of ratings in this case are for *Slightly different* and *Too brief / short*. However, it is noticeable that even though the performance of the model is generally good and the ratings for the detailed rating indicate no major flaws, more *Wrong information* and *Totally different* are given than in comparison to the FT-HQ model. This is most likely because this model is still based on Zero-shot learning. Although it regularly produces good results, there are occasional situations in which it, like STOCK-1, completely misinterprets the task and produces incorrect results. Even though the performance of LLMs regarding Zero-shot learning is already at a great level, the problems occur with “outliers” when the system provides false information.

In summary, it can be stated that the STOCK-2 and FT-HQ models perform best in the detailed rating. All others either receive many ratings around *Added details*, *Totally different* or *Wrong information*.

7.1.4. Adjusted Rating

Since, as mentioned in section 7.1.3, the detailed rating is crucial, this chapter deals with an adjusted rating system. Various behavioural patterns are enormously critical if the system is used to speak for a human being.

As mentioned earlier, it is not necessarily a serious problem if a system’s answer is *Slightly different* from intended. However, it is worse if details are added or the conversation is completely misinterpreted by the system. Although a *Wrong* rating is usually associated with a poor overall rating, the adjusted rating system is designed to show how ratings change when detailed ratings are introduced even more into the overall rating.

The adjustment system will punish or adjust the ratings of the models according to the following principle of table 7.4.

Detailed Rating	Adjustment Factor
Slightly different	1
Too brief/short	0.9
Missed details	0.6
Added details	0.5
Wrong information	0.25
Totally different	0.1

Table 7.4.: Adjustment factor with regard to detailed ratings

One means no adjustment in this case. The lower the number, the greater the adjustment. The ratings received (from 1-5) are multiplied by formula: $1 * \text{Adjustment Factor}$. For example, a rating of *Good* (4) but with *Added details* results in a rating of: $4 * 0.5 = 2$.

If the detailed ratings are considered according to the adjustment system, a slightly different picture emerges as shown in table 7.5.

	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
Rating (avg.)	4.6217	4.3868	4.3017	4.0913	4.3691
Rating (SD)	0.6870	0.9784	1.0246	1.1972	1.0219
Adj. Rating (avg.)	4.4337	3.9926	4.0389	3.5554	4.1530
Adj. Rating (SD)	1.0532	1.5117	1.4188	1.7435	1.4029

Table 7.5.: Adjusted rating in comparison to normal ratings

With regard to the fine-tuned models, it is immediately noticeable that the Creative Model (FT-CR), after its comparatively good rating in section 7.1.1, now loses ground. This is mainly due to the many additional details that the model includes.

The FT-XL model also loses 0.5 rating points, since the answers comparatively often not only deviate marginally from the result, but contain additional information or misinterpret the conversations.

Regarding the HQ model, it can be said that it scores best with only 0.2 lost points. The model often generates answers that are *Slightly different*, misses a few minor details, or creates them too briefly. It is still the highest scoring model with an average rating of 4.43 and by far the lowest standard deviation. The constant practice of never completely misunderstanding a conversation, but also giving less detailed answers, pays off.

The two prompt-based models also lose drastically in rating. The STOCK-1 model loses the most with 1.2 points. This is mainly because conversations are often misinterpreted, and new details are constantly being invented. The STOCK-2 model loses about 0.5 percentage points, similar to the FT-XL model. Even though most of the answers only received a *Slightly different* and *Too brief/short*, there is another factor that comes into play here. Since it happens sporadically that the model misjudges the conversation and gives a entirely wrong answer. This is precisely what is punished with the adjusted rating and is reflected in the evaluation scores.

7. Evaluation

In summary, the consistency of the FT-HQ pays off in the adjusted evaluation. Outliers and frequently added details that are not correct are unacceptable in real human conversations. The adjusted rating gives a good indicator of which models avoid these errors.

7.2. Automatic Evaluation

To ensure a reliable comparability of multiple texts, I carried out both human and automatic evaluations using various tools and methods. In the automatic evaluation, I used different metrics to measure the similarity between the texts, ranging from word-based comparisons to semantic-based. The methods were chosen to ensure a broad range of different comparison functions. The metrics are set to compare two sentences. First, the sentence generated by the system, and the second is the reference sentence entered by the participants.

Automatic metrics of comparability leverages several advantages, including:

- **Speed and efficiency:** Automatic metrics can process large amounts of text data quickly and provide results in real-time, which is crucial for many NLP applications.
- **Objectivity and consistency:** Unlike manual evaluations, which are prone to human biases and errors, automatic metrics provide objective and consistent results.
- **Scalability and reproducibility:** Automatic metrics can be easily applied to large datasets and can be reproduced by other researchers, making it easier to compare different studies and results.

One of the main goals of the automatic evaluations was to establish a correlation with the results of the human evaluations. By finding a correlation between the two metrics, I would be able to determine which automatic methods contain human like accuracy and reliability. With this, I could establish a benchmark for future models.

7.2.1. Different Metrics

For the evaluation, different metrics were used to measure the comparability of the texts based on various key indicators. The core aspects of the individual methods are highlighted briefly in the following.

BLEU (Bilingual Evaluation Understudy) is an evaluation metric primarily used for machine translation tasks. It calculates the precision of n-grams (uni-, big-, tri-, and 4-grams) in the generated text compared to the reference text. The settings are adjusted so that BLEU mainly compares unigrams during the evaluation. Often the sentences or keywords have few words only, so that bi-, tri- and 4-grams are not present at all. The BLEU score ranges from 0 to 1, with 1 indicating that the generated and reference texts are identical. To account for the brevity of the generated text, a brevity penalty is applied, which penalizes shorter generated sentences compared to the reference sentence [103].

The ROUGE metric (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics and a software package designed for evaluating automatic summarization and machine translation. It compares an automatically produced summary or translation against human-produced reference summaries or translations. ROUGE-1 F1-score is a specific metric within the ROUGE family. It measures the overlap of unigrams (individual words) between the system-generated text and the human-produced reference. The ROUGE-1 F1-score is a measure of the trade-off between precision and recall, providing a single score that balances the two [85].

METEOR (Metric for Evaluation of Translation with Explicit ORdering) is another evaluation metric used for machine translation. It calculates the harmonic mean of precision and recall, while also considering word order and synonymy. This makes it more robust to issues of lexical choice compared to BLEU and ROUGE. METEOR also includes a penalty for sentence length differences [14].

spaCy calculates the similarity using cosine similarity between word vectors or word embeddings, which are a multidimensional representation of a word. The similarity between two sentences or documents is computed as the average of all the word vectors that constitute them. A larger pipeline package was used, to avoid spaCy yielding to a high similarity score despite sentences having opposite meanings. Otherwise, spaCy's default similarity implementation is based on a general-purpose definition of similarity and can sometimes fail semantically [121].

FuzzyWuzzy is a Python library that calculates similarity scores between strings using the Levenshtein distance [81], which is a measure of the number of required single-character edits (e.g. insertions, deletions, or substitutions) to transform one string into another. The library provides several functions to match strings based on their similarity, returning a score from 0 to 100, with 100 representing a perfect match [116].

The JARO similarity measure compares two strings of text and calculates the similarity between them by determining the number of matching characters between them, the number of transpositions between the two strings, and the number of characters that need to be inserted or deleted to make them equal. It is commonly used to determine the similarity of two different versions of the same text [113].

Similarity comparison with BERT involves utilizing a pre-trained model to compare the similarity between two pieces of text. This can be done by using a BERT-based model to encode both pieces of text and then comparing the resulting vectors. For similarity comparison, BERT generates contextualized embeddings that represent the input sentences. These embeddings capture the semantic information of the sentences. The model then calculates the similarity between the two documents by computing the cosine similarity between their respective embeddings [117].

7. Evaluation

7.2.2. Results of Automatic Evaluation

Table 7.6 illustrates the results of the explained metrics (see appendix A.11 for the standard deviations). The results are highly diverse, as it is evident in the table. For all ratings, 1 indicates high concordance of the generated and the reference sentence and 0 indicates low concordance. The numbers in bold represent the highest rating of a metric across all models, the numbers in italics are the second-best value.

Model	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
BERT	0.7278	0.6722	0.6917	0.6327	<i>0.7268</i>
FUZZ	0.6255	0.4965	0.6010	0.4688	<i>0.6219</i>
JARO	<i>0.6848</i>	0.6412	0.6769	0.6308	0.6849
BLEU	<i>0.3187</i>	0.2221	0.3084	0.1880	0.3371
METEOR	<i>0.3254</i>	0.2539	0.3113	0.2199	0.3428
ROUGE	0.4621	0.3868	<i>0.4655</i>	0.3434	0.4745
SPACY	<i>0.7589</i>	0.7835	0.7794	0.7666	0.7845
Human (Adj.)	4.4337	3.9926	4.0389	3.5554	<i>4.1530</i>
Human	4.6217	<i>4.3868</i>	4.3017	4.0913	4.3691

Table 7.6.: Results of automatic evaluation metrics

According to the automatic evaluation methods, both the FT-HQ and STOCK-2 models perform best. Both have the highest score on 3-4 different metrics each or are in second place else.

It is noteworthy that the differences between the best ratings are often only marginal. However, the differences between rather poorly performing models compared to a well rated model are very significant. The ratings of the automatic evaluations are therefore trend-setting similar to the human ratings.

To examine this trend, and pursue the goal of finding a automatic similarity score close to the human ratings, a correlation matrix was created.

Fig. 7.4 examines whether a correlation between the automatic and human metrics can be identified. A value closer to 1 indicates a strong positive correlation, while a value close to 0 indicates no correlation. For comparison, the participants' rating and the adjusted evaluation are also included in this chart. It can be seen here that these two indicators, unsurprisingly, show a very high correlation.

Between an automatic evaluation and a human evaluation, all correlations are negligible, with maximum values of 0.24.

None of the automatic metrics rate the sentences slightly similarly to the human ratings. Remarkably, the correlation between the automatic metrics and the adjusted human rating is always a little higher. The reason for this is that most automatic metrics award a lower score when the sentences have more differences in terms of their length. The adjusted rating also assigns lower ratings when, for example, details are added, forgotten or completely misinterpreted. This in turn leads to a difference in length in the sentences.

7.2. Automatic Evaluation

Furthermore, there are high correlations between the automatic metrics FUZZ, JARO, BLEU, METEOR, and ROUGE. This is because all these metrics calculate their score based on the matching strings and characters present in the sentences.

In addition, to test how the different metrics behave with “only perfect ratings” from humans, these were filtered across all models and scored by the automatic metrics. In appendix A.16, the table compares this approach. Again, it can be observed that the automatic metrics produce a score that is thoroughly different from perfect.

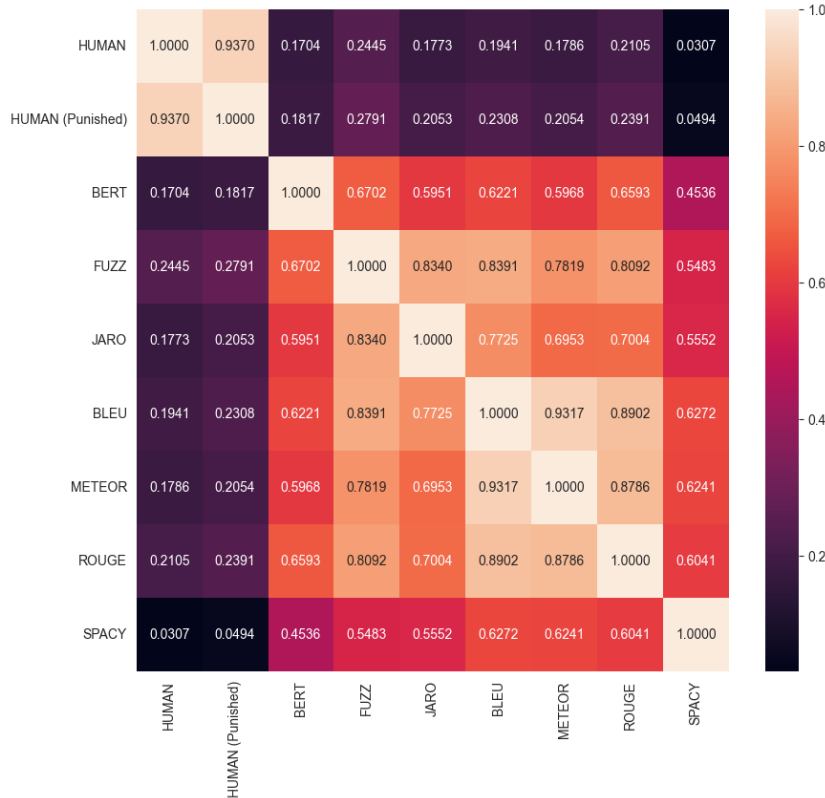


Figure 7.4.: Correlation matrix of automatic evaluations and human ratings

To conclude, none of the automatic comparison methods, regardless of the underlying technology, resembles human judgement.

The main reasons for this may be as follows:

- Human reference sentences are not always formulated to 100 % accuracy, so perfect human ratings occur even when the human reference sentence is a little off. Automatic evaluations tend to rate such comparisons worse.
- Humans are not as strict in evaluating the addition of details as automatic metrics are. As noted earlier, the embellishing of sentences is not as harshly punished by humans.

