

Leobot - Chatbot für die HTL Leonding

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

**Höhere Lehranstalt für Informationstechnologie,
Ausbildungsschwerpunkt Medientechnik**

Eingereicht von:

Felix Dumfarth
Lukas Starka

Betreuer:

Thomas Stütz

Leonding, April 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

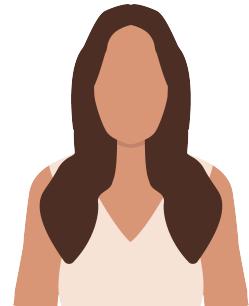
Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022

Felix Dumfarth & Lukas Starka

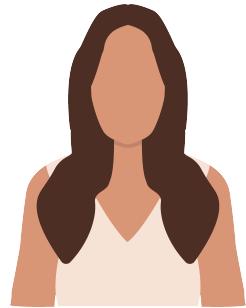
Abstract

This diploma thesis deals with the topic of a chatbot that was created with the help of Rasa and can be found on the HTL Leonding website. This chatbot should be able to provide answers about school-specific topics quickly and easily. The bot's topics range from school-specific questions to small talk and general information about the work. The users have the opportunity to rate the given answers to their questions and to give feedback. In addition, a so-called dashboard was developed, in which an overview of all the conversations with the chatbot can be seen. With a built-in editor, the knowledge of the bot can be constantly expanded.



Zusammenfassung

Die vorliegende Diplomarbeit beschäftigt sich mit dem Thema eines Chatbots, der mithilfe von Rasa erstellt worden ist und auf der Website der HTL Leonding zu finden sein soll. Dieser Chatbot soll interessierten Personen einfach und schnell Antworten über schulspezifische Themen liefern können. Dabei reichen die Themengebiete des Bots von schulspezifischen Fragen bis hin zu Small Talk und generellen Informationen über die Arbeit. Den Benutzerinnen und Benutzern steht dabei die Möglichkeit zu, die Beantwortung ihrer Fragen zu bewerten und Feedback zu geben. Zusätzlich wurde noch ein sogenanntes Dashboard entwickelt, bei dem eine grafische Aufbereitung aller Unterhaltungen mit dem Chatbot zu sehen ist. Mit einem eingebauten Editor kann dabei das Wissen des Bots ständig erweitert werden.



Contents

1 Einleitung	1
1.1 Struktur der Diplomarbeit	1
1.2 Ausgangslage	1
1.3 Istzustand	2
1.4 Problemstellung	2
1.5 Aufgabenstellung	2
1.6 Use-Cases	2
2 Technischer Hintergrund	4
2.1 Text Analysis	4
2.2 Natural Language Processing (NLP)	5
3 Toolstack	19
3.1 Programmiersprachen	19
3.2 Technologien	20
3.3 Werkzeuge	22
4 Chatbots am Beispiel von Rasa	26
4.1 Allgemeines	26
4.2 Pipeline	28
4.3 Welche Rolle spielen neuronale Netze in Rasa	43
4.4 Komponenten	44
4.5 Initialisieren	51
4.6 Trainieren	52
4.7 Interagieren	53
5 Implementierung	55
5.1 Systemarchitektur	55
5.2 Backend	55

5.3 Chat Widget	57
5.4 Dashboard	62
5.5 Einbindung in Wordpress	67
5.6 Deployment	69
6 Evaluation	75
6.1 Schwierigkeiten in der Umsetzung	75
6.2 Fazit	75
Bibliography	VI
List of Figures	XII
List of Tables	XIV
Quellcodeverzeichnis	XV
Anhang	XVII

1 Einleitung

1.1 Struktur der Diplomarbeit

Die folgenden Themengebiete wurden in der Diplomarbeit abgewickelt:

Technischer Hintergrund: Im technischen Hintergrund geht es um notwendiges Vorabwissen über maschinelles Lernen und damit verbundene Themengebiete wie Deep Learning und neuronale Netze, deren Verständnis relevant für die darauffolgenden Themengebiete ist.

Toolstack: Der Toolstack präsentiert die verwendeten Tools und Technologien, die zur Erstellung der Diplomarbeit genutzt wurden.

Chatbots am Beispiel Rasa: Dieses Kapitel beschreibt die Komponenten und die Entwicklung eines Chatbots mit Rasa im Detail.

Implementierung: Dieses Kapitel handelt von der Entwicklung und der Umsetzung des Chatbots und seinen Begleitprodukten aus dem praktischen Teil der Diplomarbeit.

Evaluation: Die Evaluation gibt einen Überblick über die Ergebnisse der Arbeit und, welche Ziele erfüllt wurden.

1.2 Ausgangslage

Zahlreiche Organisationen bauen heutzutage auf eine starke Webpräsenz und wickeln dabei komplexe Produkte oder Leistungsfunktionalitäten ab. Eine wichtige Rolle spielt dabei nicht nur der Kundenservice, sondern auch die Kommunikation, denn die Nutzer erwarten eine möglichst zeitnahe Beratung. Die Bereitstellung von solchen Services ist derzeit immer noch mit sehr hohen Kosten verbunden.

1.3 Istzustand

Es gibt viele Personen, die an dem Ausbildungsangebot der HTL Leonding interessiert sind und sich gerne schnell über die Schule informieren wollen.

1.4 Problemstellung

Die HTL Leonding hat ein großes Informationsangebot, zu diesem gibt es viele Zugänge. Dabei ist es zum Beispiel auf der Website oft schwierig den Überblick zu bewahren mit den vielen Unterseiten und daher wäre eine Unterstützung zum Finden für die vielen Informationen hilfreich.

1.5 Aufgabenstellung

In dieser vorliegenden Arbeit soll ein Chatbot für die HTL Leonding Website erstellt werden, der Besucherinnen und Besuchern einfach und schnell schulspezifische Fragen beantworten soll. Jedoch soll es auch möglich sein, dass der Bot auch als Grundlage für weitere Arbeiten in der Schule, wie zum Beispiel der 3-D-Leonie, zum Einsatz kommen kann.

1.6 Use-Cases

Der User kann hauptsächlich schulspezifische Unterhaltungen mit dem Chatbot führen und diese Unterhaltung anhand ihrer Qualität bewerten. Aber auch generelle Informationen zum Bot, zu Rasa und Small Talk können bearbeitet werden. Eine verantwortliche Person kann sich die Unterhaltungen und Bewertungen von Usern ansehen, sodass diese beurteilt werden können. Außerdem kann die verantwortliche Person die Rasa Files `nlu.yml`, `stories.yml`, `rules.yml` und `domain.yml` bearbeiten und den Chatbot auffordern, sein neu erlerntes Wissen einzutrainieren.

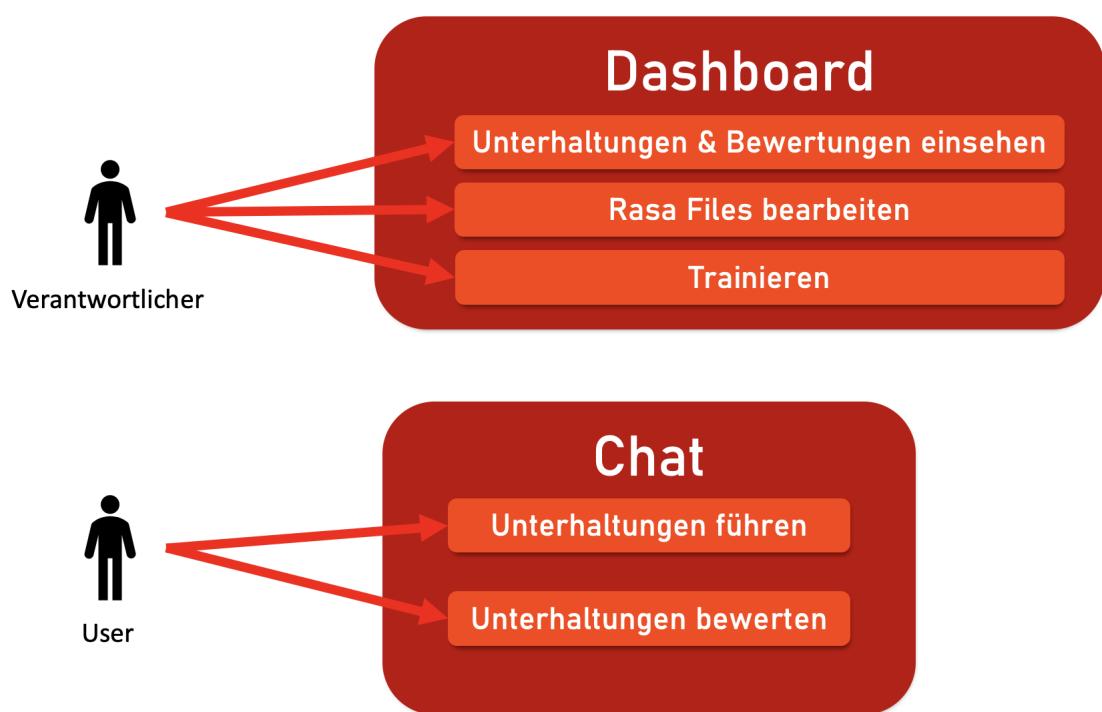


Figure 1: Use-Case Diagramm

2 Technischer Hintergrund

2.1 Text Analysis

2.1.1 Text analysis vs. Text Mining vs. Text Analytics

Meistens werden die Begriffe Text Analysis und Text Mining im selben Zusammenhang verwendet. Dabei bedeuten sie eigentlich dasselbe, nämlich, dass man den Sinn einer Nachricht extrahiert. Deswegen wird in den folgenden Kapiteln nur von Text Analysis gesprochen.

Es gibt jedoch einen Unterschied zwischen Text Analysis und Text Analytics. Grundsätzlich ist dabei zu sagen, dass Text Analysis qualitative Ergebnisse liefert, wohingegen bei Text Analytics die Quantität mehr im Vordergrund steht.[1, 2]

Bei Text Analysis werden also wichtige Informationen aus der Nachricht herausgelesen. Oder anders formuliert geht es darum, dass trotz der Unterschiede der menschlichen Sprache trotzdem die Kernaussage herausgefunden werden kann, mit der dann gearbeitet wird. Es können also dann Informationen herausgefiltert werden, wie beispielsweise, ob etwas positiv oder negativ ist oder was das Hauptthema des Textes ist.[1, 2]

Auf der anderen Seite werden bei Text Analytics verschiedene Muster aus einer großen Menge an Nachrichten herausgefiltert, welche dann in Graphen, Tabellen oder Berichten gezeigt werden können. Bei Text Analytics geht es also darum, Muster und Entwicklungen von numerischen Ergebnissen herauszufinden, bei denen dann Informationen herausgefiltert werden können, wie beispielsweise die Prozentzahl der positiven Bewertungen.[1, 2]

2.1.2 Warum Text Analysis?

Im Gegensatz zum manuellen Aufbereiten von Texten kann man durch Machine Learning Texte rund um die Uhr und zur Echtzeit bearbeiten. Zusätzlich werden durch die Verwendung von Text Analysis Arbeitskräfte, die für das Aufbereiten der Texte und deren Bedeutung verantwortlich wären, eingespart. Durch Algorithmen werden

außerdem Fehler reduziert, die bei manuellem Bearbeiten leicht auftreten können und Daten können genauer aufbereitet werden.[1]

2.1.3 Machine Learning mit Text Analysis

Ein Text Analysis Tool besteht grundsätzlich aus drei verschiedenen Schritten.