7. Evaluation

- Complete misinterpretations of sentences are not recognized by automatic metrics. For example, when comparing a human sentence *I would like to order pizza* with a generated *You can order any pizza*, an automatic evaluation may still award a high similarity score on a word basis.

Since none of the automated metrics resemble human evaluation, this again emphasizes the importance of the experiment in evaluating the system by humans.

7.3. Further Evaluation Metrics

In addition to the human and automatic ratings, there are several other criteria that can be examined. Some of these attempt to explain the findings of the detailed ratings. Other subsequent metrics seek to draw conclusions between influential factors in sentences, keywords, or scenarios on the rating.

7.3.1. Length of the Sentences

Many models tend to be too creative and produce longer sentences than necessary. A potential metric to measure that is the length of the sentences.

To ensure comparability, three different lengths are considered:

- The length of the human reference sentence
- The length of the generated sentence
- The length of the keywords

By length, it is referred to the number of words in the sentence. As shown in table 7.7, the average length of the entered keywords is nearly consistent across all answers and models at around 2.5 words. The standard deviation is also almost constant across all models at approximately 1.2–1.9. Considering that patients with aphasia speak in less than 4 keywords (see chapter 2), this key figure is an important consideration.

	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
Length Keyword (avg.)	2.5039	2.5091	2.5038	2.6340	2.5331
Length Generated (avg.)	7.0469	14.36	8.0568	15.5019	8.2568
Length Human (avg.)	6.6445	6.8982	6.7955	7.2415	6.8833
Length Keyword (SD)	1.6966	1.1569	1.2209	1.8905	1.2311
Length Generated (SD)	3.2050	7.0848	4.0903	7.6994	3.3395
Length Human (SD)	3.4269	3.4962	3.5147	4.3487	3.5699

Table 7.7.: Evaluation of sentence and keyword length [in words]

Almost all answers during the experiment were formed with less than 4 keywords. In addition, the length of the human references averaged 6.6 words, with a standard deviation of about 3.4 words. Again, hardly any differences were found between the individual models with regard to the length of the human responses.

However, drastic differences were found in the average number of words in the sentence creation of the individual models. The models considered creative (see section 7.1.3) STOCK-1 and FT-CR have an average number of words more than twice as long as the human-generated reference. The standard deviation is also more than twice as high at over 7 words.

The STOCK-2 and FT-XL models, with an average of 8 words, are still above the human reference, although not as drastically. The standard deviation is also comparatively low here at 3–4 words.

Unsurprisingly, the FT-HQ model comes closest to the human reference. The slight surprise is due to the fact that the model produces the most scores for answers that are too short, and therefore the smallest value was to be expected here. With only 0.4 words more than the human reference and a lower standard deviation, the model is closest to the human-generated sentences. Nevertheless, it is surprising that the ratings *Too brief / short* appeared so often. Comparing a few example ratings, it can be stated that this rating was also given even though the generated response was longer than the human input.

7.3.2. Keywords Present in Sentences

A potential indicator of how good a result is, is derived from whether the keywords are present in the generated sentences.

To evaluate this, however, it is not sufficient to check whether the word of the keywords is present in the generated sentences. This approach would often not be correct, as occasionally the words do not appear in the same form as they do in the keywords. For instance, in the following example, the keyword would not appear the same way in the sentence, but the same meaning is still present.

Keywords: book table

Generated sentence: I would like to reserve a table

Even though book is not mentioned, reserve is a synonym in this case. It is necessary to verify whether the topic of the word occurs in the sentence generation. To realize this, I used a Zero-shot classification pipeline of Huggingface [147]. If a sentence contains all topics from the keywords, a rating of 1 is assigned. The fewer keywords it contains, the closer to zero the rating is. As a reference, I have also subjected the human sentence to the above procedure for each generated sentence. For each model, the score of the human sentences serves as a reference value, as illustrated in table 7.8.

7. Evaluation

Model	Generated (avg.)	Generated (SD)	Human (avg.)	Ref.	Human (SD)	Ref.
FT-HQ	0.8807	0.1735	0.8815		0.1805	
FT-CR	0.8886	0.1801	0.8882		0.1876	
FT-XL	0.8747	0.1933	0.8746		0.2015	
STOCK-1	0.8690	0.1786	0.8698		0.1776	
STOCK-2	0.8874	0.1750	0.8895		0.1738	

Table 7.8.: Comparison of models if keywords are present in created sentences

The similarities between the human scores and the scores of the generated sentences are very high, if not on par. As table 7.8 highlights, the results are nearly identical to the human references except for 1-2 %. The models known to be creative, even achieve a higher score than the human reference. Overall, even the models with the most ratings for *Missed details*, the models omit relatively few keywords.

7.3.3. Number of Keywords and Rating

There could be a connection that depending on how many keywords are entered, the system can form sentences with varying ratings. Theoretically, the more keywords entered for a sentence, the more information the system possesses. This means that the model must add less information and sentence components on its own.

For this purpose, the ratings were examined in this chapter as a dependency of the number of keywords given by the participants. Table 7.9 illustrates the average ratings of the individual models when 1,2,3,4 or ≥ 5 keywords were passed to the system for answer generation. Although there are slight differences between the ratings for different numbers of keywords, there is no discernible trend.

	1 keyword	2 keywords	3 keywords	4 keywords	≥ 5 keywords
FT-HQ	4.5319	4.7586	4.4912	4.6207	4.6667
FT-CR	4.4186	4.4433	4.3273	4.44	4.1739
FT-XL	4.2045	4.4565	4.1875	4.6364	3.80
STOCK-1	4.2245	4.1932	3.7667	4.1667	4.20
STOCK-2	4.50	4.5059	4.3651	3.8077	4.1538

Table 7.9.: Ratings in comparison to the number of keywords

Across all models, there is no significant change in models for a different number of keywords. The average rating differs only marginally depending on how many keywords are chosen.

However, a few minor differences can be observed. In fact, the ratings for 1–2 keywords are relatively constant across all models.

Surprisingly, the ratings of the stock models deviate strongly downwards for a higher number keywords. STOCK-1 performs very poorly for 3 keywords, while STOCK-2 has a poor rating for more than 4 keywords.

The FT-XL model also has a worse rating for 5 or more keywords. One possible reason for lower ratings with more keywords is that the models tend to simply reproduce what is said in the keywords without embellishing the answer. The created sentences become shorter, and the experiment participants expect the generated sentences to be richer than just what is contained in the keywords.

In addition, it should be noted that the standard deviation is usually greater with more keywords than with a few keywords (see appendix A.13.1).

As elaborated in section 7.3.1, the average number of keywords across all models was about 2.5. This means that most of the ratings for the models are between 2 and 3 keywords (see appendix A.13.2). Thus, the more keywords, the fewer the ratings and the greater the influence of outliers.

In conclusion, however, it can be said that no correlation can be identified and that the models perform consistently, regardless of the number of keywords. The best performers in this metric are the FT-HQ and FT-CR models, both of which show consistently good results with any number of keywords.

7.3.4. History Length and Rating

The longer the course of the conversation that is handed to the system, the more precise the answers could be. The possibilities mentioned in the opening chapter 7.1, in which directions a conversation can develop, diminish as the conversation progresses. This could mean that the model has fewer outliers and the ratings improve accordingly at the end of the conversation.

To this end, the ratings were examined in relation to their position in the conversation. Since a new scenario comes up every three questions (see section 6.6), there are only three different positions at which an answer has to be formulated by the model: position 2, 4 and 6. Before position 2, there is already one question, before position 4, there are two questions and one answer and before position 6, there are already three questions and two answers.

Table 7.10 shows the average ratings of the individual models at the corresponding positions (see appendix A.14 for the standard deviations).

Against the hypothesis, there is no trend apparent here, on the contrary. Some models perform better at position 1 or 2 than at position 3. In conclusion, there is no significant correlation between the length of the conversation and the performance of the individual models.

7. Evaluation

Model	Position 2	Position 4	Position 6
FT-HQ	4.5070	4.7083	4.6437
FT-CR	4.3371	4.4588	4.3529
FT-XL	4.4881	4.1358	4.2667
STOCK-1	4.0645	4.0811	4.1351
STOCK-2	4.2769	4.4235	4.3855

Table 7.10.: Ratings in comparison to the length of the history

7.3.5. Rating Depending on the Scenario

As described in section 6.3, the scenarios are classified into defined and open. Open is intended to represent an open outcome of the conversation, and closed defines an already predictable outcome of the conversation. In addition, the defined scenarios have a clear goal, such as “reserve a table”.

The question arises as to whether there is a correlation between the performance of the model and the type of scenario.

Model	Open Scenarios	Defined Scenarios
FT-HQ	4.5652	4.6783
FT-CR	4.4123	4.3643
FT-XL	4.3197	4.2833
STOCK-1	4.0463	4.1278
STOCK-2	4.3017	4.4359

Table 7.11.: Ratings in comparison to the scenario type

Table 7.11 examines the ratings of the individual models depending on the type of scenario. It is clearly evident that there are no significant differences between the individual scenarios. In some cases, the defined scenarios perform marginally better, in other cases the open scenarios.

The standard deviation (appendix A.15.1) is also identical across all ratings and categories. In order to completely reject a correlation between the type of scenario and the rating, the ratings were checked depending on the exact scenario.

The exact scenarios refer to the eight different use cases mentioned in section 6.3.

Again, the tables presented in appendix A.15.2 underline that no scenario leads to a noticeably better or worse rating than others.

The standard deviation is slightly inconsistent for some scenarios and models. However, this may also be due to the fact that with approx. 30–40 ratings per scenario, outliers carry a lot of weight.

Finally, it can be summarized that no significantly different ratings can be determined based on the different scenarios or types of scenarios.

7.4. Cost Factor

When using a model in practice, besides performance, the cost factor of the different models is also crucial. Unless a model can be trained and run locally, costs are incurred for all major large language models (see section 5.1.4).

As mentioned in section 5.3, the cost structure for training in fine-tuning and using the models is different. The training in fine-tuning costs less per token than the usage of the model. The costs for the Davinci model from OpenAI break down as follows, whereby one token corresponds to approximately 0.7 words.

- Fine-tuning: \$ 0.0300 per 1K tokens
- Usage: \$ 0.1200 per 1K Token

The corresponding cost of training is four times less than the cost of using a model. The costs of using a model are the same, regardless of whether the use is the “stock” model (not fine-tuned) or a fine-tuned model.

The advantage of fine-tuned models, as described in section 5.1.4.1 is, that no description of what the model is supposed to do has to be provided for each request to the model. This is because it has been trained on a specific use case in the fine-tuning process and has already seen numerous examples of the desired task.

However, for prompt-based requests (zero-, one-, few-shot), a description and optionally examples of the expected result must be provided *with each request*. This means that prompt-based models are pricier to use, since they consume more tokens.

On the other hand, fine-tuning incurs additional costs for training, which is not required for prompt-based models. However, these training costs are cheaper than the use of the model, which in turn saves costs due to a lower number of tokens used in the requests later on.

If the break-even is calculated based on an example, it becomes clear that fine-tunes are cost-effective in practice, provided they are used frequently.

The costs can be represented with linear equations. The assumptions are, that a fine-tuning model costs about \$ 5 (see section 5.3). Furthermore, the average request to a fine-tuned model contains about 36 words.

This corresponds to costs of $\$0.00012 * (36/0.7) = \0.00617 per request.

For prompt-based models, an average request has about 60 words, which corresponds to $\$0.00012 * (60/0.7) = \0.013 per request. This results in the two equations:

$$f(x) = 5 + 0.00617 * x \text{ for fine-tuned models.}$$

$$g(x) = 0.0103 * x \text{ for prompt-based models.}$$

The intersection of the two straight lines is at $x = 1,210.65$. This means that from 1,211 requests onwards, the fine-tuned model is cheaper to use than the prompt-based one.

7. Evaluation

Even if the costs play a subordinate role in applications with fewer requests, the following can be concluded. From a purely cost perspective, it makes sense to use prompt-based models for prototypes and smaller examples. However, if a system is used in practice or is used by several users, it makes sense, purely from a cost perspective, to resort to fine-tunes.

7.5. Latency

An important consideration when selecting the best model is the speed of the models. Latency in this context refers to the time it takes from a request to the model with keyword and history to the model's response with a generated sentence. The advantages of fine-tune models, as listed in section 5.1.4.1, include a fast response time. However, the evaluation of the models in terms of latency shows a different picture, as table 7.12 illustrates.

Model	Runtime (avg.)	Runtime (SD)
FT-HQ	1.0101	0.5439
FT-CR	1.1816	0.6709
FT-XL	0.9568	0.5193
STOCK-1	1.2220	0.6747
STOCK-2	0.9204	0.5231

Table 7.12.: Latency of the models [in seconds]

The fastest model as well as the slowest is in each case one of the two STOCK models. The STOCK-2 model is about 25 % faster than the STOCK-1 model. The three fine-tune models are in between this range. The FT-XL is almost as fast as the STOCK-2 model. Compared to the FT-HQ model, the STOCK-2 model is about 9 % faster. The slowest fine-tune model is the creative model FT-CR. Both models, which produce long sets, are by far the slowest, while the other models are about equal. To summarize, the speed of the fastest three models is comparably fast, so that this does not have a major effect on the overall evaluation.

7.6. Which Dataset Performed Best?

Based on the wide variety of evaluations, the consistency of the FT-HQ model clearly emerges. The model always comes out on top in human evaluations, detailed evaluations and the adjusted rating. In addition to having the most ratings for *Perfect*, it has the fewest outliers and misinterpretations. In numerous instances, the FT-HQ model also received the best rating in the automatic evaluations. Other evaluation criteria, such as costs, latency and other correlations, show only slight differences between the best performing models, but the FT-HQ is always in the top positions too.

However, it is difficult to generalize this statement for the best model. Different applications require different models. While some patients may only be able to answer in very few keywords, but still want to utter meaningful sentences, a creative model may be the better choice. Nonetheless, the findings of the evaluation clearly show that an appropriate model can be created for each use case. This can be done either via prompt-based models or via fine-tuned. The former benefit from an enormously fast implementation, no training data and no initial costs for fine-tuning. Provided that the instructions for the model are clear, explicit, detailed and defined, excellent results can be achieved. As the dramatic difference between STOCK-1 and STOCK-2 demonstrates, a precise task description to the language model is the key to success in prompt-based models. Nevertheless, it is difficult to completely avoid outliers from the prompt-based models in the form of misinterpretations resulting in very wrong answers.

Using fine-tuning, models can be created in a wide variety of directions by appropriately creating the training data and selecting the parameters for fine-tuning. The different creation of training data between the FT-HQ and the FT-CR model and the resulting results underlines this fact.

In conclusion, the evaluation can be summarized with the following statement: Consistency on the cost of creativity. If a consistent model is created, the ability to create creative answers is missing. Conversely, a creative model occasionally falls into the trap of giving wrong answers and tends to be inconsistent.

7.7. Evaluation of Keyword Extraction Tools

In addition to evaluating the models for creating sentences from keywords, this section reviews the keyword extraction tools used to create the dataset (see section 5.2.2.2). The reason for this is to evaluate how well each method matches the human ability to create keywords from a sentence. The aim is to evaluate and discover which method is best at creating keywords for future fine-tune dataset creation.

In order to analyse how well the keyword extractors perform in comparison to the human-generated keywords, different metrics were used. On the one hand, the Sentence Similarity methods from section 7.2 are used. These compare how similar the generated keywords are to the human keywords. On the other hand, similar to section 7.3.2, it is examined whether a certain topic of the human-generated keywords is present in the generated keywords of the methods and vice versa. In addition, with a simple method, it is proved whether the keyword itself (as a string) is present in the generated keywords of the human and the extractor.

Table 7.13 illustrates these evaluation results, where a rating close to 1 resembles full compliance with the human keywords (refer to appendix A.17 for the standard deviation of the means). It should be noted that the tool RAKE (see section 5.2.2.2) could not be used for the comparison. Due to a bug in the library, the strings of the answers could not be decoded and therefore no keywords could be created for comparison.

7. Evaluation

Comparison method	KeyBERT	YAKE	YAKE (Detailed)	Human
Generated KWs in human KWs (String)	0.6541	0.6860	0.5896	1
Human KWs in generated KWs (String)	0.7355	0.6547	0.6175	1
Generated KWs in human KWs (Topic)	0.8580	0.7918	0.7514	0.9923
Human KWs in generated KWs (Topic)	0.8583	0.8012	0.8083	0.9923
BERT	0.7772	0.7495	0.7227	1
FUZZ	0.7228	0.5574	0.6297	1
JARO	0.7404	0.6243	0.6713	1
BLEU	0.5673	0.5475	0.4593	1
METEOR	0.5547	0.4049	0.4139	0.8735
ROUGE	0.6647	0.6420	0.5648	1
SPACY	0.7462	0.7092	0.6255	1
Average	0.7163	0.6517	0.6231	0.9871

Table 7.13.: Means of the evaluation of keyword extraction tools (KWs = keywords)

The last row of table 7.13 represents the average rating across all evaluation methods. The percentage refers to the similarity of the generated keywords to the human keywords. In this average rating, YAKE and YAKE (Detailed) are in second and third place with 65 % and 62 % respectively. Even though the two tools often extract differently, their scores are very similar. KeyBERT performs best with a score of 72%. This makes it the closest to human keyword creation. For future use, it is therefore advisable to rely on KeyBERT.

7.8. Reproducibility

In order to reproduce the code of the evaluation itself, the following section provides an overview of the code used. For more detailed information on what adjustments can be made and how the code works, the well-commented code itself can be consulted.

To perform the evaluation oneself, the code *Evaluation/post_processing.ipynb* in the repository explained under appendix A.19.2 must be used. The requirements are installed automatically, only the file with the corresponding paths to access the data on MongoDB is needed as described under appendix A.19.2.

The programme then proceeds as follows:

1. First, the necessary data is read from the cleansed database *logs_cleaned_3* of MongoDB with some further minor cleaning.
2. Then, all evaluation metrics such as automatic evaluations, lengths of various strings, keyword generators, etc., are applied to each document and stored directly in the cleansed database.
3. Thirdly, this data is read to create all metrics mentioned in section 7.
4. Finally, covariance matrices are created to correlate the individual studies and all further evaluations are printed out in clear tables.

The user is informed about all progress and results via prints.

8. Conclusion and Discussion

This final chapter attempts to draw together the major findings of this research and to discuss the implications of the conclusions. The aim is to reflect on the knowledge gained through the research process and to suggest areas of further study as well as considered limitations of this work.

8.1. Results

What is the best strategy for designing a system that forms full sentences out of uttered keywords, thus giving a voice to speech impaired people?

The preceding chapters serve to answer the research question, which accompanies this thesis. Towards achieving this goal, chapter 2 examines the applications for such a system. As a result, a group of people with the limited ability to speak are identified. In addition to Broca's aphasia, this chapter determines more people with speech restrictions, caused by both medical and nonmedical conditions. The affected people can only speak in single words instead of complete sentences.

The rapid development in the field of Natural Language Processing, more precisely, Transformers, makes it now more possible than ever to enable these people to speak normally again. The fundamental understanding of these newly developed technologies, as well as those employed in the subsequent sections as the basis for the implementation, is laid out in chapter 3. After further examining the current developments in the field of chatbots (chapter 4), no system can be found that meets the requirements to allow the aforementioned groups of people to have a flawless conversation.

To be more specific, there is no a system that can construct meaningful sentences in conversations out of the few words that persons with speech impairments can still utter. To fill this gap, several methods to implement such a system have been compared, with the most promising ones being tested on their capabilities (section 5.1). As a consequence, large language models are used in two distinct manners to implement versions DialoKEY. One way to utilise the model is based on various approaches of Zero-shot learning, and the other is to fine-tune it to the specific application on hand. To achieve the latter, multiple datasets are generated to fine-tune the largest available language model to date, GPT-3. The datasets consist of up to nearly 5,000 data points and have been designed in multiple iterations, each with different properties. Among them, as described in detail in section 5.2.3, are models that are designed to be creative and models that assess whether the quantity or quality of the datasets is pivotal.

8. Conclusion and Discussion

Moreover, an experiment was designed and conducted to test the capabilities of the implemented systems against human evaluations. A total of five different versions of DialoKEY were tested with respect to their performance and characteristics in the corresponding experiment. In the custom designed experiment, more than 2,000 data points of human evaluations were collected for evaluation. These data points were additionally subjected to other automatic, qualitative, and quantitative criteria, such as e.g. the length of sentences or the relevance of misinterpretation, in addition to the human judgements. The results clearly prove that some models are almost on a par with human performance and only rarely lead to misinterpretations. This means that meaningful sentences can be created from keywords in a conversation.

To answer the research question posed in chapter 1, the best method is, to fine-tune a large language model with a tailored and high-quality dataset as elaborated in detail in section 7.6. The consistency of the fine-tuned model is evidently demonstrated by the broad range of evaluations in. More than 70 % of the outcomes generated by this model were rated *Perfect*, and it had the fewest outliers overall. The score is significantly higher than that of the other models, averaging at 4.62 out of 5. On top of that, this is not the only insight, as the following discussion demonstrates.

Ultimately, with DialoKEY, speech-impaired people can be empowered a little further to satisfy the human need for communication, affection and social belonging.

8.2. Future Work and Limitations

During the elaboration process, numerous possible extensions have evolved for subsequent work based on this thesis, which will be briefly outlined alongside the limitations of the current approach in the following.

Different languages DialoKEY is designed for the English language. In order to ensure holistic use with any language, future work can be based on developing the system robustly for other languages.

Several approaches are conceivable: On the one hand, due to their capacity for transfer learning, large language models by default allow the usage of additional languages. This would not require any further adaptation. However, it is necessary to prove whether the capabilities of the language model are at a similarly good level in other languages for the use case on hand.

Another strategy is, to provide the system a variety of translators both before, (from any language to English) and after (from English to the input language) sentence construction. Here, it's crucial to assess whether the latency grows excessively and whether a fluid translation can be ensured.

In general, both approaches offer a adequate solution that can be implemented quickly. It is necessary to evaluate which of the approaches delivers better results in practice.

Speech Input and Output The present work focuses primarily on creating a system that can form complete sentences from keywords in the context of a conversation. However, it is vital to implement elements like STT (Speech-To-Text) and TTS (Text-To-Speech) in order to use such a system in real-life situations.

Similar to translators, there are already some interfaces for this purpose, ready to use. With this extension, it is particularly important that the STT function works robustly and reliably recognizes the spoken keywords. Additionally, it is necessary to distinguish between terms that should be filtered (Hmm, Uhm, etc.) and words that are relevant for the input.

This is especially important as people with speech impairment usually use several such filling words, that are not necessary for the comprehension of the spoken language.

App Development It is worth noting that these two extensions (language and translation) can also be implemented natively with Huggingface and Gradio. Furthermore, the Gradio web application can already be used on mobile devices without any issues.

For an actual MVP (Minimum viable product), however, it would be advantageous to develop a native app that is integrated into the phone app on the mobile phone.

When calling another person, this app would first record the spoken keywords of the speech impaired person, process them into a sentence and then forward it to the other person using TTS. At best, the other person would not even notice that the person is not speaking, but rather a computer giving the answers.

Sentiment and Keywords The words used when speaking have a significant impact on what the speaker desires to convey and express. However, the sentiment, or current state of mood, also has a significant impact on how someone speaks. An extension of DialoKEY is therefore not only to generate the answers based on the keywords, but also to let the sentiment flow into the creation.

A similar implementation to the keyword-based sentence generation is feasible. During the creation of a dataset, a sentiment of the sentence could be analysed in addition to the keyword extraction. With these two parameters, the model will learn how to incorporate sentiments into the sentence creation.

However, other questions arise, such as what happens if the sentiment is unfavourable, but the response is affirmative. For this, appropriate regulations for the system's must be developed.

Model for Sentence Similarity As observed in section 7, there is no correlation between the automated evaluation techniques and human evaluations. The training data produced in the experiment can be used to fill this gap. To drastically reduce the development cycle of future models and save costs, a model for evaluation can be developed based on the training data produced in the experiment.

8. Conclusion and Discussion

Since the human ratings, as well as the generated sentences, human reference sentences and keywords are available in the data, an LLM can be taught how the human ratings are made. As input, the model receives two sentences and the conversation history of the conversation (and additionally the keywords).

The completion to train the system on, is the human rating, including detailed information if the rating is not perfect.

If a significant correlation between this new model and the human evaluation can be established, this evaluation model can be used to evaluate other sentence generating models and in general might outperform existing sentence similarity methods.

8.3. Discussion

The experiment was deliberately designed to compare the diversely created models. Hence, the evaluation of the results provides an answer to the research question. As the analysis shows, the fine-tuned model FT-HQ achieves the best rating of all five implementation methods.

Despite the remarkable high performance of all models and the best model's nearly perfect production of sentences, it is challenging to employ one model for universal purposes. Since people's pronunciation varies in terms of length, word choice, creativity, etc., distinct models might be developed based on these differences.

To combat this, one other important observation emerged from the evaluation. The drastically different detailed evaluations of the systems show that in both fine-tuning and zero-shot learning, the models can be steered in the direction of various characteristics. In prompt-based models, this is expressed by the task description that a model receives. Even though outliers and misinterpretations cannot be completely avoided, a clear statement can be made: The more detailed and specific the task description to the model, the better the results in every respect.

With regard to fine-tuned models, it can be stated that the training data fully reflects the behaviour of the model. With the right settings, a model can be trimmed in different directions, for example, to respond more deliberately or more creatively. As implemented in the code for producing datasets, they can be steered in a single direction using already available methods.

Although the fine-tuned model achieves significantly better scores, the use of prompt-based models (zero-shot learning) should not be neglected. The benefits of quickly employing a system without fine-tuning and dataset generation outweighs the benefits of fine-tuned models' superior performance for applications such as proof of concepts, for instance.

Ultimately, one of the most important insights is: Consistency on the cost of creativity and vice versa. Models with highly creative answers were penalised by the evaluations as being less consistent. In other words, the answers may contain additional made up information or miss the intended essence of a sentence.