1. Zunächst wird überlegt, welche Daten gesammelt werden sollen, um das Modell zu trainieren und zu testen. Hierbei wird zwischen **Internal Data** und **External Data** unterschieden. Unter External Data werden Quellen wie Zeitungen oder Foren bezeichnet und zu der Internal Data zählen sämtliche Daten, die eine Firma jeden Tag generiert, wie E-Mails, Reports, Chats oder Umfragen.
2. Danach müssen die Daten vorbereitet werden, damit das Programm diese versteht. Dieser Schritt wird meistens als **Data Preprocessing** bezeichnet.
3. Zum Schluss wird dann ein Machine Learning Algorithmus hinzugefügt, welcher sich um die Analyse kümmert. Diesen kann man entweder komplett selber implementieren oder man nimmt sich Libraries zur Hilfe.[2]

2.2 Natural Language Processing (NLP)

Natural Language Processing, oft auch als Akronym **NLP** abgekürzt, sorgt dafür, dass Computer Text auf dieselbe Art und Weise verstehen, wie wir es als Menschen tun. NLP vereint dabei die Modellierung der menschlichen Sprache mit statistischen Machine Learning und Deep Learning Modellen. Dies ermöglicht der Maschine, die menschliche Sprache in Form von Text- oder Sprachdaten zu verarbeiten und sozusagen die komplette Bedeutung und Absicht der Benutzerin oder des Benutzers zu verstehen.[3]

2.2.1 Corpus

Unter dem Corpus werden die Daten bezeichnet, die verwendet werden, um das NLP Modell zu trainieren, damit dieses Modell menschliche Sprache versteht und damit arbeiten kann. Der Textkorpus ist also eine Menge von strukturierten Texten, die für den Computer lesbar sind.[4]

2.2.2 Tokenization

Bei der Tokenization repräsentiert jeder Token eine sinnvolle Einheit. Darunter sind Wörter, Zeichen oder Sonderzeichen gemeint, die alle einen eigenen Token darstellen, lediglich Leerzeichen in einem Satz stellen keine eigenen Token dar.[2, 5]

Die verschiedenen Token, die dabei entstehen, können dabei wie in Listing 1 aussehen:

Listing 1: Beispiel für die Tokenization

```

1 Let us go to the park.
2
3 0: Let
4 1: us
5 2: go
6 3: to
7 4: the
8 5: park
9 6: .

```

Die Tokenization ist dabei besonders für verschiedene Sprachen von Vorteil, da es in diesen Sprachen immer andere Regeln gibt, wie die Wörter aufgeteilt werden können. In dem Beispiel 1 wirkt es womöglich so, als könnte man mit einer `split()` Methode dasselbe Resultat erzielen, aber es gibt auch Besonderheiten einer Sprache, die vom Tokenizer bedacht werden müssen, wie Beispiel 2 aufzeigt.[2, 5]

Listing 2: Ausnahme für bestimmte Tokens

```

1 "Let's go to the U.K.!"
2
3 0: "
4 1: Let
5 2: 's
6 3: go
7 4: to
8 5: the
9 6: U.K.
10 7: !
11 8: "

```

2.2.3 Part-of-speech-Tagging (POS-Tagging)

Beim Part-of-speech-Tagging geht es um das Zuordnen von Wörtern mit deren zugehörigem Teil der Sprache. Als Teil einer Sprache, also dem Part-of-speech, werden grundsätzlich Wörter bezeichnet, die ähnliche grammatischen Eigenschaften oder Nutzungen besitzen. Im deutschen sind hierbei also die verschiedenen Wortarten gemeint.

Beim Tagging werden Statistiken angewendet, wie beispielsweise, dass es sehr wahrscheinlich ist, dass nach einem Artikel ein Nomen folgt. Außerdem gibt es sogenannte Rule-Based POS-Taggers, deren Aufgabe es ist, vordefinierte Regeln zu verwenden, um das Tagging zu vollziehen.[2, 5]

Ein Beispiel für POS-Tagging könnte also wie in Listing 3 aussehen:

Listing 3: Beispiel für POS-Tagging

```

1 I am going to the U.K.
2
3 I: Pronoun
4 am: Verb
5 going: Verb
6 to: Part
7 the: Article
8 U.K.: Noun

```

2.2.4 Named-entity recognition (NER)

Named-entity recognition ist eine der wichtigsten Säulen von Natural Language Processing 2.2. Entities sind strukturierte Stücke von Informationen, die sich innerhalb der Nachricht einer Benutzerin oder eines Benutzers befinden. Dabei werden Entities aus Texten erkannt und anschließend mit einem Tag der zugehörigen Kategorie versehen.[6]

Beispiele für Entities wären also:

Listing 4: Beispiele für Entitites

```

1 PERSON      Personen (inklusive fiktionalen Personen)
2 NORG        Nationalit ten oder religi se oder politische Gruppen
3 FACILITY    Gebaeude, Flughafen, Bruecken, Strassen
4 LOC          Orte
5 PRODUCT      Objekte, Fahrzeuge, Essen
6 LANGUAGE     Sprachen

```

2.2.5 Stemming

Beim Preprocessing von Texten sind außerdem das Stemming und die Lemmatization 6 Schritte, die häufig durchgeführt werden.

Beim Stemming wird der Wortstamm eines Tokens von den Präfixen und Suffixen gelöst. Unter einem Präfix versteht man eine Vorsilbe, die dem Wortstamm vorangestellt wird und bei dem Suffix handelt es sich im Gegenzug dazu um Worterweiterungen, die dem Stamm folgen. Hierbei gibt es wieder die Möglichkeit, selber einen Stemmer zu implementieren oder bereits vorgefertigte Stemmer zu verwenden.[1, 2]

Ein Beispiel für Stemming von deutschen Wörtern wäre also in Listing 5 zu sehen.

Listing 5: Stemming von deutschen Wörtern

```

1
2 verarbeiten ---> ver + arbeit + en
3 ver                      prefix
4 arbeit                   base word
5 en                       suffix

```

2.2.6 Lemmatization

Unter der Lemmatization oder auch Lemmatisierung wird der Prozess verstanden, bei dem alle verschiedenen Arten von einem Wort zu ihrem Wortstamm umgeformt werden.

Die Lemmatisierung ist sozusagen eine Weiterentwicklung des Stemming 2.2.5 und versucht im Gegensatz zum Stemming den Kontext zu den Wörtern zu bringen. Je nachdem in welchem Kontext die Verben gebraucht werden, können sie sich dabei von ihrer Wortart unterscheiden und so eine andere Bedeutung haben. Generell lässt sich sagen, dass Stemmer meist einfacher zu implementieren sind und für schnellere Laufzeiten sorgen, allerdings wird von vielen auch die Lemmatization dem Stemming vorgezogen, aufgrund der Vorteile, die das Einbeziehen des Kontextes mit sich bringt.[2, 1, 7]

Beispiele für die Lemmatisierung sind:

Listing 6: Lemmatisierung von deutschen Wörtern

```

1 Gut ---> gut
2 besser ---> gut
3 las ---> lesen
4 gelesen ---> lesen
5 lesend ---> lesen
6 mag ---> moegen

```

2.2.7 Parsing

Parsing ist eine Art, um einen Satz auseinanderzuteilen, um die Struktur des Satzes besser zu verstehen. Beim Parsing wird also die syntaktische Struktur eines Textes bestimmt. Um dies zu erreichen, nimmt der Parsing Algorithmus die Eigenheiten der Grammatik von der jeweiligen Sprache, in der der Text geschrieben ist, her.[1]

Es gibt zwei verschiedene Arten von Parsing. Diese sind das sogenannte **Dependency Parsing** 2.2.7 und das sogenannte **Constituency Parsing** 2.2.7.

Dependency Parsing

Beim Dependency Parsing wird die grammatische Struktur eines Satzes bestimmt. Außerdem kann mithilfe des Dependency Parsing herausgefunden werden, welche Wörter zusammenhängen und welche Beziehung diese zueinander aufweisen. Dies bedeutet beispielsweise, dass man bei den Wörtern **blauer Ball**, **nette Frau** oder Ähnlichem erkennen kann, dass sich die Wörter **blau** oder **nett** auf das nachfolgende Wort bezieht. Ein anderes Beispiel könnten die Wörter **black** und **car** sein.

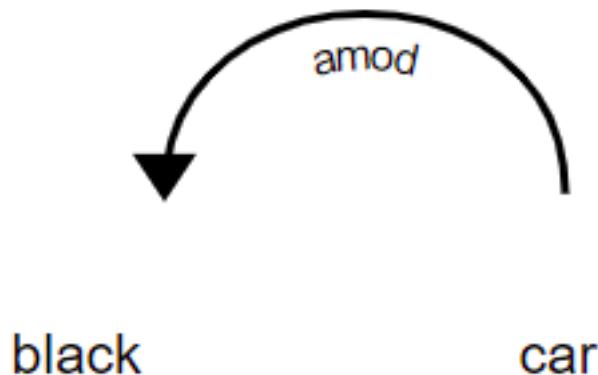


Figure 2: Beispiel für die Beziehung zweier Wörter beim Dependency Parsing [8]

Wie aus der Grafik 2 entnommen werden kann, bezieht sich die Farbe Schwarz in diesem Fall auf das Auto. Im Englischen wird das Wort **car** deswegen auch als **Head** bezeichnet und **black** ist ein **Child**, also ein Wort, dass abhängig von diesem **Head** ist. Die Beziehung, die zwischen diesen Wörtern vorliegt, wird mit **amod** bezeichnet. Dies ist eine Abkürzung für **Adjectival Modifier**, also ein Adjektiv, welches ein Nomen modifiziert und zu der Bedeutung des Nomens beiträgt. In diesem Fall gibt dieses eine genaue Beschreibung der Farbe des Autos.[8]

Diese Beziehungen kann man auch in einem Baumdiagramm, wie in 3 darstellen.

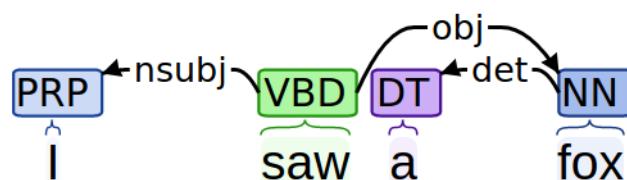


Figure 3: Darstellung des Dependency Parsing [9]

Constituency Parsing

Beim Constituency Parsing wird ein Satz in mehrere Teilphrasen zerteilt, die auch **Constituents** genannt werden. Im Gegensatz zum Dependency Parsing, bei dem Beziehungen zwischen den Wörtern innerhalb eines Satzes aufgezeigt werden, werden hierbei also die Sätze gruppieren. Dies kann helfen, um die Struktur von Sätzen aufzuzeigen. Der Nachteil hierbei ist, dass man nun keinen Context in den Sätzen mehr

hat, dafür kann man aber beispielsweise gut überprüfen, ob ein Satz grammatikalisch korrekt oder inkorrekt ist.[2]

Die berühmtesten Tags, um Constituents zu bezeichnen sind:

- VP für Verbalphrasen (Verb Phrases)
- NP für Nominalphrasen (Noun Phrases)

Der Satz **I saw a fox** wird hierbei als Tree des Constituency Parses dargestellt.

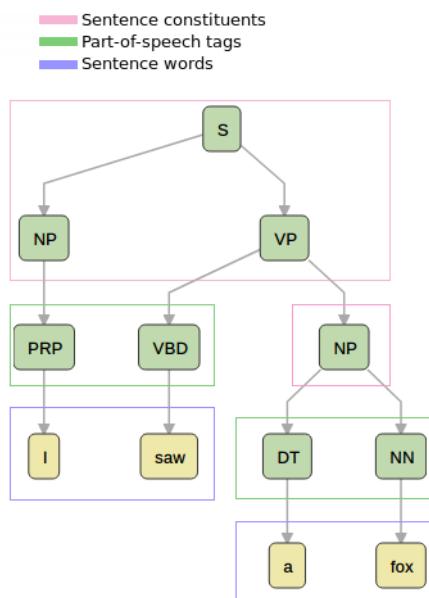


Figure 4: Darstellung des Constituency Parsing [9]

In dem Beispiel 4 ist zu sehen, dass der Satz in zwei Teile aufgeteilt wurde. Der Textbaustein **I** wurde als Substantiv (NP) erkannt und **saw a fox** ist hierbei die Phrase des Verbs (VP). Darunter sind die Regeln angeführt, die dazu führten, dass hierbei erkannt wurde, dass es sich um eine Nominalphrase und eine Verbalphrase handelt. Diese Regeln werden durch die Grammatik der Sprache erstellt und so sagt beispielsweise die Regel **S → NPVP** aus, dass ein ganzer Satz (S) durch eine Nominalphrase (NP) und eine Verbalphrase (VP) entsteht.[9]

2.2.8 Stopwords

Unter Stopwords oder Stoppwörtern werden Wörter verstanden, die häufig in einem Satz vorkommen, aber nicht wirklich zum Informationsgehalt des Textes beitragen. Deswegen werden diese oft nicht beachtet, da sie keine Relevanz für die Bedeutung des Satzes haben.

Bei den Stopwörtern handelt es sich meistens um die am häufigsten vorkommenden Wörter einer Sprache. Deutsche Stopwörter sind zum Beispiel:[10]

- was
- aber
- auch
- der
- die
- das
- du
- wir
- ihr
- können
- müssen
- dürfen

2.2.9 Vectorization

Durch Machine Learning können Systeme basierend auf vorherigen Eingaben und Mustern Dinge voraussagen. Diese Systeme benötigen allerdings Trainingsdaten, um ihre Präzision dabei zu steigern. Wenn nun also ein System trainiert werden soll, wird eine Darstellungsform, die auch von Maschinen verstanden werden kann, benötigt. Hierbei kommen nun Vektoren zum Einsatz, weil das System dadurch die wichtigsten Informationen extrahieren und daraus lernen kann.

2.2.10 Skalarprodukt

Beim Natural Language Processing 2.2 wird sehr oft das sogenannte Skalarprodukt benötigt. Dieses kann auch als inneres Produkt oder Punktprodukt bezeichnet werden. Das Skalarprodukt ordnet zwei Vektoren eine Zahl zu. Ein Skalarprodukt wird wie in 5 berechnet.

$$\vec{a} \bullet \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \bullet \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

Figure 5: Skalarprodukt von zwei Vektoren [11]

2.2.11 Bag of Words

Eine der einfachsten Techniken, um einen Text nummerisch zu repräsentieren, ist mithilfe der sogenannten Bag of Words Vectorization. Dabei wird zunächst jedes Wort, welches im Corpus vorkommt, als sogenanntes Vokabular gespeichert. Danach wird in jedem Satz die Anzahl dargestellt, wie oft ein bestimmtes Wort vorkommt.[1]

Listing 7: Bag of Words

```

1 Dies ist ein Bag of Words Beispiel.
2
3 Das      0
4 Dies     1
5 ist      1
6 ein      1
7 eine     0
8 Bag      1
9 of       1
10 Words   1
11 Beispiel 1
12 Example  0

```

Um die Überlappungen von Bag of Words zu berechnen, wird das Skalarprodukt der beiden Vektoren von Sätzen, die verglichen werden sollen, genutzt. Bei diesem Skalarprodukt kommt dann immer ein Wert (Skalar) als Ergebnis heraus.

2.2.12 Bag of n-grams

Ein Bag of n-grams ist ähnlich zu einem Bag of Words 2.2.11. Allerdings wird bei einem Bag of n-grams ein Text als eine unsortierte Sammlung von n-grams dargestellt. In dem Bag of n-grams wird dann die Anzahl, wie oft ein n-gram vorkommt, gespeichert. Ein n-gram ist eine Sammlung von n aufeinander folgender Wörter. Wenn das n = 2 ist, wird von einem bigram gesprochen, wenn n = 3 ist von einem trigram und so weiter. Somit würde das Beispiel von 7 als bigram folgendermaßen aussehen:

Listing 8: Bag of n-grams

```

1 Dies ist ein Bag of Words Beispiel.
2
3 1: <start> Dies
4 2: Dies ist

```

```

5 3: ist ein
6 4: ein Bag
7 5: Bag of
8 6: of Words
9 7: Words Beispiel
10 8: Beispiel <end>

```

2.2.13 Term Frequency times Inverse Document Frequency (TF-IDF)

TF-IDF steht für Term Frequency times Inverse Document Frequency und dies ist die meist verwendete Technik, um die Vorkommen von Wörtern innerhalb eines Satzes zu zählen. Mithilfe von TF-IDF kann ermittelt werden, wie wichtig ein Wort in einem Satz ist. Bei TF-IDF werden zwei verschiedene Metriken miteinander multipliziert.[12]

- Term Frequency TF:** Die Term Frequency wird ausgerechnet, indem die Anzahl eines Terms, der in einem Dokument vorkommt, durch die Anzahl der Terme insgesamt gerechnet wird.
- Inverse Document Frequency IDF:** Diese gibt an, wie selten ein Wort im ganzen Dokument ist. Dabei wird die Anzahl der Dokumente (N) durch die Anzahl des Terms (n) gerechnet und davon wird der Logarithmus berechnet.

$$TF(t, d) = \frac{\text{number of times } t \text{ appears in } d}{\text{total number of terms in } d}$$

$$IDF(t) = \log \frac{N}{1 + df}$$

$$TF - IDF(t, d) = TF(t, d) * IDF(t)$$

Figure 6: Berechnung von TF-IDF [13]

Wenn diese zwei Werte nun also miteinander multipliziert werden, erhält man so einen Wert, der indiziert, ob ein Wort relevant für die Bedeutung in dem Dokument ist. Je höher der Wert ist, desto relevanter ist das Wort sozusagen.

Listing 9: TF-IDF Beispiel

```

1 Heute ist ein warmer Tag
2
3 Term      Count
4 Heute     2
5 ist       1
6 ein       1
7 warmer    2
8 Tag       5

```

Bei dem Beispiel 9 wird gezeigt, dass die Wörter, die häufig in einem deutschen Satz vorkommen, wie **ist** oder **ein** einen niedrigen Score erhalten und das Wort **Tag** einen

hohen Score aufweist. Dies bedeutet, dass dieses Wort vermutlich wichtig für die Bedeutung des Satzes ist.

Bei der Text Analysis mit Machine Learning werden solche TF-IDF Algorithmen verwendet, um Daten in Kategorien einzuordnen und wichtige Schlüsselwörter zu extrahieren. Auch Suchmaschinen wie Google verwenden diese Technologien, um aus dem Suchbegriff des Users die zentrale Aussage herauszufinden und die Suchergebnisse nach ihrer Relevanz zu sortieren.[12]

Word Vectors

Zurzeit wurde der Kontext von Wörtern in einem Satz neben dem zu analysierenden Wort ignoriert. Tatsächlich tragen die Wortnachbarn aber viel zur Bedeutung eines Wortes bei. Mit Word Vectors kann man beispielsweise Synonyme (Wörter mit gleicher Bedeutung), Antonyme (Wörter mit gegenseitiger Bedeutung) oder auch Wörter, die zu derselben Kategorie zählen, erkennen. Word Vectors im Allgemeinen sind numerische Repräsentationen von Wortmerkmalen oder der Bedeutung von Wörtern.

Wofür benötigt man Word Vectors

Word Vectors sind hilfreich, um Wörter zu finden, welche sich aus der Bedeutung verschiedener Wörter zusammensetzen. Ein Beispiel dafür ist die Phrase: Eine Frau, die in Europa im frühen 20. Jahrhundert im Bereich der Physik etwas erfunden hat. Wenn nach einer solchen Frage im Internet gesucht wird, würde nur eine Liste von berühmten Physikerinnen und Physikern ausgegeben werden. Sobald allerdings die korrekte Antwort definiert wurde, kann es sein, dass beim nächsten Mal direkt die korrekte Antwort ausgegeben wird. Bei der Verwendung von Word Vectors ist das Prinzip dasselbe. Dabei kann nach einem Wort gesucht werden, welches die Begriffe Physik, weiblich oder Europa 20. Jahrhundert.[14]

2.2.14 Word2Vec

Word2Vec ist das berühmteste Model für Word Embedding. Unter Word Embedding wird verstanden, dass Texte, in denen Wörter dieselbe Bedeutung haben, zusammen repräsentiert werden. Es werden also alle Wörter in einem Koordinatensystem dargestellt, wobei Wörter, die zusammen gehören, nah beieinander sind. Word2Vec nimmt dabei einen großen Corpus an Text und produziert daraus einen Vektor mit den einzigartigen

Wörtern, die ähnlich zueinander sind. Es ist sehr berühmt dafür, dass demonstriert werden kann, wie Wörter miteinander verbunden sind.