As the extensive search for a perfect way of implementation in section 5.1 has shown, Transformer-based large language models are superior to other investigated alternatives (such as e.g. other types of neural networks). Brown et al. [21] and section 5.1.4 demonstrate that the size of the language model correlates strongly and positively with its performance. The results of the proposed solution are therefore directly correlated with the capability of the underlying language model. The rapid development and new releases of larger models inevitably leads to improvements of DialoKEY. It is therefore essential to constantly adapt the underlying model. The algorithms created in implementation 5.2, enable to create new training data and fine-tune newly released language models based on provided or new conversation data.

Furthermore, the designed experiment in section 6 facilitates the evaluation of new models with marginal modifications in order to strive for continuous optimization for the system.

Finally, DialoKEY can be utilized to provide a better life for speech impaired people and serves as a foundation for further research. Therefore, this system opens up a new avenue for this population to communicate effectively.

Let's make their voices heard.

Bibliography

- [1] *6 Types of Aphasia - Regional Neurological Associates*. URL: <https://www.regionalneurological.com/types-of-aphasia/>. Date accessed: 23/11/2022.
- [2] *A ten-minute introduction to sequence-to-sequence learning in Keras*. URL: <https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html>. Date accessed: 10/04/2023.
- [3] Martín Abadi et al. *TensorFlow: A System for Large-Scale Machine Learning*. URL: <https://tensorflow.org>.
- [4] Hervé Abdi and Lynne J Williams. *Tukey's Honestly Significant Difference (HSD) Test*.
- [5] Abubakar Abid et al. *Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild*. URL: www.gradio.app.
- [6] *About | Happy Transformer*. URL: <https://happytransformer.com/>. Date accessed: 10/04/2023.
- [7] Eleni Adamopoulou and Lefteris Moussiades. "An Overview of Chatbot Technology". In: *IFIP Advances in Information and Communication Technology*. Vol. 584 IFIP. Springer, 2020, pp. 373–383. ISBN: 9783030491857. DOI: 10.1007/978-3-030-49186-4_{_}31.
- [8] Daniel Adiwardana et al. *Towards a Human-like Open-Domain Chatbot*. URL: <http://arxiv.org/abs/2001.09977>.
- [9] *AI in healthcare market size worldwide 2030 | Statista*. URL: <https://www.statista.com/statistics/1334826/ai-in-healthcare-market-size-worldwide/>. Date accessed: 14/05/2023.
- [10] *Amazon Mechanical Turk*. URL: <https://www.mturk.com/>. Date accessed: 01/05/2023.
- [11] *Amazon mTurk - Problem to pay "You have exceeded your monthly credit limit" | AWS re:Post*. URL: <https://repost.aws/questions/QU0XxBxGDQ3WTSdCtBHL4Iw/questions/QU0XxBxGDQ3WTSdCtBHL4Iw/amazon-mturk-problem-to-pay-you-have-exceeded-your-monthly-credit-limit?>. Date accessed: 07/04/2023.
- [12] *Aphasia Definitions - National Aphasia Association*. URL: <https://www.aphasia.org/aphasia-definitions/>. Date accessed: 23/11/2022.
- [13] *API Reference - OpenAI API*. URL: <https://platform.openai.com/docs/api-reference/completions/create>. Date accessed: 07/04/2023.

Bibliography

- [14] Satanjeev Banerjee and Alon Lavie. *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments*. URL: <https://aclanthology.org/W05-0909>.
- [15] Jose Belda-Medina and José Ramón Calvo-Ferrer. “Using Chatbots as AI Conversational Partners in Language Learning”. In: *Applied Sciences (Switzerland)* 12.17 (Sept. 2022). ISSN: 20763417. DOI: 10.3390/app12178427.
- [16] Yoshua Bengio, Patrice Simard and Paolo Frasconi. *Learning Long-Term Dependencies with Gradient Descent is Difficult*. *IEEE TRANSACTIONS ON NEURAL NETWORKS*.
- [17] Rita Sloan Berndt and Alfonso Caramazza. “A redefinition of the syndrome of Broca’s aphasia: Implications for a neuropsychological model of language”. In: *Applied Psycholinguistics* 1.3 (1980), pp. 225–278. ISSN: 14691817. DOI: 10.1017/S0142716400000552.
- [18] *Best practices for fine-tuning GPT-3 to classify text - Google Docs*. URL: <https://docs.google.com/document/d/1rqj7dkuvl7Byd5KQPURxc19BJt8wo0yHNwK84KfU3Q/edit#>. Date accessed: 25/03/2023.
- [19] Bin Bi et al. *Incorporating External Knowledge into Machine Reading for Generative Question Answering*. URL: <http://arxiv.org/abs/1909.02745>.
- [20] Sid Black et al. “GPT-NeoX-20B: An Open-Source Autoregressive Language Model”. In: *2022 Challenges and Perspectives in Creating Large Language Models, Proceedings of the Workshop* (Apr. 2022), pp. 95–136. DOI: 10.18653/v1/2022.bigscience-1.9. URL: <https://arxiv.org/abs/2204.06745v1>.
- [21] Tom B Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.
- [22] *Cambridge English*. URL: <https://www.cambridgeenglish.org/>. Date accessed: 18/03/2023.
- [23] Ricardo Campos et al. “YAKE! Keyword extraction from single documents using multiple local features”. In: *Information Sciences* 509 (Jan. 2020), pp. 257–289. ISSN: 00200255. DOI: 10.1016/j.ins.2019.09.013.
- [24] *Character-level recurrent sequence-to-sequence model*. URL: https://keras.io/examples/nlp/lstm_seq2seq/. Date accessed: 10/04/2023.
- [25] Hao Cheng et al. “UnitedQA: A Hybrid Approach for Open Domain Question Answering”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing* 1 (2021), pp. 3080–3090. DOI: 10.18653/v1/2021.acl-long.240.

- [26] Saeyeon Cheon and Insung Ahn. “Fine-Tuning BERT for Question and Answering Using PubMed Abstract Dataset”. In: *2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)* (2022), pp. 681–684. DOI: 10.23919/APSIPAASC55919.2022.9980097.
- [27] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing ({EMNLP})* (2014), pp. 1724–1734.
- [28] Eunsol Choi et al. “QuAC : Question Answering in Context”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018), pp. 2174–2184. URL: <http://quac.ai..>
- [29] Francois et al. Chollet. *keras-team/keras: Deep Learning for humans*. URL: <https://github.com/keras-team/keras>. Date accessed: 10/04/2023.
- [30] *Cloud computing, evolved | Paperspace*. URL: <https://www.paperspace.com/>. Date accessed: 10/04/2023.
- [31] *Communication problems after stroke | Stroke Association*. URL: <https://www.stroke.org.uk/effects-of-stroke/communication-problems>. Date accessed: 14/05/2023.
- [32] *conversational-datasets/amazon_qa at master · PolyAI-LDN/conversational-datasets*. URL: https://github.com/PolyAI-LDN/conversational-datasets/tree/master/amazon_qa. Date accessed: 08/04/2023.
- [33] *conversational-datasets/opensubtitles at master · PolyAI-LDN/conversational-datasets*. URL: <https://github.com/PolyAI-LDN/conversational-datasets/tree/master/opensubtitles>. Date accessed: 08/04/2023.
- [34] *conversational-datasets/reddit at master · PolyAI-LDN/conversational-datasets*. URL: <https://github.com/PolyAI-LDN/conversational-datasets/tree/master/reddit>. Date accessed: 08/04/2023.
- [35] Antonio R. Damasio. “Aphasia”. In: *New England Journal of Medicine* 326.8 (Feb. 1992), pp. 531–539. ISSN: 0028-4793. DOI: 10.1056/NEJM199202203260806. URL: <http://www.nejm.org/doi/abs/10.1056/NEJM199202203260806>.
- [36] Sumanth Dathathri et al. “Plug and Play Language Models: A Simple Approach to Controlled Text Generation”. In: *CoRR* (Dec. 2019). URL: <http://arxiv.org/abs/1912.02164>.
- [37] Thomas Davenport and Ravi Kalakota. “The potential for artificial intelligence in healthcare”. In: *Future Healthcare Journal* 6.2 (June 2019), p. 94. ISSN: 2514-6645. DOI: 10.7861/FUTUREHOSP.6-2-94. URL: </pmc/articles/PMC6616181/%20/pmc/articles/PMC6616181/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6616181/>.
- [38] *Deepnote: Analytics and data science notebook for teams*. URL: <https://deepnote.com/>. Date accessed: 10/04/2023.

Bibliography

- [39] Tim Dettmers et al. “8-bit Optimizers via Block-wise Quantization”. In: *arXiv preprint arXiv:2110.02861* (Oct. 2021). URL: <https://arxiv.org/abs/2110.02861v2>.
- [40] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. URL: <https://github.com/tensorflow/tensor2tensor>.
- [41] Emily Dinan et al. “The Second Conversational Intelligence Challenge (ConvAI2)”. In: *The Second Conversational Intelligence Challenge (ConvAI2)* (Jan. 2020), pp. 187–208. URL: <http://arxiv.org/abs/1902.00098>.
- [42] *Docker: Accelerated, Containerized Application Development*. URL: <https://www.docker.com/>. Date accessed: 09/04/2023.
- [43] *EleutherAI/gpt-j-6b · Hugging Face*. URL: <https://huggingface.co/EleutherAI/gpt-j-6b>. Date accessed: 09/04/2023.
- [44] *EleutherAI/gpt-neo-2.7B · Hugging Face*. URL: <https://huggingface.co/EleutherAI/gpt-neo-2.7B>. Date accessed: 09/04/2023.
- [45] *ELLLO - English Listening Lesson Library Online*. URL: <https://elllo.org/index.htm>. Date accessed: 18/03/2023.
- [46] Pam Enderby. “Disorders of communication: dysarthria”. In: *Handbook of clinical neurology* 110 (2013), pp. 273–281.
- [47] *ESL Fast - English Teaching and Learning Resources*. URL: <https://www.eslfast.com/>. Date accessed: 18/03/2023.
- [48] *Fine-tuning - OpenAI API*. URL: <https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset>. Date accessed: 19/03/2023.
- [49] *finetune-gpt-j-6B-8bit.ipynb - Colaboratory*. URL: <https://colab.research.google.com/drive/1ft6wQU0BhqG5PRlwgaZJv2VukKKjU4Es>. Date accessed: 17/12/2022.
- [50] Julius Fridriksson and Argye Elizabeth Hillis. “Current approaches to the treatment of post-stroke aphasia”. In: *Journal of Stroke* 23.2 (2021), pp. 183–201. ISSN: 22876405. DOI: 10.5853/jos.2020.05015.
- [51] Julius Fridriksson et al. “Anatomy of aphasia revisited”. In: *Brain* 141.3 (Mar. 2018), pp. 848–862. ISSN: 14602156. DOI: 10.1093/brain/awx363.
- [52] Romain Futrzynski. *Self-attention: step-by-step video | Peltarion*. URL: <https://peltarion.com/blog/data-science/self-attention-video>. Date accessed: 28/11/2022.
- [53] Gagan Bhatia. *GitHub - gagan3012/keytotext: Keywords to Sentences*. URL: <https://github.com/gagan3012/keytotext>. Date accessed: 21/09/2022.
- [54] Jianfeng Gao, Michel Galley and Lihong Li. “Neural approaches to conversational AI”. In: *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference Tutorial Abstracts*. 2018. DOI: 10.18653/v1/p18-5002.

- [55] Xiang Gao et al. *Structuring Latent Spaces for Stylized Response Generation*.
- [56] Anthony Gillioz et al. “Overview of the Transformer-based Models for NLP Tasks”. In: *Proceedings of the 2020 Federated Conference on Computer Science and Information Systems, FedCSIS 2020*. Institute of Electrical and Electronics Engineers Inc., Sept. 2020, pp. 179–183. ISBN: 9788395541674. DOI: 10.15439/2020F20.
- [57] *GitHub*. URL: <https://github.com/>. Date accessed: 01/06/2023.
- [58] *GitHub - lukalabs/cakechat: CakeChat: Emotional Generative Dialog System*. URL: <https://github.com/lukalabs/cakechat>. (Date accessed: 01/12/2022).
- [59] *GPT_Neo_fine-tuning_notebook/GPT_Neo_Fine-tune.ipynb at main · mallorbc/GPT_Neo_fine-tuning_notebook*. URL: https://github.com/mallorbc/GPT_Neo_fine-tuning_notebook/blob/main/GPT_Neo_Fine-tune.ipynb. Date accessed: 10/04/2023.
- [60] *Gradio*. URL: <https://gradio.app/>. Date accessed: 09/04/2023.
- [61] *Gradio Docs*. URL: <https://gradio.app/docs/>. Date accessed: 14/05/2023.
- [62] Isel Grau et al. “Backpropagation through Time Algorithm for Training Recurrent Neural Networks using Variable Length Instances”. In: *Computación y Sistemas* 17.1 (Mar. 2013), pp. 15–24. ISSN: 2007-9737. DOI: 10.13053/cys-17-1-1477. URL: <https://www.cys.cic.ipn.mx/ojs/index.php/CyS/article/view/1477>.
- [63] Alex Graves. “Long Short-Term Memory”. In: *Supervised sequence labelling with recurrent neural networks* (2012), pp. 37–45. DOI: 10.1007/978-3-642-24797-2_4. URL: https://link.springer.com/chapter/10.1007/978-3-642-24797-2_4.
- [64] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012. ISBN: 978-3-642-24796-5.
- [65] Matthew Henderson et al. *A Repository of Conversational Datasets* github.com/PolyAI-LDN/conversational-datasets.
- [66] *hivemind/gpt-j-6B-8bit · Hugging Face*. URL: <https://huggingface.co/hivemind/gpt-j-6B-8bit>. Date accessed: 10/04/2023.
- [67] *How AI software is making its way into language learning - Goethe-Institut*. URL: <https://www.goethe.de/en/spr/spr/24202694.html>. Date accessed: 29/11/2022.
- [68] *How to Fine-tune GPT-3 to Get Better Results and Save Cost | NLP | Python | #gpt3 #openai - YouTube*. URL: <https://www.youtube.com/watch?v=muxtjezs3BQ&t=1146s>. Date accessed: 08/04/2023.
- [69] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. URL: <https://arxiv.org/abs/2106.09685v2>.
- [70] Minlie Huang, Xiaoyan Zhu and Jianfeng Gao. “Challenges in Building Intelligent Open-domain Dialog Systems”. In: *ACM Transactions on Information Systems* 38.3 (June 2020). ISSN: 15582868. DOI: 10.1145/3383123.