Rasa hat als kleines Nebenprojekt mit **WhatLies** eine Library geschaffen, die versuchen soll, dass man Word Embeddings besser versteht. Hierbei werden Grafiken erstellen, die die Wörter in einem Koordinatensystem repräsentieren.[15]

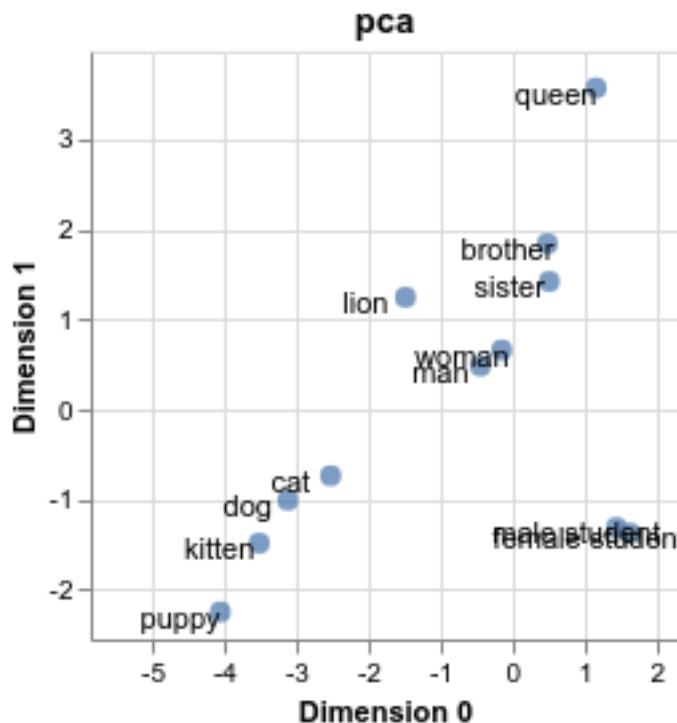


Figure 7: WhatLies Illustration [15]

Aus der Abbildung 7 ist nun zu erkennen, dass die Wörter **brother** und **sister** oder **woman** und **man** nahe beieinander im Raum dargestellt werden. Außerdem kann daraus gelesen werden, dass sich **man** zu **brother** ähnlich wie **woman** zu **sister** oder **puppy** zu **dog** sich ähnlich wie **kitten** zu **cat** verhält. Dies wird dadurch herausgefunden, dass man ein ähnliches Ergebnis bekommt, wenn man die zwei Unterschiede der Vektoren miteinander vergleicht. So ist zum Beispiel das Ergebnis bei der Rechnung **King** – **Man** + **Woman** = ? die Position des Wortes **Queen**.

Ein weiteres Beispiel wäre, dass durch Word Vectors Ähnlichkeiten gefunden werden können. Abbildung 8 zeigt dabei, dass zum Beispiel eine Fußballmannschaft für Seattle gefunden werden kann, indem man die Vektoren Seattle plus der Mannschaft aus Portland minus Portland rechnet. In Vektorschreibweise ist das Ergebnis dabei der Vektor der **Seattle Sounders** 9.[14]

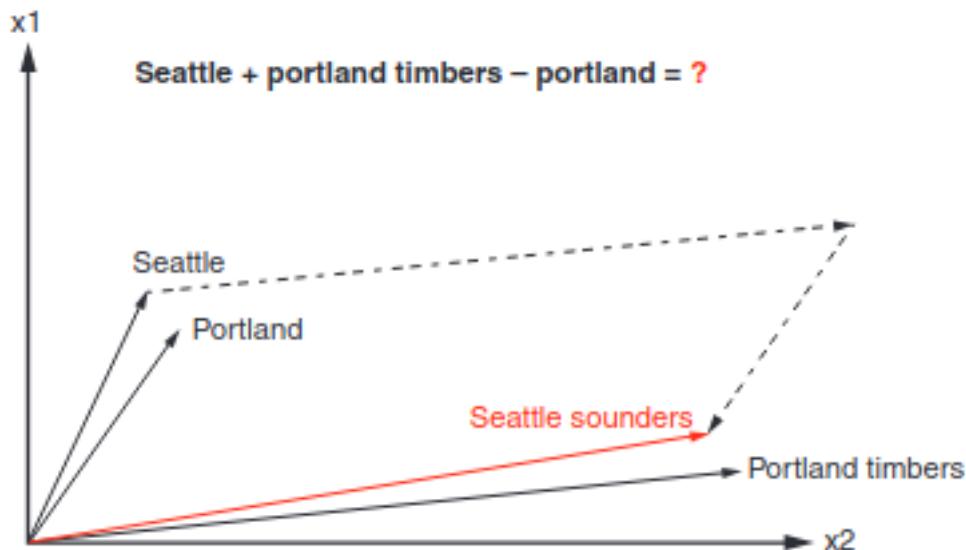


Figure 8: Whatlies Illustration [15]

$$\begin{bmatrix} 0.0168 \\ 0.007 \\ 0.247 \\ \dots \end{bmatrix} + \begin{bmatrix} 0.093 \\ -0.028 \\ -0.214 \\ \dots \end{bmatrix} - \begin{bmatrix} 0.104 \\ 0.0883 \\ -0.318 \\ \dots \end{bmatrix} = \begin{bmatrix} 0.006 \\ -0.109 \\ 0.352 \\ \dots \end{bmatrix}$$

Figure 9: Whatlies Illustration [15]

2.2.15 F-Score

Der F-Score wird oft auch als F1-Score oder F-measure bezeichnet und dieser ist ein Maß für die Genauigkeit eines Tests. Die Formel dafür ist in Abbildung 10 beschrieben. In unserem Fall wird der F-Score beispielsweise verwendet, um zu beschreiben, wie oft Intents ?? richtig klassifiziert wurden. Die `precision` gibt dabei an, wie viele Intents richtig klassifiziert wurden, von der Anzahl, wie viele es tatsächlich waren 12. Es wird also beispielsweise geschaut, wie viele Intents als `ask_ht1` identifiziert wurden und, wie viele es tatsächlich sein hätten sollen. Beim `recall` ist es auf der anderen Seite so, dass geschaut wird, wie viele Intents richtig klassifiziert wurden, und diese Anzahl wird durch die Anzahl, die der Intent vorhergesagt wurde gerechnet. 11. In unserem Beispiel würde dies bedeuten, dass die Anzahl, wie oft ein Intent tatsächlich `ask_ht1` war, dividiert durch, wie oft dieser Intent vorhergesagt wurde, wird. Je näher der F-Score dann insgesamt bei 1 liegt, desto besser ist das Modell und desto besser ist die Genauigkeit.[16]

$$\begin{aligned}
 F_1 &= \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \\
 &= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}
 \end{aligned}$$

Figure 10: F-Score Berechnung [16]

$$\begin{aligned}
 F_1 &= \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \\
 &= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}
 \end{aligned}$$

Figure 11: Precision Berechnung [17]

$$\begin{aligned}
 F_1 &= \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \\
 &= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}
 \end{aligned}$$

Figure 12: Recall Berechnung [17]

2.2.16 NLP und NLU Tools

Ein paar Beispiele für berühmte Tools, die für Natural Language Understanding (NLU) und Natural Language Processing (NLP) genutzt werden, sind:

- NLTK: Das Natural Language Toolkit (kurz NLTK) ist wahrscheinlich die berühmteste Python Library für NLP. NLTK besitzt über 50 Ressourcen für das Bearbeiten von Corpora. Außerdem gibt es viele eingebaute Libraries für das Analysieren eines Textes wie die Klassifikation, Tokenization, Stemming, Tagging, Parsing und Semantic Reasoning.
- SpaCy: SpaCy ist ebenfalls eine sehr berühmte Bibliothek für Python. SpaCy beschreibt sich selbst als **Industrial-Strength Natural Language Processing** und unterstützt über 64 Sprachen. SpaCy hat nur einen POS-Tagging-Algorithmus und pro Sprache nur einen Named-Entity-Recognizer, was den Vorteil hat, dass es nicht voller unnötiger Features ist und sich nur auf ein paar wichtige Funktionalitäten konzentriert.
- TensorFlow: TensorFlow wird von Google betrieben und ist bei weitem die meist verwendete Library für Deep Learning. Außerdem ist die Plattform Open-Source.

- Keras: Keras ist eine Deep Learning Plattform, die in Python geschrieben ist. Sie wurde entworfen um schnelle Iterationen und schnelles Ausprobieren mit deep neural networks zu ermöglichen. Außerdem bietet Keras ein Interface zu deep neural networks an, welche dann beispielsweise auf TensorFlow, Theano oder anderen Backends laufen können.
- CoreNLP: Das CoreNLP Projekt von Stanford ist ein NLP Toolkit, welches zwar in Java geschrieben wurde, aber auch in anderen Sprachen verwendet werden kann, die auf der JVM Plattform sind.
- Gensim: Gensim ist eine Open-Source Library für Python, mit der man semantische NLP Modelle trainieren kann. Außerdem ist Gensim Plattform unabhängig und läuft auf jedem Betriebssystem.
- TextBlob: TextBlob ist eine Python Library für das Bearbeiten von Texten.
- FastText: FastText ist eine Open-Source Library, die es Benutzern erlaubt text representations und text classifier zu verwenden.

2.2.17 NLP in Rasa

In Rasa kann man sich die verschiedenen Schritte vom Natural Language Processing ansehen, indem man sich die Pipeline ansieht. Diese befindet sich in der **config.yml** Datei und dort sind alle Komponenten aufgelistet, die Rasa verwendet und man kann diese nach Belieben auch selbst bearbeiten.

3 Toolstack

3.1 Programmiersprachen

3.1.1 Java



Figure 13: Java Logo[18]

Für das Backend wurde sich für Java entschieden. Java ist laut dem TIOBE-Index [19] eine der populärsten Programmiersprachen. Java ist eine für Menschen sehr einfach lesbare Sprache, aber damit Maschinen sie auch lesen können, wird ein Java-Compiler der den Code zu einem Java-Bytecode umwandelt, benötigt. Ohne diesen Vorgang kann der Code nicht ausgeführt werden. Java ist eine objektorientierte Programmiersprache und besitzt dadurch Klassen und Vererbung[18].

3.1.2 Python

Python ist eine Programmiersprache, die mehrere Arten der Programmierung unterstützt, wie zum Beispiel die objektorientierte, die aspektorientierte und die funktionale Programmierung. Es handelt sich hier um eine Programmiersprache, die versucht einen knappen und dadurch gut lesbaren Programmierstil zu besitzen. Dadurch, dass Python mit nur wenigen Schlüsselwörtern auskommt und die Syntax reduziert ist, ist eine gute Übersichtlichkeit und eine optimierte Simplizität gegeben. Python wird oft für Machine Learning

verwendet, auch bei dieser Arbeit wurde Python zusammen mit Rasa verwendet. Zum einen ist Rasa ein Python-Framework und zum anderen werden die Custom Actions 4.4.10 mit Python umgesetzt.

3.1.3 Typescript

Typescript ist eine im Jahr 2012 von Microsoft entwickelte Programmiersprache, die eine kompakte und einfache Syntax zur Programmierung von Webseiten und Anwendungen bietet. Typescript baut auf Javascript auf und hilft zum Beispiel beim frühzeitigen Erkennen von Fehlern. Bei TypeScript wird mit Unterstützung von Modulen das Kapseln von Klassen, Interfaces, Funktionen und Variablen in eigene Namensräume ermöglicht. Es gibt hierbei eine Unterscheidung zwischen internen und externen Modulen.

3.2 Technologien

3.2.1 REST Service

REST steht für Representational State Transfer. REST Anfragen sind CRUD – Operationen (Create, Read, Update, Delete). Diese sind GET, POST, PUT und DELETE Requests. Außerdem gibt es noch OPTIONS, PATCH, HEAD, TRACE und CONNECT. Diese werden aber in der vorliegenden Arbeit nicht benutzt.

- Die GET Methode ist zuständig für das Abfragen von Daten. Dabei soll ein Request geschickt werden und es werden nur Daten zurückgegeben. Auf dem Server, zu dem die Anfrage geschickt wird, soll nichts geschehen außer das Lesen von Daten und das anschließende Zurückgeben dieser.
- Bei POST sollen Daten hinzugefügt werden. Im Normalfall ist in der Response der URI der neu gespeicherten Daten.
- Bei PUT sollen Daten hinzugefügt werden. Falls diese schon existieren, werden sie upgedatet und falls nicht, wird ein neuer Eintrag erstellt.
- Bei DELETE sollen Daten gelöscht werden.

3.2.2 Quarkus

Quarkus ist ein Framework für Java, welches sich auf die REST-Service-API ausrichtet. Das Backend 5.2 wurde mit Quarkus umgesetzt.

3.2.3 Angular

Angular ist ein auf TypeScript basierendes Framework für Webseiten. Dieses ist ausgelegt für Single Page Applications. Das Chat Widget 5.3 und das Dashboard 5.4 wurden mit Angular umgesetzt

3.2.4 Angular Elements

”Webkomponenten sind eine Gruppe von Web-Technologien, die es ermöglichen, benutzerdefinierte, wiederverwendbare HTML Elemente zu erstellen, deren Funktionalität gekapselt ist und damit vollständig getrennt von anderem Code.”[20]

Um die Chatkomponente auf die Wordpress Seite zu bringen, wurde entschieden, dass die Komponente mithilfe von Angular Elements als Webkomponente exportiert wird um somit eine einfache Einbindung auf die Schulhomepage zu ermöglichen.

Eine Webkomponente wird in Angular Elements in 3 Javascript Files erstellt. Diese sind

1. main.js
2. polyfills.js
3. runtime.js

Dabei können diese drei Komponenten jedoch ohne Bedenken zu einem einzelnen Javascript File zusammengefasst werden.

Um die Webkomponente in HTML zu benutzen, müssen alle JS-Dateien eingebunden werden und schon kann die eigene Komponente wie jedes andere Element benutzt werden.

Listing 10: HTML File

```
1 ...
2 <name-der-komponente></name-der-komponente>
3
4 <script src="./runtime.js"></script>
5 <script src="./polyfills.js"></script>
6 <script src="./main.js"></script>
7 ...
```

3.3 Werkzeuge

3.3.1 Rasa X

Rasa X ist ein Werkzeug um Conversation-Driven Development 5.4.1 in die Tat umzusetzen.[21] Rasa X liegt über Rasa und liefert dadurch erweiterte Funktionalitäten, die nur mit Rasa alleine nicht gegeben sind. Rasa X wurde mit Docker Compose [22] auf der VM installiert 5.1.

Rasa X kommt mit einer grafischen Weboberfläche, auf dieser können Intents 4.4.1, Responses 4.4.2 und so weiter verändert und angesehen werden. Außerdem besteht die Möglichkeit ein eigenes Model zu trainieren oder ein Model hochzuladen.

Da jedoch ein eigenes Dashboard im Rahmen dieser Diplomarbeit verwendet wird, wird die grafische Oberfläche nicht benötigt, sondern nur die erweiterte Funktionalität die auch über die API aufgerufen werden kann.

Damit das Frontend auf Rasa, welches eine Ebene unter Rasa X liegt, zugreifen kann, muss das automatisch erstellte `credentials.yml` File bearbeitet werden. Dies ist in Listing 11 beschrieben.

Listing 11: credentials.yml

```

1 rasa:
2   url: ${RASA_X_HOST}/api
3   rest:
```

Eine Herausforderung stellte dabei die Möglichkeit, dass Rasa X mit https läuft, dar. Es wird ein Zertifikat benötigt. Dabei ist es möglich ein Zertifikat zu erstellen mithilfe des Certbots [23], diese Zertifikate müssen dann in den `certs` Ordner gegeben werden.

Listing 12: Install certbot and create certificates

```

1 sudo ln -s /snap/bin/certbot /usr/bin/certbot
2 sudo certbot certonly -d vm45.htl-leonding.ac.at
3 sudo cp /etc/letsencrypt/live/vm45.htl-leonding.ac.at/privkey.pem /etc/rasa/certs/
4 sudo cp /etc/letsencrypt/live/vm45.htl-leonding.ac.at/fullchain.pem
      /etc/rasa/certs/
5 sudo chmod 640 certs/privkey.pem
```

Außerdem müssen beim Nginx Server noch diverse Einstellungen vorgenommen werden. In dem File `nginx-config-files/rasax.nginx.template` muss die Zeile mit “include” auskommentiert werden.

Listing 13: rasax.nginx.template

```

1 ...
2 server {
3     listen           8080;
```

```

4      include          /etc/nginx/conf.d/ssl.conf;
5      ...

```

Und in “nginx-config-files/ssl.conf.template” müssen die Zeilen, wie in Listing 14 beschrieben, auskommentiert werden.

Listing 14: ssl.conf.template

```

1 listen           8443 ssl;
2
3 # server_name    example.com;
4 ssl_certificate  /etc/certs/fullchain.pem;
5 ssl_certificate_key /etc/certs/privkey.pem;

```

Nun muss Rasa X gestoppt und wieder hochgefahren werden, um die neuen Einstellungen zu übernehmen.

3.3.2 IntelliJ IDEA

IntelliJ ist ein IDE, welches eine Vielzahl von Programmiersprachen unterstützt und hauptsächlich für diese Arbeit verwendet wird.

3.3.3 GitHub

GitHub ist ein Online-Repository, welches die gemeinsame Programmierung von Anwendungen erleichtert.

3.3.4 GitHub Actions

Mit Github Actions können Workflows für ein Github Repository erstellt werden. Ein Workflow besteht aus einem oder mehreren Jobs, wobei ein Job aus einen oder mehreren Schritten besteht.

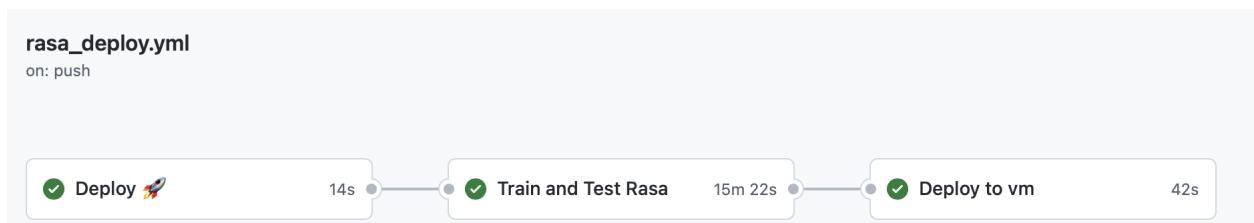


Figure 14: Beispiel eines Workflows

Die Jobs werden dabei automatisch abgearbeitet und meistens wird am Ende des Workflows das Produkt woher hochgeladen. Bei jeder Action wird standardmäßig ein checkout benutzt:

Listing 15: In der Action

```
1 ...  
2 jobs:  
3   build:  
4     runs-on: ubuntu-latest  
5     steps:  
6       - name: Checkout  
7         uses: actions/checkout@v2  
8 ...
```

Checkout sorgt dabei dafür, dass der Workflow auf das Projekt zugreifen kann.

3.3.5 Docker

Docker ist ein Container-Manager, der es erlaubt Anwendungen zu isolieren. Dies wird in sogenannten Docker-Containern gemacht.

3.3.6 Wordpress

WordPress ist ein freies und open-source Content-Management-System, das in PHP geschrieben ist. Für die Speicherung der Daten wird entweder eine MySQL oder eine MariaDB Datenbank verwendet. WordPress verwendet Plugins für die Erweiterung der Funktionalitäten und Themes für die Anpassung des Designs. Das CMS wurde ursprünglich für das Erstellen von Blogs benutzt, hat sich jedoch inzwischen zu einem System für das Verwalten verschiedenster Web-Inhalte entwickelt. Mit einem Marktanteil von 42,8% der 10 Millionen meistbesuchten Websites ist WordPress das beliebteste Content-Management-System.

Die HTL Leonding Schulhomepage wurde mithilfe von Wordpress gemacht, deshalb war es wichtig, dass das Endprodukt der Arbeit auch mit Wordpress kompatibel ist.

Auf Wordpress können Beiträge verfasst werden, wie in Abbildung 16 zu sehen ist.

Diese Beiträge können in einem eigenen Editor, der viele vorgefertigte Elemente bereits zur Verfügung hat verfasst werden. Dies wird in Abbildung 17 beschrieben.