Bibliography

- [71] *Hugging Face – The AI community building the future*. URL: <https://huggingface.co/>. Date accessed: 09/04/2023.
- [72] *Introducing ChatGPT*. URL: <https://openai.com/blog/chatgpt>. Date accessed: 16/05/2023.
- [73] Chenglin Jiang et al. “An affective chatbot with controlled specific emotion expression”. In: *Science China Information Sciences* 65.10 (Oct. 2022). ISSN: 18691919. DOI: 10.1007/s11432-020-3356-4.
- [74] Zahraa Saddi Kadhim, Hasanen S. Abdullah and Khalil Ibrahim Ghathwan. “Artificial Neural Network Hyperparameters Optimization: A Survey”. In: *International journal of online and biomedical engineering* 18.15 (2022), pp. 59–87. ISSN: 26268493. DOI: 10.3991/ijoe.v18i15.34399.
- [75] *Kaggle: Your Machine Learning and Data Science Community*. URL: <https://www.kaggle.com/>. Date accessed: 10/04/2023.
- [76] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. URL: <http://arxiv.org/abs/2001.08361>.
- [77] *Keywords to Text With GPT-Neo*. URL: <https://www.vennify.ai/keyword-to-text-gpt-neo/>. Date accessed: 17/12/2022.
- [78] Salman Khan et al. “Transformers in Vision: A Survey”. In: *ACM Computing Surveys* 54.10s (2022). DOI: 10.1145/3505244. URL: <https://doi.org/10.1145/3505244>.
- [79] Hema Krishnan and M Sudheep Elayidom. “MongoDB-a comparison with NoSQL databases Detection of Community support or dissent in social systems of a government decision using blockchain View project MongoDB-a comparison with NoSQL databases”. In: *International Journal of Scientific & Engineering Research* 7.5 (2016). ISSN: 2229-5518. URL: <http://www.ijser.org>.
- [80] *Learn English Online | British Council*. URL: <https://learnenglish.britishcouncil.org/>. Date accessed: 18/03/2023.
- [81] *Levenshtein distance*. URL: <https://xlinux.nist.gov/dads/HTML/Levenshtein.html>. Date accessed: 16/04/2023.
- [82] Patrick Lewis, Ludovic Denoyer and Sebastian Riedel. *Unsupervised Question Answering by Cloze Translation*. URL: <http://arxiv.org/abs/1906.04980>. DOI: 10.18653/v1/P19-1484. URL: <http://dx.doi.org/10.18653/v1/P19-1484>.
- [83] Jiwei Li et al. *A Persona-Based Neural Conversation Model*.
- [84] Yanran Li et al. *DailyDialog: A Manually Labelled Multi-turn Dialogue Dataset*. URL: <http://yanran.li/>.
- [85] Chin-Yew Lin. *ROUGE: A Package for Automatic Evaluation of Summaries*. URL: <https://aclanthology.org/W04-1013>.

- [86] Tianyang Lin et al. “A Survey of Transformers”. In: *AI Open* (June 2022). ISSN: 26666510. DOI: 10.1016/j.aiopen.2022.10.001. URL: <http://arxiv.org/abs/2106.04554>.
- [87] *Lingua.com - Free tools for language learning*. URL: <https://lingua.com/>. Date accessed: 18/03/2023.
- [88] *Listening Inglés - Tests con todos los Niveles de Inglés*. URL: <https://listeningingles.com/>. Date accessed: 18/03/2023.
- [89] Andrea Madotto et al. *Findings of the Association for Computational Linguistics Plug-and-Play Conversational Models*.
- [90] Andrea Madotto et al. “Personalizing Dialogue Agents via Meta-Learning”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), pp. 5454–5459.
- [91] Yuning Mao et al. *Generation-Augmented Retrieval for Open-domain Question Answering*. URL: <http://arxiv.org/abs/2009.08553>.
- [92] Mathew Alexander. *Data to Text generation with T5; Building a simple yet advanced NLG model* / by Mathew Alexander / Towards Data Science. URL: <https://towardsdatascience.com/data-to-text-generation-with-t5-building-a-simple-yet-advanced-nlg-model-b5cce5a6df45>. (Date accessed: 27/11/2022).
- [93] *MongoDB: Die Plattform Für Anwendungsdaten* / MongoDB. URL: <https://www.mongodb.com/de-de>. Date accessed: 09/04/2023.
- [94] Aishwarya Nair and Jaya S Nirmala. “A Question Answering and Quiz Generation Chatbot for Education; A Question Answering and Quiz Generation Chatbot for Education”. In: *2019 Grace Hopper Celebration India (GHCI)* (2019).
- [95] Jinjie Ni et al. “Recent Advances in Deep Learning Based Dialogue Systems: A Systematic Survey”. In: *Artificial intelligence review* (May 2023), pp. 3055–3155. URL: <http://arxiv.org/abs/2105.04387>.
- [96] NIDCD Information Clearinghouse. *What Is Aphasia? — Types, Causes and Treatment*. URL: <https://www.nidcd.nih.gov/health/aphasia>. NIH Pub. No. 97-4257 (Date accessed: 23/11/2022).
- [97] Nir Ben-Zvi. *How To Increase GPU Quota* – Nir Ben-Zvi – Computer Vision Researcher. URL: <https://nirbenz.github.io/gce-quota/>. Date accessed: 10/04/2023.
- [98] *NLP Cloud Playground*. URL: <https://nlpcloud.com/home/playground/chatbot-and-conversational-ai>. Date accessed: 28/12/2022.
- [99] *openai/openai-cookbook: Examples and guides for using the OpenAI API*. URL: <https://github.com/openai/openai-cookbook>. Date accessed: 08/04/2023.
- [100] Eva Ostertagova and Oskar Ostertag. “Methodology and Application of One-way ANOVA”. In: *American Journal of Mechanical Engineering* 1 (2013), pp. 256–261. URL: https://www.researchgate.net/publication/259291691_Methodology_and_Application_of_One-way_ANOVA.

Bibliography

- [101] Sebastião Pais, João Cordeiro and M. Luqman Jamil. “NLP-based platform as a service: a brief review”. In: *Journal of Big Data* 9.1 (Dec. 2022). ISSN: 21961115. DOI: 10.1186/S40537-022-00603-5.
- [102] Ilias Papathanasiou and Patrick Coppens. *Aphasia and related neurogenic communication disorders*. 2021. ISBN: 9781284184099.
- [103] Kishore Papineni et al. “BLEU: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (2002), pp. 311–318.
- [104] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in neural information processing systems* 32 (2019).
- [105] Prakhar Mishra. *Generating Sentences from Keywords using Transformers in NLP* | by Prakhar Mishra | MLearning.ai | Medium. URL: <https://medium.com/mllearning-ai/generating-sentences-from-keywords-using-transformers-in-nlp-e89f4de5cf6b>. Date accessed: 27/11/2022.
- [106] *Pricing Advanced Artificial Intelligence API*. URL: <https://nlpccloud.com/#plans>. Date accessed: 08/04/2023.
- [107] *Pricing OpenAI*. URL: <https://openai.com/pricing>. Date accessed: 08/04/2023.
- [108] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: *OpenAI blog* 1 (2019). URL: <https://github.com/codelucas/newspaper>.
- [109] Jack W. Rae et al. *Scaling Language Models: Methods, Analysis & Insights from Training Gopher*. URL: <http://arxiv.org/abs/2112.11446>.
- [110] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *Journal of Machine Learning Research* 21 (2020), pp. 1–67. URL: <http://jmlr.org/papers/v21/20-074.html>.
- [111] Pranav Rajpurkar, Robin Jia and Percy Liang. *Know What You Don’t Know: Unanswerable Questions for SQuAD*. URL: <http://arxiv.org/abs/1806.03822>.
- [112] Jeff Rasley et al. “DeepSpeed: System Optimizations Enable Training Deep Learning Models with over 100 Billion Parameters”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug. 2020), pp. 3505–3506. DOI: 10.1145/3394486.3406703. URL: <https://dl.acm.org/doi/10.1145/3394486.3406703>.
- [113] richmilne/JaroWinkler: *Original, standard and customisable versions of the Jaro-Winkler functions*. URL: <https://github.com/richmilne/JaroWinkler>. Date accessed: 16/04/2023.
- [114] Stuart Rose et al. “Automatic keyword extraction from individual documents”. In: *Text Mining: Applications and Theory* (2010), pp. 1–20.
- [115] Mieke W.M.E. van de Sandt-Koenderman. *High-tech AAC and aphasia: Widening horizons? Aphasiology*. ISSN: 02687038. DOI: 10.1080/02687030344000571.

- [116] *seatgeek/thefuzz: Fuzzy String Matching in Python*. URL: <https://github.com/seatgeek/thefuzz>. Date accessed: 16/04/2023.
- [117] *Semantic Similarity with BERT*. URL: https://keras.io/examples/nlp/semantic_similarity_with_bert/. Date accessed: 16/04/2023.
- [118] Prafull Sharma and Yingbo Li. *Self-Supervised Contextual Keyword and Keyphrase Retrieval with Self-Labelling*. URL: <https://www.preprints.org/manuscript/201908.0073/v1>. DOI: 10.20944/PREPRINTS201908.0073.V1.
- [119] Kurt Shuster et al. *BlenderBot 3: a deployed conversational agent that continually learns to responsibly engage*. URL: <http://arxiv.org/abs/2208.03188>.
- [120] Daniela America da Silva et al. “Could a conversational ai identify offensive language?”. In: *Information (Switzerland)* 12.10 (2021). ISSN: 20782489. DOI: 10.3390/info12100418.
- [121] *spaCy · Industrial-strength Natural Language Processing in Python*. URL: <https://spacy.io/>. Date accessed: 16/04/2023.
- [122] *Speak English Fluently with our FREE Basic English Speaking Lessons - Basic English Speaking*. URL: <https://basicenglishspeaking.com/>. Date accessed: 18/03/2023.
- [123] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. URL: <http://arxiv.org/abs/1909.09586>.
- [124] Nadezda Sternic et al. “Dysarthria and Mutism”. In: 30 (2012), pp. 83–85. DOI: 10.1159/000333422. URL: <https://www.karger.com/Article/FullText/333422%20https://www.karger.com/Article/Abstract/333422>.
- [125] Chi Sun et al. “How to Fine-Tune BERT for Text Classification?” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11856 LNAI (May 2019), pp. 194–206. ISSN: 16113349. DOI: 10.1007/978-3-030-32381-3_{_}16. URL: <https://arxiv.org/abs/1905.05583v3>.
- [126] Ilya Sutskever, James Martens and Geoffrey Hinton. “Generating Text with Recurrent Neural Networks”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)* (2011), pp. 1017–1024.
- [127] Ilya Sutskever Google, Oriol Vinyals Google and Quoc V Le Google. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in neural information processing systems* 27 (2014).
- [128] Yi Tay et al. “Efficient Transformers: A Survey”. In: *ACM Computing Surveys* (Apr. 2022). ISSN: 0360-0300. DOI: 10.1145/3530811.
- [129] *Text Generation With LSTM Recurrent Neural Networks in Python with Keras - MachineLearningMastery.com*. URL: <https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/>. Date accessed: 10/04/2023.