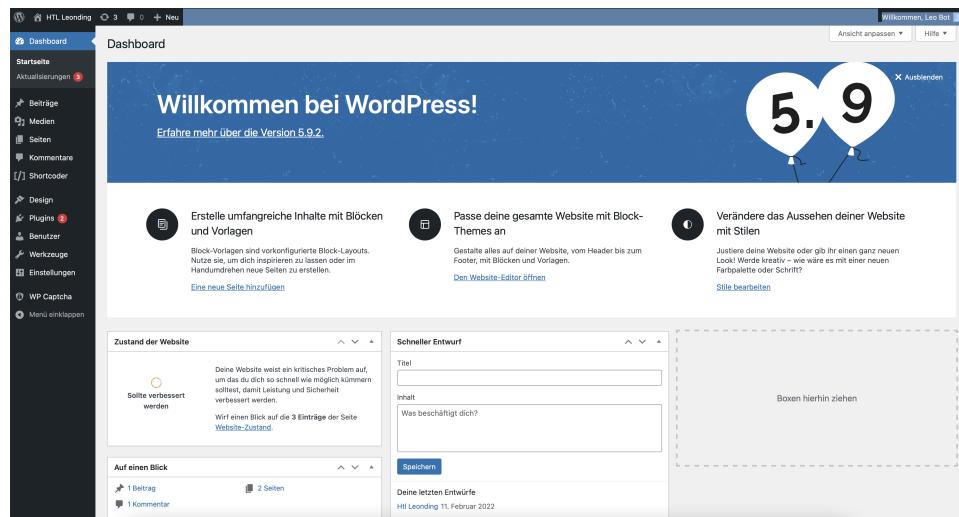


Figure 15: Startseite Wordpress

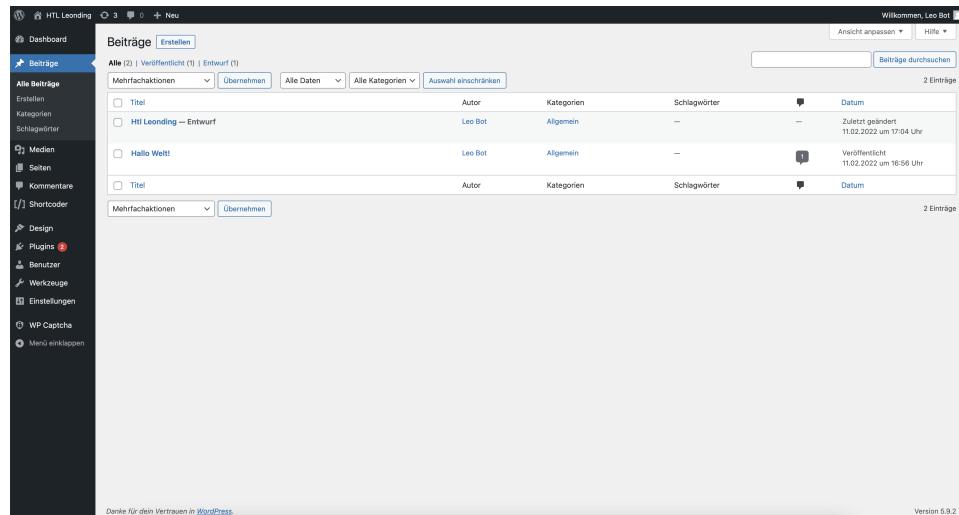


Figure 16: Wordpress Liste der Blogeinträge

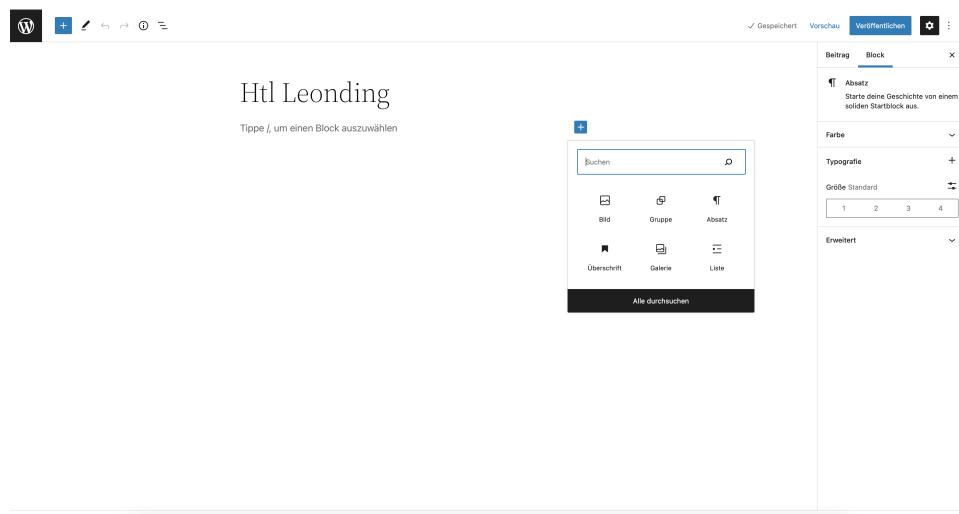


Figure 17: Wordpress Blopost Editor

4 Chatbots am Beispiel von Rasa

4.1 Allgemeines

4.1.1 Rasa Produkte

Der Rasa Stack wird in Rasa NLU und Rasa Core aufgeteilt. Diese sind so aufgebaut, dass sie unabhängig voneinander eingesetzt werden können. So besteht die Möglichkeit, nur einen Teil der Architektur auf Rasa aufzubauen und zusätzlich weitere Services mit einzubinden.

4.1.2 Rasa Core

Rasa Core bezieht sich auf die Hauptkomponente, die die Nachrichten erhält und darauf antwortet.[24]

Der Rasa Core hält für jede Session, also für jeden User, einen Tracker. Dieser enthält den momentanen Zustand der Konversationen der jeweiligen User. Bekommt der Bot nun eine Nachricht, wird zuerst der Interpreter durchlaufen, welcher den Originaltext als Eingabe bekommt und die Eingabe den Intent und die extrahierten Entities zurückgibt. Zusammen mit dem aktuellen Zustand des Trackers entscheidet die Policy 4.1.3 Komponente nun, welche Action 4.4.9, also Antwort des Bots, als Nächstes ausgeführt werden soll. Diese Entscheidung wird nicht durch einfache Regeln getroffen, sondern genauso wie Intents 4.4.1 oder Entities 4.4.6, auf der Grundlage von einem, mit Machine Learning, trainierten Model.[24, 25]

4.1.3 Policies

Policies sind ein Teil von **Rasa Core** 4.1.2 und der Assistent benutzt Policies um zu entscheiden, welche Action als Nächstes ausgeführt werden soll. Es gibt machine-learning und rule-based Policies.[26]

Hierbei können die Policies geändert werden, indem man diese in der **config.yml** Datei ändert.

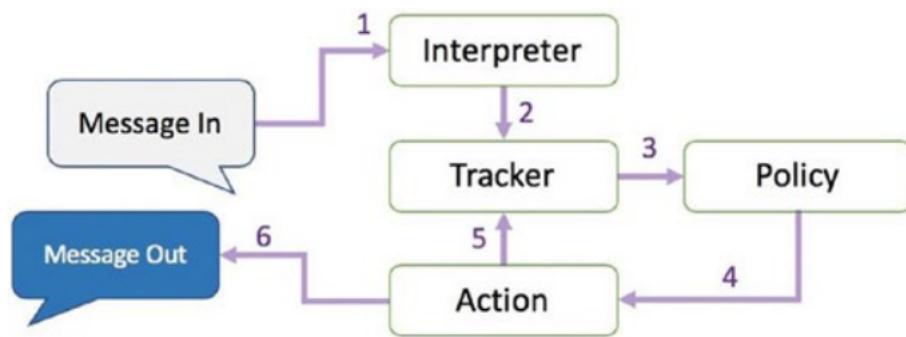


Figure 18: Rasa Core Aufbau [25]

Bei den Policies gibt es unterschiedliche Priorities, die dann zum Einsatz kommen, wenn mehrere Policies dieselbe Confidence vorhergesagt haben.[27]

TED Policy

Die TED Policy steht für Transformer Embedding Dialogue Policy und wird meistens standardmäßig verwendet.[28]

Bei jedem Dialog bekommt die TED Policy drei Informationen als Input. Die Nachricht des Users, die vorherige Action 4.4.9 die vorhergesagt wurde und Slots 4.4.5 und aktive Forms 4.4.7. Danach wird die Ähnlichkeit zwischen den System Actions und dem eingegebenen Text berechnet und zum Schluss werden noch CRF 4.2.7 Algorithmen verwendet, um Entities 4.4.6 zu erkennen.[28]

4.1.4 Rasa NLU

Rasa NLU hat grundsätzlich zwei Hauptaufgaben.

Zum einen wäre da die Intent Recognition und die Entity Recognition.[29, 30]

Die Intent Recognition, ist die Erkennung der Nutzer-Absichten. Dazu muss die NLU mit ausreichend Utterances, also Responses 4.4.2 trainiert werden. Dabei gibt die NLU alle zugehörigen Intents 4.4.1 geordnet nach dem Confidence Score zurück. Dieser Score gibt an, wie sicher sich das Modell ist, dass die jeweilige Antwort die richtige Antwort für den Text der Benutzerin oder des Benutzers ist. Rasa verfügt demnach über ein **Multi Intent Matching**.[29, 30]

Außerdem gibt es noch die Entity Recognition, die dafür zuständig ist Entities 4.4.6, also wichtige Informationen, aus natürlicher Sprache zu extrahieren.[29]

Der Aufbau der NLU ist vollständig konfigurierbar mithilfe der sogenannten Pipeline 4.2.[31]

4.2 Pipeline

Rasa Open Source bietet bei der Initialisierung 4.5 des Projekts eine Standard-NLU-Konfiguration.[32]

In Rasa Open Source werden eingehende Nachrichten von einer Reihe von Komponenten 4.4 verarbeitet. Diese Komponenten werden nacheinander in einer sogenannten Processing Pipeline ausgeführt, die im `config.yml` File definiert ist.[31]

In der Abbildung 19 werden die Komponenten und ihre Lifecycle abgebildet.

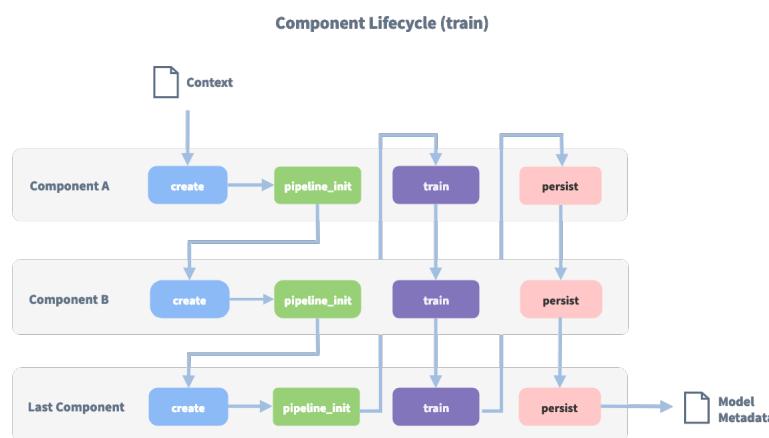


Figure 19: Component Lifecycle [33]

Bevor die erste Komponente mit der Create-Funktion erstellt wird, wird ein sogenannter Kontext erstellt. Dieser ist ein Python Dictionary, also ein Key-Value-Paar und er wird verwendet, um Informationen zwischen den Komponenten zu übergeben.[33, 34]

Zunächst wird der Kontext mit allen Konfigurationswerten gefüllt, die Pfeile im Bild 20 zeigen die Aufrufreihenfolge und visualisieren den Pfad des übergebenen Kontexts.[33, 34] Dies ist in der Grafik 20 zu erkennen.