Bibliography

- [130] Romal Thoppilan et al. *LaMDA: Language Models for Dialog Applications*. URL: <http://arxiv.org/abs/2201.08239>.
- [131] M Onat Topal, Anil Bas and Imke Van Heerden. *Exploring Transformers in Natural Language Generation: GPT, BERT, and XLNet*. URL: <https://arxiv.org/abs/2102.08036>. Date accessed: 24/11/2022.
- [132] *Topics extraction & named entity recognition | MeaningCloud*. URL: <https://www.meaningcloud.com/products/topics-extraction>. Date accessed: 27/04/2023.
- [133] Kiyotaka Uchimoto and Text Generation from Keywords. “Text Generation from Keywords”. In: *COLING 2002: The 19th International Conference on Computational Linguistics* (2002).
- [134] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in neural information processing systems* 30 (2017).
- [135] *Weights & Biases – Developer tools for ML*. URL: <https://wandb.ai/site>. Date accessed: 08/04/2023.
- [136] *Welcome To Colaboratory - Colaboratory*. URL: <https://colab.research.google.com/>. Date accessed: 10/04/2023.
- [137] Paul J Werbos. “Backpropagation Through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.
- [138] *What is Question Answering? - Hugging Face*. URL: <https://huggingface.co/tasks/question-answering>. Date accessed: 28/11/2022.
- [139] Thomas Wolf et al. *TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents*. URL: <http://arxiv.org/abs/1901.08149>.
- [140] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (2020), pp. 38–45.
- [141] *Xirider/finetune-gpt2xl: Guide: Finetune GPT2-XL (1.5 Billion Parameters) and finetune GPT-NEO (2.7 B) on a single GPU with Huggingface Transformers using DeepSpeed*. URL: <https://github.com/Xirider/finetune-gpt2xl>. Date accessed: 09/04/2023.
- [142] Yichong Xu et al. *Human Parity on CommonsenseQA: Augmenting Self-Attention with External Attention*. URL: <https://github.com/jessionlin/csqa>.
- [143] Li Yang and Abdallah Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. In: *Neurocomputing* 415 (Nov. 2020), pp. 295–316. ISSN: 18728286. DOI: 10.1016/j.neucom.2020.07.061.
- [144] Bill Yuchen Lin et al. *EMNLP-Findings’20 CommonGen: A Constrained Text Generation Challenge for Generative Commonsense Reasoning*. URL: <http://inklab.usc.edu/CommonGen/>.

- [145] Hamada M Zahera et al. “Fine-tuned BERT Model for Multi-Label Tweets Classification”. In: *TREC* (2019), pp. 1–7. URL: https://www.nltk.org/_modules/nltk/stem/wordnet.html.
- [146] *zeloru/small-english-smalltalk-corpus: small english smalltalk corpus for traning chat bot*. URL: <https://github.com/zeloru/small-english-smalltalk-corpus>. Date accessed: 17/12/2022.
- [147] *Zero-Shot Learning in Modern NLP | Joe Davison Blog*. URL: <https://joeddav.github.io/blog/2020/05/29/ZSL.html>. Date accessed: 16/04/2023.
- [148] Peixiang Zhong et al. “CARE: Commonsense-Aware Emotional Response Generation with Latent Concepts”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (2021), pp. 14577–14585. URL: www.aaai.org.

Appendices

A. Appendix

A.1. Keyword Extractor Settings

```
1 def kw_yake(text):
2     language = "en"
3     max_ngram_size = 1
4     deduplication_threshold = 0.9
5     numOfKeywords = 10
6     custom_kw_extractor = yake.KeywordExtractor(lan=language,
7         n=max_ngram_size, dedupLim=deduplication_threshold,
8         top=numOfKeywords, features=None)
9     keywords = custom_kw_extractor.extract_keywords(text)
10    keywords_str = ""
11    for index, keyword in enumerate(keywords):
12        if index == len(keywords) - 1:
13            keywords_str += f"{keyword[0]}"
14        else:
15            keywords_str += f"{keyword[0]} "
16    return keywords_str
17
18
19 def kw_yake_detailed(text):
20     language = "en"
21     max_ngram_size = 5
22     deduplication_threshold = 0.0
23     numOfKeywords = 10
24     custom_kw_extractor = yake.KeywordExtractor(lan=language,
25         n=max_ngram_size, dedupLim=deduplication_threshold,
26         top=numOfKeywords, features=None)
27     keywords = custom_kw_extractor.extract_keywords(text)
28     keywords_str = ""
29     for index, keyword in enumerate(keywords):
30         if index == len(keywords) - 1:
31             keywords_str += f"{keyword[0]}"
32         else:
33             keywords_str += f"{keyword[0]} "
34     return keywords_str
35
36
37
38
39
40
41
42
```

A. Appendix

```
43 def kw_rake(text):
44     rake = Rake()
45     keywords = rake.apply(text)
46     keywords_str = ""
47     for index, keyword in enumerate(keywords):
48         if index == len(keywords) - 1:
49             keywords_str += f"{keyword[0]}"
50         else:
51             keywords_str += f"{keyword[0]} "
52     return (keywords_str)
53
54
55 def kw_keybert(text):
56     keywords = kw_model.extract_keywords(
57         text,
58         keyphrase_ngram_range=(1, 7),
59         stop_words='english',
60         highlight=False,
61         top_n=1)
62     keywords_list = list(dict(keywords).keys())
63     if not keywords_list:
64         return ""
65     return (keywords_list[0])
```

A.2. Topics in Conversation Dataset

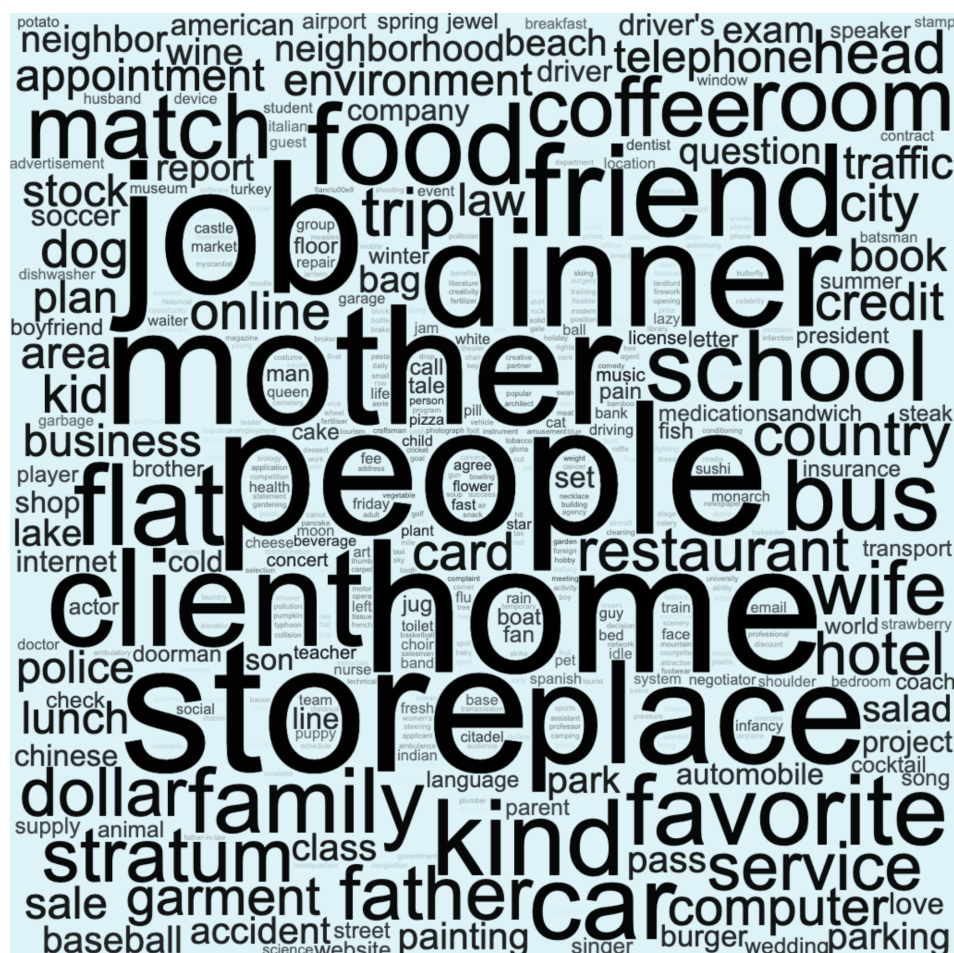



Figure A.1.: World cloud of different topics of the conversation data

A.3. Use Case Instructions


Scenario	Description to GPT-3
Book a table	You are the waiter of a Restaurant called Giovanni's Pizza and get a phone call. Ask all the questions needed to make a reservation for the customer including: The date, the name, amount of people, and if they would like to eat from the menu or buffet. Be friendly, polite, and talkative. Ask one question at the time.
Hairdresser	You are working in a hair saloon and get a phone call. Ask all the questions needed to make an appointment for the customer, including the name, if the customer was in your store before, the time for the appointment, if the customer wants to have a head massage included. Be friendly, polite, and talkative. Ask one question at the time.
Doctor	You are named Lisa and are working in a doctor's practice called Dr. Pepper. A patient wants to make an appointment. Ask all the questions needed to the package correctly labelled and shipped: if the person is ill at the moment, if he or she has fever or other symptoms, when the patient wants to come, if he or she was here before (and when), say that you call him or her back. Be friendly, polite and talkative. Ask one question at the time.
Order food	You are the waiter of an Indian Restaurant called Tikka on Wheels and get a phone call from someone who wants to order food. Ask all the questions needed to complete the order for the customer including: The dish, the street for delivering, some additional drinks if he or she wants to pick up the order or delivery. Be friendly, polite and talkative. Ask one question at the time.
Hobbies	You are having a phone call with a new friend and you talk about hobbies. Ask all the following questions: When did he do this the first time, how often he or she does that hobby, how good he is in the hobby, what is the best way to learn it. Be friendly, polite and talkative. Ask one question at the time.
Calling an old friend	You are getting a phone call from an old friend. Ask all the following questions: what is the friend doing for a living, how is it going with his family, where is he living currently, if he wants to meet soon. Be friendly, polite and talkative. Ask one question at the time.
Customer service	You are working in a customer service for an online retailer named Mississippi and a customer has a problem. Ask all the questions needed to help the customer solve the problem. However, DO NOT ask about: An Order Number, to describe the product received. Be friendly, polite and talkative. Ask one question at the time.
Auto shop	You are a mechanic of an auto shop called Wheely Great. A customer wants to get his or her car repaired. Ask all the questions needed to get the repair done: The car brand, how old the car is, if was broken before, when the car needs to be ready. Be friendly, polite and talkative. Ask one question at the time.

Table A.1.: Use case instructions for GPT-3


A.4. Settings Model Comparison

Use GPU 


☒

Length no input 


☒


Remove input 


☐


Do sample 

☒


Min length 


Max length 


End sequence 


Remove end sequence 

☐

Bad words (comma separated words) 

Num returned sequences 

Top k 

Top p 


Temperature 

Figure A.2.: GPT-J and GPT-Neo settings

A. Appendix

Mode

☰ Complete ▼

Model

text-davinci-003 ▼

Temperature 0.7

Maximum length 1360

Stop sequences

Enter sequence and press Tab

Question × ×

Top P 1

Frequency penalty 0

Presence penalty 0

Best of 1

Inject start text

✓

Inject restart text

✓

Show probabilities

Off ▼

Figure A.3.: OpenAI Settings

A.5. Next Tasks Experiment

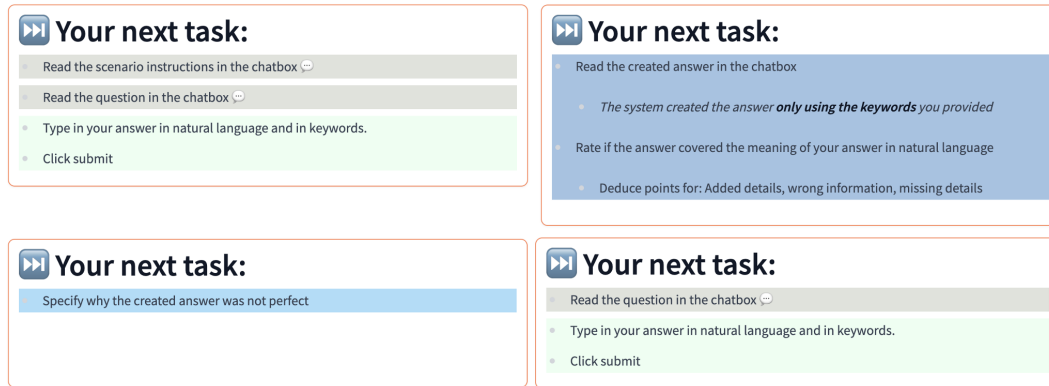


Figure A.4.: All next task instructions in the experiment

A.6. Pre-Testing Interview

1. Read through the Explanation and step by step guide once

Without making an example, do you understand the following?

- Did you get the gist of the experiment and what you have to do
- Do you get what "Keywords" are?
- Would a short video clip help you to understand the task better?
- Do you know what the system is supposed to do?

2. Make one example together

- Do you know what to rate?
 - What things to punish?
- Do you find the keyword suggestions useful?
- Do you know what the different scenarios are?
- Do you know when the experiment is finished?
- Do you know how to handle errors?
- Do you know how to handle the attention check?

3. Other suggestions / complaints?

- Any kind of other scenario for this use case?

Figure A.5.: Pre-Testing interview guideline

A.7. Mechanical Turk Best Practices, Explanations, and Cost

To create a survey as a requester, the following requirements have to be met and steps need to be taken in order to be allowed to create a survey for workers.

1. An AWS (Amazon Web Services) account
2. A request and confirmation from Amazon to approve the monthly budget for MTurk
3. Settings for the survey (monetary, time, etc)
4. A layout for the experiment (see section 6.8.2)
5. Appropriate methods for evaluating whether the workers have carried out the experiment conscientiously (see section 6.8.2)

Step 1 is self-explanatory. To open an account, only an email address and a suitable payment method is needed. By default, the monthly budget of an MTurk account is limited to \$ 0. To change this, a request must be made to Amazon to adjust the monthly budget to the desired amount [11]. After the approval, a new survey can be created. First, the basic settings have to be configured, e.g.

- Name of the survey
- Short description
- Time needed per experiment per participant
- How many answers you wish to receive
- Until when the experiment should be open

Important key figures such as the reward a worker receives for a survey are set. In addition, qualitative factors that a worker must fulfil can be specified. For example, it can be specified how many surveys a worker must have already completed. What worked very well in the first experience was to only ask for workers who had already successfully passed more than 50 surveys. Furthermore, Fig. A.6 provides an overview of the other settings that worked well in practise.

A total of €48.65 was spent on MTurk surveys. This means that excluding all unusable results (whether provided a wrong code, unfinished, etc.), about €0.80 was paid for a survey with a usable data.

A.7. Mechanical Turk Best Practices, Explanations, and Cost

Describe your survey to Workers

Title

Describe the survey to Workers. Be as specific as possible, e.g. "answer a survey about movies", instead of "short survey", so Workers know what to expect.

Description

Give more detail about this survey. This gives Workers a bit more information before they decide to view your survey.

Keywords

Provide keywords that will help Workers search for your tasks.

Setting up your survey

Reward per response

This is how much a Worker will be paid for completing your survey. Consider how long it will take a Worker to complete your survey.

Number of respondents

How many unique Workers do you want to complete your survey?

Time allotted per Worker

Maximum time a Worker has to complete the survey. Be generous so that Workers are not rushed.

Survey expires in

Maximum time your survey will be available to Workers on Mechanical Turk.

Auto-approve and pay Workers in

This is the amount of time you have to reject a Worker's assignment after they submit the assignment.

Worker requirements

Require that Workers be Masters to do your tasks ([Who are Mechanical Turk Masters?](#))

☒ Yes ☐ No

Specify any additional qualifications Workers must meet to work on your tasks:

[Remove](#)

[\(+\)](#) **Add another criterion** (up to 4 more)

(Premium Qualifications incur additional fees, see [Pricing Details](#) to learn more)

Project contains adult content ([See details](#))

☐ This project may contain potentially explicit or offensive content, for example, nudity.

Task Visibility ([What is task visibility?](#))

☒ **Public** - All Workers can see and preview my tasks

☐ **Private** - All Workers can see my tasks, but only Workers that meet all Qualification requirements can preview my tasks

☐ **Hidden** - Only Workers that meet my Qualification requirements can see and preview my tasks

Figure A.6.: Amazon Mechanical Turk settings

A.8. Distribution of Human Ratings

Rating	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
1	0.43 %	2.88 %	3.31 %	5.39 %	2.58 %
2	1.74 %	3.7 %	3.72 %	7.88 %	5.58 %
3	3.91 %	7.41 %	10.74 %	11.2 %	8.15 %
4	23.04 %	23.87 %	23.97 %	23.24 %	19.74 %
5	70.87 %	62.14 %	58.26 %	52.28 %	63.95 %

Table A.2.: Distribution of human ratings for different models

A.9. ANOVA Results

	sum_sq	df	F	PR(>F)
C(models)	34.509565	4.000000	8.644460	0.000001
Residual	1142.739130	1145.000000	NaN	NaN

Table A.3.: ANOVA test results between all models

A.10. Distribution of Detailed Ratings

Rating	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
Slightly different	37.31 %	28.26 %	34.65 %	17.39 %	45.24 %
Too brief/short	29.85 %	10.87 %	25.74 %	8.7 %	16.67 %
Missed details	14.93 %	8.7 %	11.88 %	9.57 %	4.76 %
Added details	8.96 %	33.7 %	11.88 %	42.61 %	15.48 %
Wrong information	7.46 %	13.04 %	9.9 %	14.78 %	15.48 %
Totally different	7.46 %	13.04 %	9.9 %	14.78 %	15.48 %

Table A.4.: Distribution of human detailed-ratings for all models

A.11. Standard Deviations of Automatic Evaluation

	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
BERT	0.2042	0.1970	0.2035	0.22	0.1861
FUZZ	0.1841	0.1869	0.1794	0.1633	0.1682
JARO	0.1097	0.0773	0.0951	0.0737	0.0917
BLEU	0.2379	0.2147	0.2395	0.1873	0.2302
METEOR	0.2563	0.2072	0.2578	0.1820	0.2447
ROUGE	0.2563	0.2278	0.2618	0.2031	0.2284
SPACY	0.1891	0.1317	0.1669	0.1462	0.1596
Human (Punished)	1.0532	1.5117	1.4188	1.7435	1.4029
Human	0.6870	0.9784	1.0246	1.1972	1.0219

Table A.5.: Standard deviations of automatic evaluation metrics

A.12. Standard Deviations of History Length and Rating

Model	2	4	6
FT-HQ	0.7723	0.6152	0.6643
FT-CR	1.0329	0.8668	1.0476
FT-XL	0.8430	1.1153	1.0946
STOCK-1	1.1869	1.2470	1.1742
STOCK-2	1.1793	0.8363	1.0688

Table A.6.: Standard deviations of ratings depending on the history length

A.13. Keywords and Rating

A.13.1. Standard Deviations

	1	2	3	4	≥ 5
FT-HQ	0.6203	0.5899	0.8686	0.6219	0.7071
FT-CR	0.9570	0.9125	1.0725	1.33	1.0725
FT-XL	0.9784	0.9068	1.1391	0.6580	1.3611
STOCK-1	0.9189	1.0706	1.4999	1.2039	1.2397
STOCK-2	0.7817	0.8398	1.1260	1.3272	1.3445

Table A.7.: Standard deviations for ratings depending on the number of keywords

A. Appendix

A.13.2. Amount of Ratings

	1	2	3	4	≥ 5
FT-HQ	47	87	57	29	9
FT-CR	43	97	55	25	23
FT-XL	44	92	64	22	20
STOCK-1	49	88	60	24	20
STOCK-2	46	85	63	26	13

Table A.8.: Amount of different ratings per number of keywords

A.14. Standard Deviations of Ratings Compared to the History Length

Model	Position 2	Position 4	Position 6
FT-HQ	0.7723	0.6152	0.6643
FT-CR	1.0329	0.8668	1.0476
FT-XL	0.8430	1.1153	1.0946
STOCK-1	1.1869	1.2470	1.1742
STOCK-2	1.1793	0.8363	1.0688

Table A.9.: Standard deviations of human ratings in comparison to the history length

A.15. Scenario and Rating

A.15.1. Standard Deviations

	Open	Defined
FT-HQ	0.7271	0.6427
FT-CR	0.9483	1.73
FT-XL	1.0307	1.0223
STOCK-1	1.1791	1.2149
STOCK-2	1.0402	1.33

Table A.10.: Standard deviations for ratings depending on the scenario type

A.15.2. Detailed Scenarios

Model	1	2	3	4	5	6	7	8
FT-HQ	4.72	4.6207	4.75	4.6757	4.48	4.6667	4.7083	4.4359
FT-CR	4.6071	4.4483	4.50	4.3659	4.5517	4.2759	4.0645	4.3438
FT-XL	4.4062	4.1379	4.2424	4.3667	4.3333	4.4828	4.2069	4.2424
STOCK-1	4.0278	4.2759	3.9688	4.0769	4.2222	4.0556	4.1724	3.9677
STOCK-2	4.25	4.5946	4.2143	4.30	4.3913	4.3636	4.50	4.25

Table A.11.: Average ratings for the models depending on the exact scenario

Model	1	2	3	4	5	6	7	8
FT-HQ	0.5416	0.7277	0.5316	0.6689	1.50	0.5547	0.6241	0.7180
FT-CR	0.6289	0.8696	0.8341	1.1781	0.9851	0.8408	1.1236	1.0957
FT-XL	0.9791	1.0930	1.0906	0.8899	1.00	0.9495	1.1458	1.0906
STOCK-1	1.2532	1.0986	1.1496	1.2223	1.1547	1.3921	1.3112	1.1397
STOCK-2	1.0195	0.8320	1.17	1.1788	0.9881	1.0553	1.86	1.0473

Table A.12.: Standard deviations for the models depending on the exact scenario

A.16. Automatic Evaluation of Perfect Sentences

	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
BERT	0.7225	0.7225	0.7225	0.7225	0.7225
FUZZ	0.6012	0.6012	0.6012	0.6012	0.6012
JARO	0.6792	0.6792	0.6792	0.6792	0.6792
BLEU	0.3136	0.3136	0.3136	0.3136	0.3136
METEOR	0.3276	0.3276	0.3276	0.3276	0.3276
ROUGE	0.4720	0.4720	0.4720	0.4720	0.4720
SPACY	0.7843	0.7843	0.7843	0.7843	0.7843
Human (Adj.)	5	5	5	5	5
Human	5	5	5	5	5

Table A.13.: Averages of automatic evaluation of only perfect human ratings

A. Appendix

	FT-HQ	FT-CR	FT-XL	STOCK-1	STOCK-2
BERT	0.1948	0.1948	0.1948	0.1948	0.1948
FUZZ	0.1882	0.1882	0.1882	0.1882	0.1882
JARO	0.0954	0.0954	0.0954	0.0954	0.0954
BLEU	0.2383	0.2383	0.2383	0.2383	0.2383
METEOR	0.2455	0.2455	0.2455	0.2455	0.2455
ROUGE	0.2465	0.2465	0.2465	0.2465	0.2465
SPACY	0.16	0.16	0.16	0.16	0.16
Human (Adj.)	0	0	0	0	0
Human	0	0	0	0	0

Table A.14.: Standard deviations of automatic evaluation of only perfect ratings

A.17. Standard Deviations of Keyword Extraction Tool Evaluations

Metric	KeyBERT	YAKE	YAKE (De-tailed)	Human
Generated KWs in human KWs (String)	0.3202	0.3637	0.3572	0
Human KWs in generated KWs (String)	0.3235	0.3537	0.3692	0
Generated KWs in human KWs (Topic)	0.2926	0.3544	0.3651	0.0427
Human KWs in generated KWs (Topic)	0.2673	0.3361	0.3352	0.0427
BERT	0.2119	0.2336	0.2452	0
FUZZ	0.2328	0.2515	0.2886	0
JARO	0.2380	0.2664	0.2867	0
BLEU	0.3038	0.34	0.3159	0
METEOR	0.3161	0.2587	0.2941	0.1849
ROUGE	0.2831	0.3249	0.3149	0
SPACY	0.2615	0.3213	0.3034	0
Average	0.14	0.1173	0.1176	0.0378