4.2.1 Arten von NLU Pipelines

Es gibt verschiedene bereits konfigurierte Pipelines.[31]

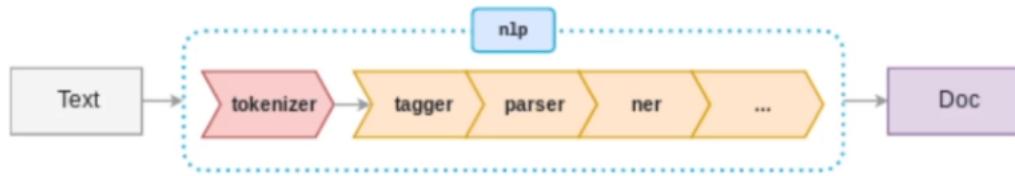


Figure 20: Component Lifecycle [35]

Grundsätzlich ist zwischen Pipelines mit oder ohne Pretrained Word Vectors zu unterscheiden.

Beispiele für Pipelines mit Pretrained Word Vectors

Der Vorteil von Pipelines, die Pretrained Word Vectors verwenden sind, dass diese bereits aus der jeweiligen Sprache Vektoren für die Wörter 2.2.13 besitzen. Somit weiß das Model beispielsweise, dass Äpfel und Birnen ähnlich sind, ohne dass dies in den Intents irgendwo spezifiziert werden muss, weil es bereits aus den Wordvektoren hervorgeht.[36, 37]

Die Vorteile hierbei sind also, dass das Model weniger Trainingsdaten benötigt, um eine gute Performance zu besitzen. Außerdem geht der ganze Trainingsprozess in der Regel schneller und die Zeit, die benötigt wird, um die Trainingsdaten zu durchlaufen, ist kürzer als bei Modellen ohne Pretrained Word Vectors.[36, 30, 37] Ein Beispiel für eine Pipeline mit vortrainierten Vektoren ist die `spacy_sklearn` Pipeline, die man in der `config.yml` Datei, wie in Listing 16 definiert.

Listing 16: SpaCy Sklearn Pipeline

```

1 language: "de"
2
3 pipeline: "spacy_sklearn"

```

Die SpaCy Pipeline verwendet Pretrained Word Vectors von GloVe oder fastText. Dies sind Algorithmen, die einen Textkorpus 2.2.1 einer gewissen Sprache in Form von Vektoren einarbeiten.[38, 39, 37]

Es gibt außerdem auch noch Pipelines von MITIE. Diese verwenden MITIE als Source für die Word Vectors 2.2.13. Ein Vorteil von MITIE ist, dass man hier auch seine eigenen Word Vectors 2.2.13 trainieren kann, indem man einen Korpus 2.2.1 von Wikipedia oder ähnlichen Seiten verwendet. Allerdings wird MITIE meistens nicht empfohlen und es könnte auch sein, dass MITIE demnächst deprecated sein wird. Die MITIE Pipeline kann man wie in Listing 17 definieren.[40, 41]

Listing 17: MITIE Sklearn Pipeline

```

1 language: "de"
2
3 pipeline: "mitie_sklearn"

```

Beispiele für Pipelines ohne Pretrained Word Vectors

Der Vorteil von Pipelines ohne Pretrained Word Vectors ist, dass diese speziell auf den Fachbereich angepasst sind, für den man den Chatbot entwickelt.[30, 42]

Als Beispiel kann man die Wörter `balance` und `symmetry` aus dem Englischen sehen. Diese Wörter sind eng miteinander verwandt. Allerdings kann im Kontext von Banken das Wort `balance` auch mit `cash` verwandt sein. Bei einem vortrainierten Modell würden diese Wortvektoren nicht nah aneinander liegen, aber wenn ein Chatbot nur Intents besitzt, die mit Banken und Rechnungswesen zu tun haben, werden diese zwei Wörter `balance` und `cash` ohne Pretrained Word Vectors als ähnlich erkannt werden.[42]

Außerdem benutzen diese Pipelines kein sprach-spezifisches Modell und somit kann man sie in allen Sprachen verwenden, die tokenisiert 1 werden können. Eine solche Pipeline kann man wie in Listing 18 definieren.[42]

Listing 18: Tensorflow Embedding Pipeline

```

1 language: "de"
2
3 pipeline: "tensorflow_embedding"

```

Ein Problem von diesem Ansatz ist jedoch, dass er oftmals keine fachspezifischen Begriffe kennt und außerdem können Typos nicht als Wortvektoren gelernt werden. Außerdem ist das Problem bei Pretrained Word Vectors, dass Zehntausende Vektoren gespeichert werden, die vermutlich nie verwendet werden.[42, 43]

Das Tensorflow-Embedding ist das genaue Pendant dazu. Diese Pipeline verwendet keine vortrainierten Vektoren, kann mit jeder Sprache verwendet werden. Sie lernt dabei Embeddings für die Intents 4.4.1 und für die Wörter und die Embeddings werden verwendet um die Ähnlichkeit zwischen dem Input und den Intents 4.4.1 zu ermitteln.[42, 43]

Eine solche Pipeline wird wie in Listing 19 definiert.

Listing 19: Supervised Embedding Pipeline

```

1 language: "de"

```

```

2
3 pipeline: "supervised_embedding"

```

Supervised Embeddings vs. Pretrained Embeddings

In der Grafik 21 ist ein Flussdiagramm abgebildet, welches beschreibt, wann Supervised Embeddings oder Pretrained Embeddings von Vorteil sind.

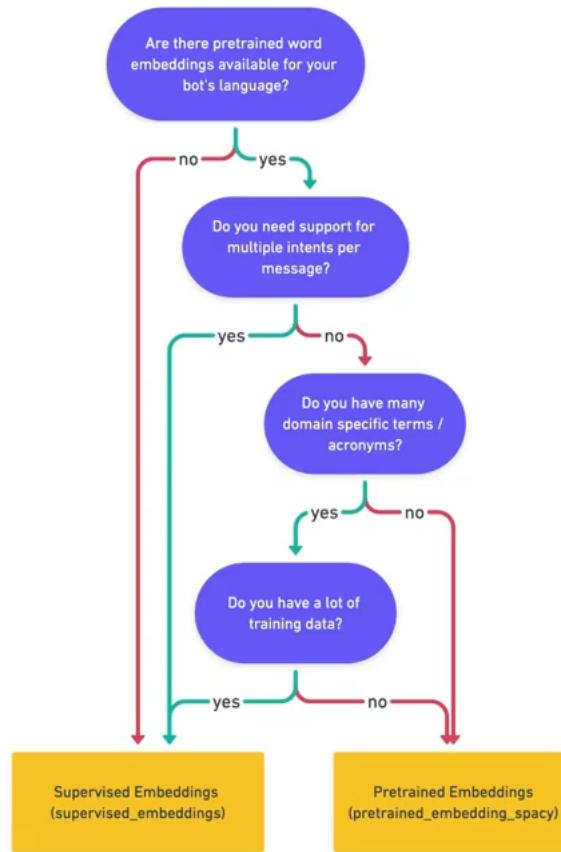


Figure 21: Pretrained oder Supervised Embeddings [30, 37]

4.2.2 Konfigurierte Pipelines

Es gibt vordefinierte Pipelines von Rasa, die benutzt werden können.

Sollte man sich für Word Embeddings entscheiden, kann man auf eine Pipeline mit SpaCy zurückgreifen, welche den SpacyFeaturizer verwendet und somit Pretrained Word Embeddings nutzt. Diese ist in Listing 20 zu sehen.[44, 45]

Listing 20: Spacy Startpipeline

```

1 language: "de" # Code f r die Sprache
2
3 pipeline:
4   - name: SpacyNLP

```

```

5     model: "de_core_news_lg" # Spezifizierung, welches Modell von SpaCy genommen
6         werden soll
7 - name: SpacyTokenizer
8 - name: SpacyFeaturizer
9 - name: RegexFeaturizer
10 - name: LexicalSyntacticFeaturizer
11 - name: CountVectorsFeaturizer
12 - name: CountVectorsFeaturizer
13     analyzer: "char_wb"
14     min_ngram: 1
15     max_ngram: 4
16 - name: DIETClassifier
17     epochs: 100
18 - name: EntitySynonymMapper
19 - name: ResponseSelector
20     epochs: 100

```

Wenn man sich dafür entscheidet, keine vortrainierten Word Embeddings zu nutzen und somit sein Modell spezifischer auf den eigenen Anwendungsfall anpassen möchte, verwendet man die Standard Rasa Pipeline. Diese verwendet den CountVectorsFeaturizer, bei dem nur die Trainingsdaten, die zur Verfügung gestellt werden, trainiert werden.[44, 45, 46] In der config.yml Datei würde die Pipeline 4.2 somit wie in Listing 21 aussehen.

Listing 21: Default Pipeline

```

1 language: "de" # Code f r die Sprache
2
3 pipeline:
4     - name: WhitespaceTokenizer
5     - name: RegexFeaturizer
6     - name: LexicalSyntacticFeaturizer
7     - name: CountVectorsFeaturizer
8     - name: CountVectorsFeaturizer
9         analyzer: "char_wb"
10        min_ngram: 1
11        max_ngram: 4
12    - name: DIETClassifier
13        epochs: 100
14    - name: EntitySynonymMapper
15    - name: ResponseSelector
16        epochs: 100

```

4.2.3 Verwendete Pipeline

Die für die Umsetzung des Chatbots verwendete Pipeline aus Listing 22 beruht dabei auf dieser Standard-Pipeline. Die jeweiligen Komponenten davon werden in den folgenden Kapiteln 4.4 genauer erläutert.