Table A.15.: Standard deviations of means of keyword tools evaluation

A.18. Hyperparameters

A.18.1. Epoch Size

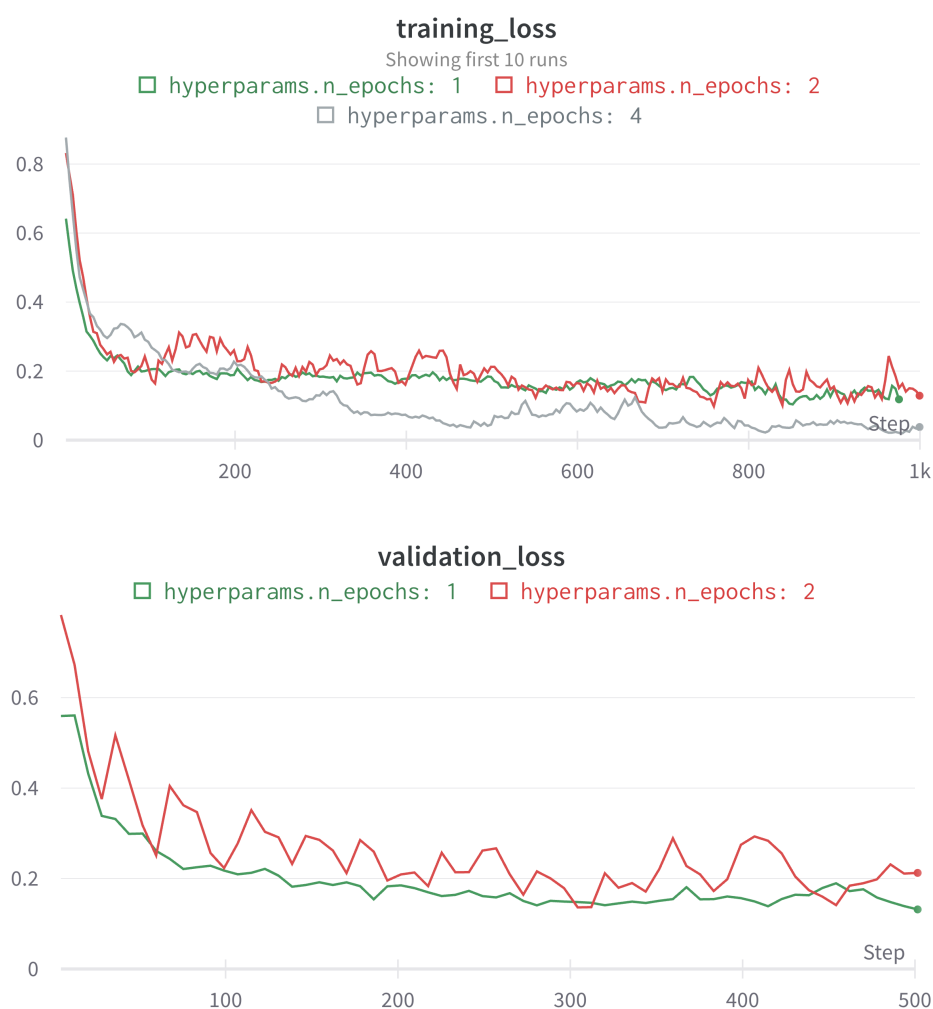


Figure A.7.: W&B evaluations of the epoch size and losses in training

A. Appendix

A.18.2. Batch Size

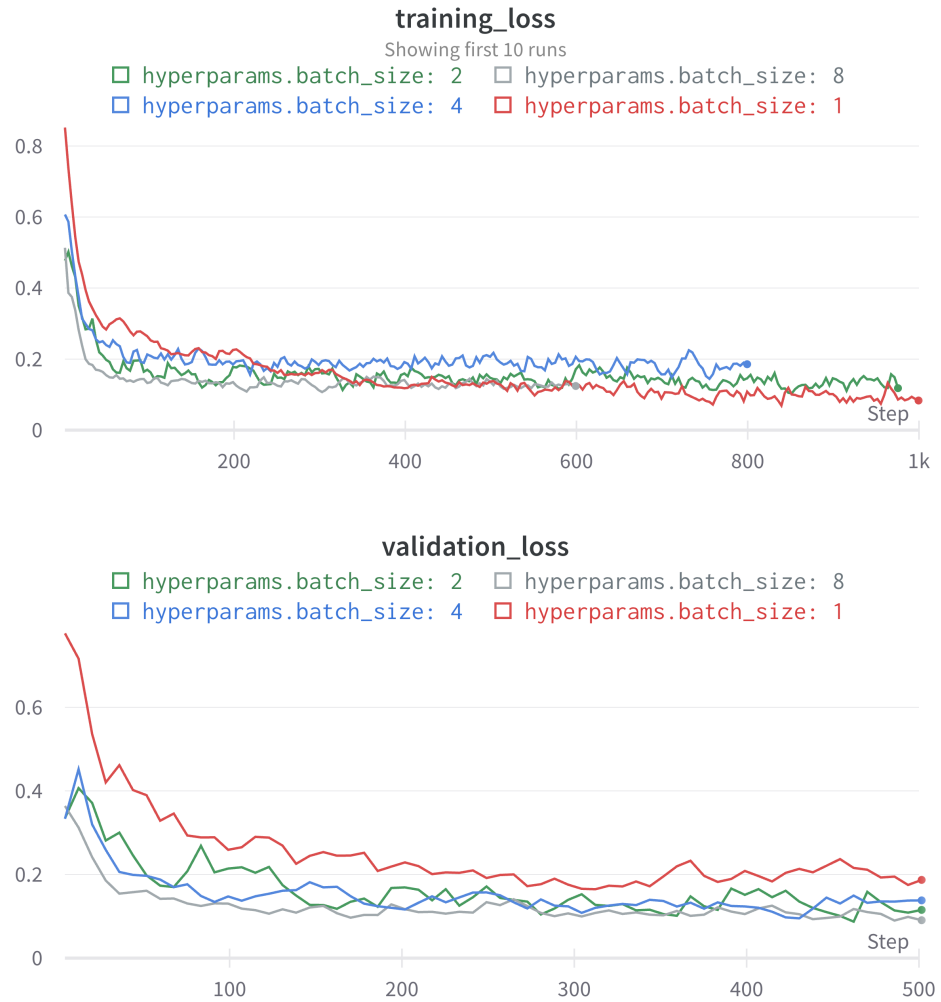


Figure A.8.: W&B evaluations of the batch size and losses in training

A.18.3. Learning Rate Multiplier

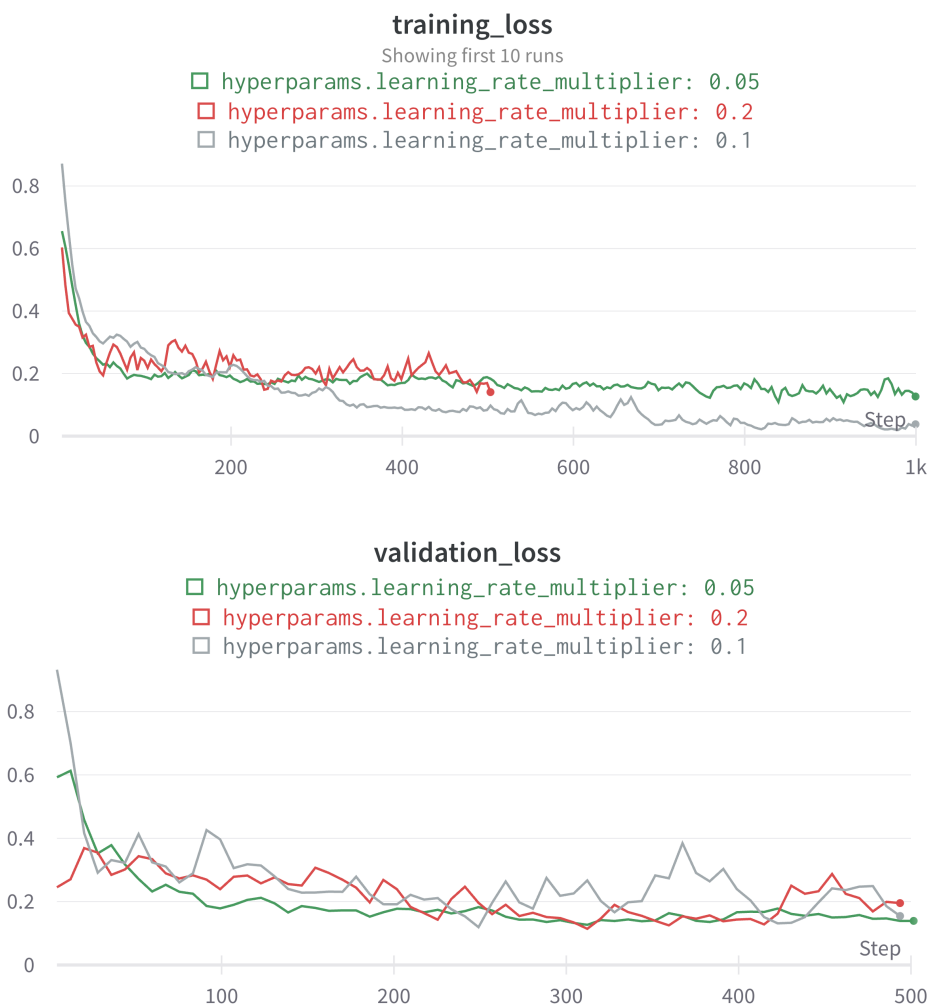


Figure A.9.: W&B evaluations of the learning rate multiplier and losses in training

A.19. Codebase

In the following the individual codes that were written within the scope of the thesis can be found. As already noted in the chapters, the codes for the evaluation and creation of the data sets are also included. In contrast to the very general and straightforward structured code for the experiment, the Jupyter notebooks, especially those for evaluating the code, are strongly and specifically adapted to the data structure of the logs. A change in the data structure must result in a corresponding change in the notebooks.

A.19.1. Experiment

The experiment itself, as well as the code for it, can be accessed via the following link: <https://huggingface.co/spaces/DialoKEY/Experiment>. By clicking on “Files” in the top right-hand corner, the corresponding repository can be displayed and cloned. In order to run the experiment locally, several environment variables have to be added. These are stored in the file *environment_variables.txt* under the following link. The experiment was run locally in a Python 3.9 environment.

- Link: <https://bit.ly/DialoKEY-Keys>
- Password: univie

The following things should be mentioned in addition to the commented code:

- If the Gradio app is password protected, it cannot be opened via the normal Hugging Face app view, but can be opened via the direct link: <https://dialokey-experiment.hf.space/>
- The password for the app is stored in the environmental variables (“PASSWORD”) and can be retrieved from the above-mentioned file.
- A potential source of error is the `openai.error.RateLimitError`. Unfortunately, OpenAI lacks transparency about the exact number of requests and limits when this error occurs. Make sure that the API key is always up-to-date and that the corresponding limits in the account are set high enough. During the course of the experiment, for example, the error did not occur (although there were an enormous number of requests to the api), but in the testing before and after there were repeated cases in which the error occurred. Clicking on “Reload Chat” allows you to continue with the experiment as normal if the error has occurred.
- As commented in the code, new survey data are no longer stored in the database collection *logs_4* but in *logs_after_thesis*. However, this can be easily adjusted again in the code.
- All data can be exported, and read as in the file *code_cleaning.ipynb* indicated. The raw data of all answers and scores can be found in database *logs_04*. After the data has been cleaned as described in section 6.9, it is stored in database *logs_cleaned_3*.

- If the experiment is accessed via the Hugging Face server, the server may have been stopped. It pauses after 72 hours of inactivity. It is sufficient to access the page again, then the server will be restarted. It may be necessary to refresh the page, though.

A.19.2. Fine-Tuning and Evaluation

The repository of codes for fine-tuning and evaluation can be found under https://github.com/Lucas-Munz/DialoKEY_Codebase. All the codes listed for reproducibility can be found here, as shown in Fig. A.10.

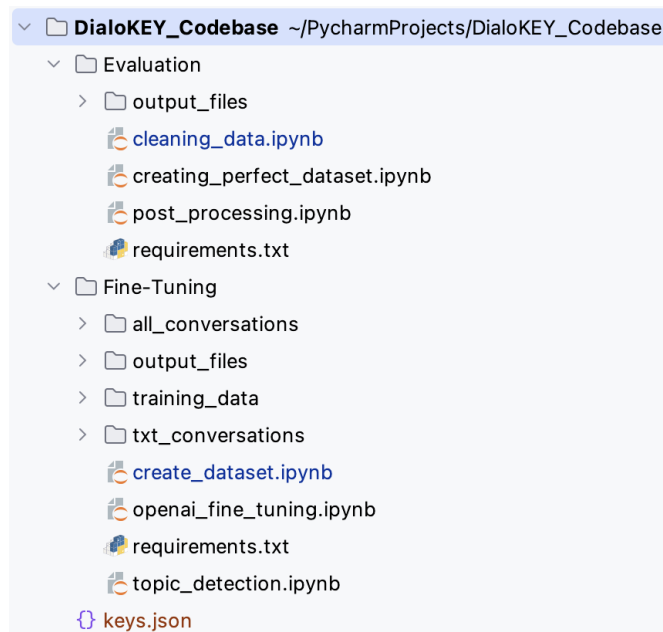


Figure A.10.: Folder structure of codebase

As listed in the corresponding places, paths, and API keys are required, which are not found in the repository for security reasons.

For this purpose, the following file named keys.json must be downloaded:

- Link: <https://bit.ly/DialoKEY-Keys>
- Password: univie

The file should be placed in the Top-Level directory, as shown in Fig. A.10. Afterwards, all Jupyter notebooks can access it, so that no further adaptation is necessary. The contained notebooks were run locally in a Python 3.10 environment.

A.19.3. Training data for the fine-tuned models

The training data used to fine-tune the models, as stated in chapter 5, can be retrieved from the folder *Fine-Tuning/training_data*. Here, all three corresponding prepared training files are located. The folder *Fine-Tuning/all_conversations* contains all raw conversation files used in the training.

A.19.4. Topic Detection

The code for the topic detection is located in the *topic_detection.ipynb* file. In order to run this file, an API Key, as instructed in A.19.2 is necessary. The code then analyses all topics of the conversations in the *Fine-Tuning/txt_conversations* directory. Anything else is documented and commented in the code itself.

A.19.5. Creating a “Perfect” Dataset

This section is dedicated to the creation of a new model, which is not part of the thesis, but can be used for future work.

In order to use the learnings from the evaluation for a new data set, I have created a code to accomplish just that. In doing so, I will use the data generated in the experiment. More specifically, I use generated answers that were rated *Perfect* by the users. It does not matter which model the generated answer comes from. Since the conversation history and the keywords of the participants are also available, it is possible to generate a data set as in section 5.2.

When generating this new data set, I decided to use the generated answers instead of the answers of the experiment participants. This is mainly because the spelling of the latter is often not flawless. Furthermore, this selection only plays a marginal role, since the users rated the answer *Perfect*.

The creation of the dataset can be reproduced via the file *creating_perfect_dataset.ipynb* in the repository. With this resulting dataset saved in *Evaluation/output_files*, a new model can be fine-tuned and tested.