Listing 22: Unsere Pipeline

```

1 language: de
2
3 pipeline:
4     - name: WhitespaceTokenizer
5     - name: RegexFeaturizer
6     - name: LexicalSyntacticFeaturizer
7     - name: CountVectorsFeaturizer
8     - name: CountVectorsFeaturizer
9         analyzer: char_wb
10        min_ngram: 1
11        max_ngram: 4

```

```

12      - name: DIETClassifier
13          epochs: 100
14          constrain_similarities: true
15      - name: EntitySynonymMapper
16      - name: ResponseSelector
17          epochs: 100
18          retrieval_intent: faq
19      - name: ResponseSelector
20          epochs: 100
21          retrieval_intent: chitchat
22      - name: FallbackClassifier
23          threshold: 0.3
24          ambiguity_threshold: 0.1

```

4.2.4 Pipelines vergleichen

Um die beste Leistung für definierte Trainingsdaten zu erzielen, gibt es die Möglichkeit zwei verschiedenen Pipelines miteinander vergleichen. Dafür wird der `rasa test` Befehl verwendet, der dabei eine Reihe von verschiedenen Schritten ausführt, wie in Listing 23 zu sehen ist.

1. Zunächst werden 80 % der Daten von der `data/nlu.yml` Datei trainiert und 20 % getestet.
2. Danach wird eine gewisse Prozentzahl von Daten ausgelassen.
3. Die Modelle werden basierend auf den überbleibenden Daten trainiert.
4. Eine Evaluation über das Modell mit den reduzierten Trainingsdaten wird erstellt.[47]

Der zweite Schritt wird dabei standardmäßig dreimal für jede übergebene Pipeline-Konfiguration ausgeführt. Dabei werden immer wieder neue Prozentzahlen gewählt. Um diese zu verändern, gibt es die `--runs` Flag, die in Listing 24 zu sehen ist.[47]

Listing 23: Unsere Pipeline

```

25 rasa test nlu --nlu data/nlu.yml
26   --config config_1.yml config_2.yml

```

Listing 24: Unsere Pipeline

```

27 rasa test nlu --nlu data/nlu.yml
28   --config config_1.yml config_2.yml
29   --runs 4 --percentages 0 25 50 70 90

```

Evaluation der Ergebnisse

Nachdem nun ein ausführlicher Test aller NLU Daten durchgeführt wurde, wird das Ergebnis in einem `results/` Ordner gespeichert.[48]

Dort befindet sich nun direkt im Ordner ein File mit dem Titel `nlu_model_comparision_graph.pdf`. In der Abbildung 22 ist zu erkennen, dass hierbei nur 1 Konfigurationsfile genutzt wurde, bei dem es sich um die im Zuge der Arbeit gewählte Pipeline 22 handelt. Auf der x-Achse befindet sich die Anzahl der Trainingsphrasen, die beim Training dabei waren. Hier ist zu sehen, dass je höher die Anzahl ist, desto genauer auch der F1-Score 2.2.15 ist. Vor allem aber sind die Abweichungen bei diesem Score viel kleiner, wenn mehr Trainingsphrasen gegeben sind. Dies beweist, wie wichtig es ist, viele Trainingsphrasen zu verwenden.

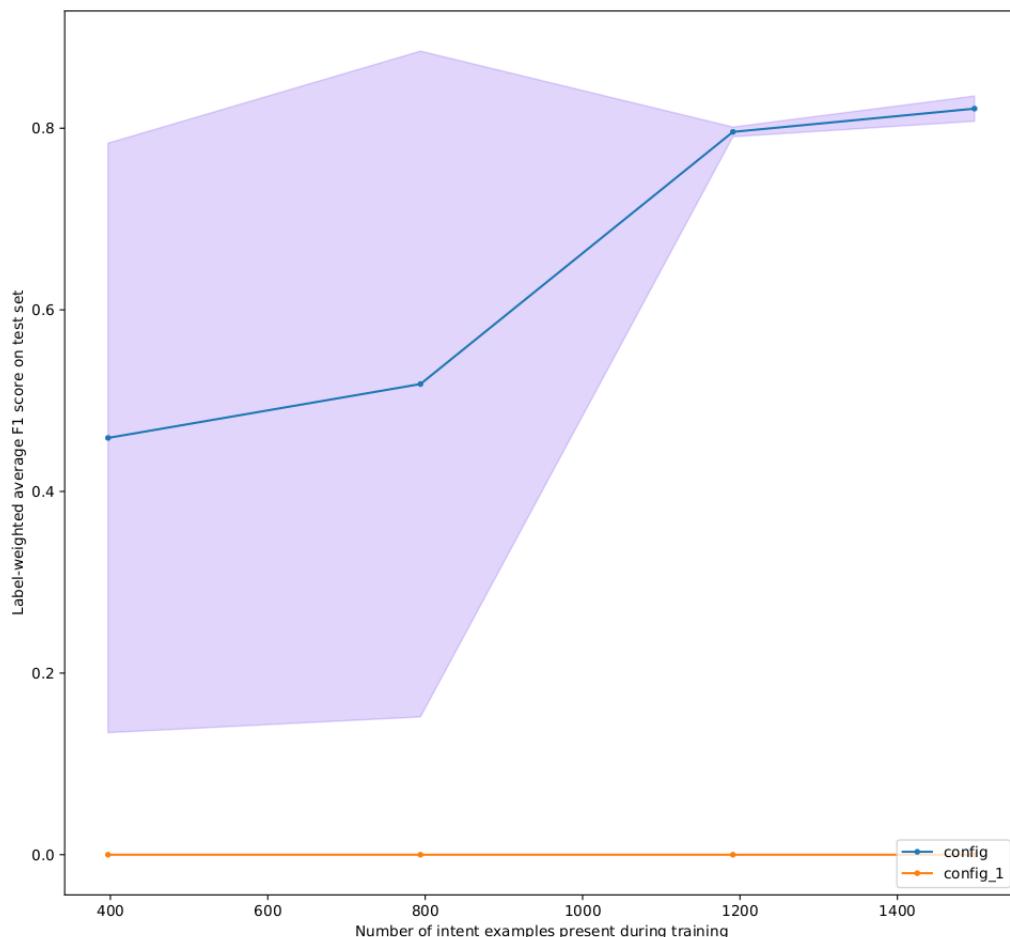


Figure 22: Comparision Graph bei unserer Pipeline

Außerdem werden noch `.json` Dateien und Bilder für die jeweiligen Ergebnisse erzeugt. Für jeden Lauf wird dabei ein Ordner angelegt, in dem es Unterordner für die jeweilige Prozentzahl gibt, die von den Trainingsphrasen ausgeschlossen wurde. Dabei findet sich jeweils eine sogenannte Konfusionsmatrix und ein Histogramm der Ergebnisse.

Beim Histogramm wird die Anzahl der korrekt vorhergesagten Intents 4.4.1 mit der von den falsch vorhergesagten Intents 4.4.1 verglichen. Dabei wird auch die Confidence, also die Sicherheit des Modells, angegeben, mit der sie vorhergesagt wurden. Im Optimalfall

sind die Balken also bei den korrekten Intents weit oben mit einer hohen Confidence versehen und die Falschen mit einer niedrigen. Außerdem soll die Anzahl, die auf der x-Achse eingetragen ist, von den richtigen Intents höher sein als die Anzahl der falschen Intents. In Abbildung 23 ist ein Histogramm für 0% Exklusion zu sehen und bei Abbildung 24 wurde 75% Exklusion genutzt. Es ist zu erkennen, dass bei mehr Exklusion auch mehr Intents 4.4.1 falsch vorhergesagt werden und die Zuversichtlichkeit, also die Confidence, des Modells dabei immer höher wird.

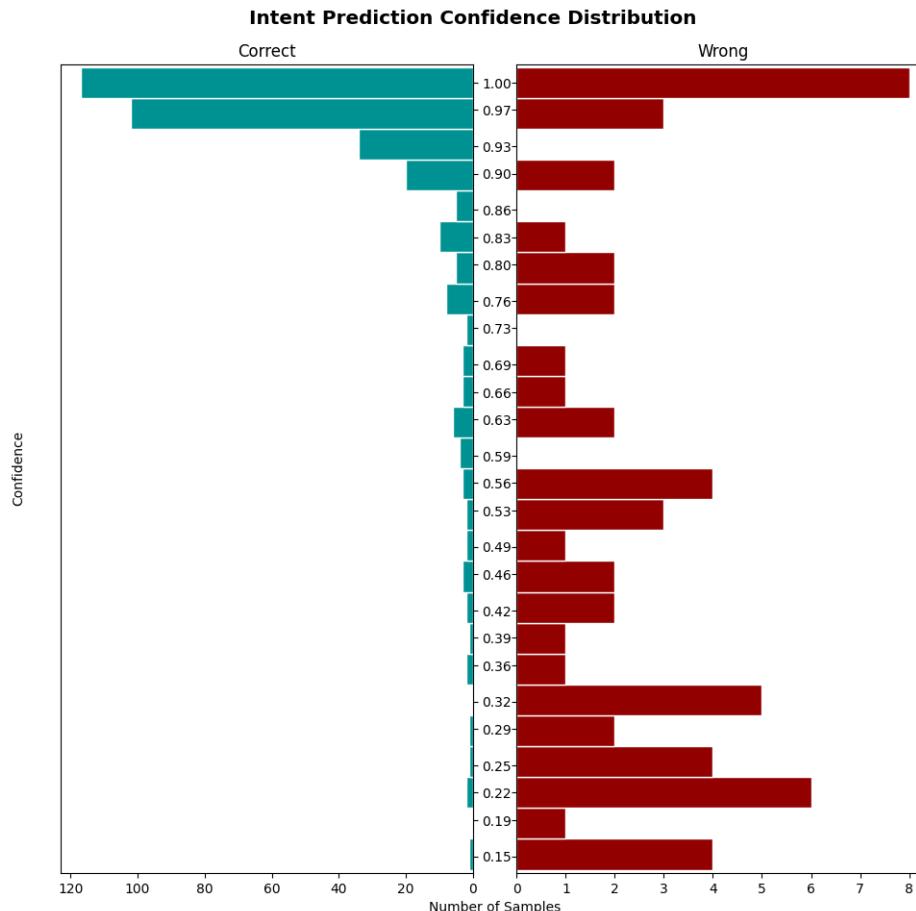


Figure 23: Histogramm für 0% Exklusion

Die Konfusionsmatrix hilft dabei festzustellen, wie viele Intents 4.4.1 richtig und falsch erkannt wurden. Dabei wird auf einer Achse aufgetragen, welcher Intent vorhergesagt wurde, und welcher Intent tatsächlich erkannt hätte werden sollen. Im Optimalfall liegt bei der Konfusionsmatrix also eine Diagonale durch die Matrix vor. Bei der Konfusionsmatrix ist anschaulich zu erkennen, welche Intents noch überarbeitet werden sollten, weil zu viele Intents falsch als dieser vorhergesagt wurden. In Abbildung 25 kann man beispielsweise ablesen, dass zwei Intents, die eigentlich `chitchat/insult` sein sollten fälschlicherweise als `chitchat/ask_creator` erkannt wurden.

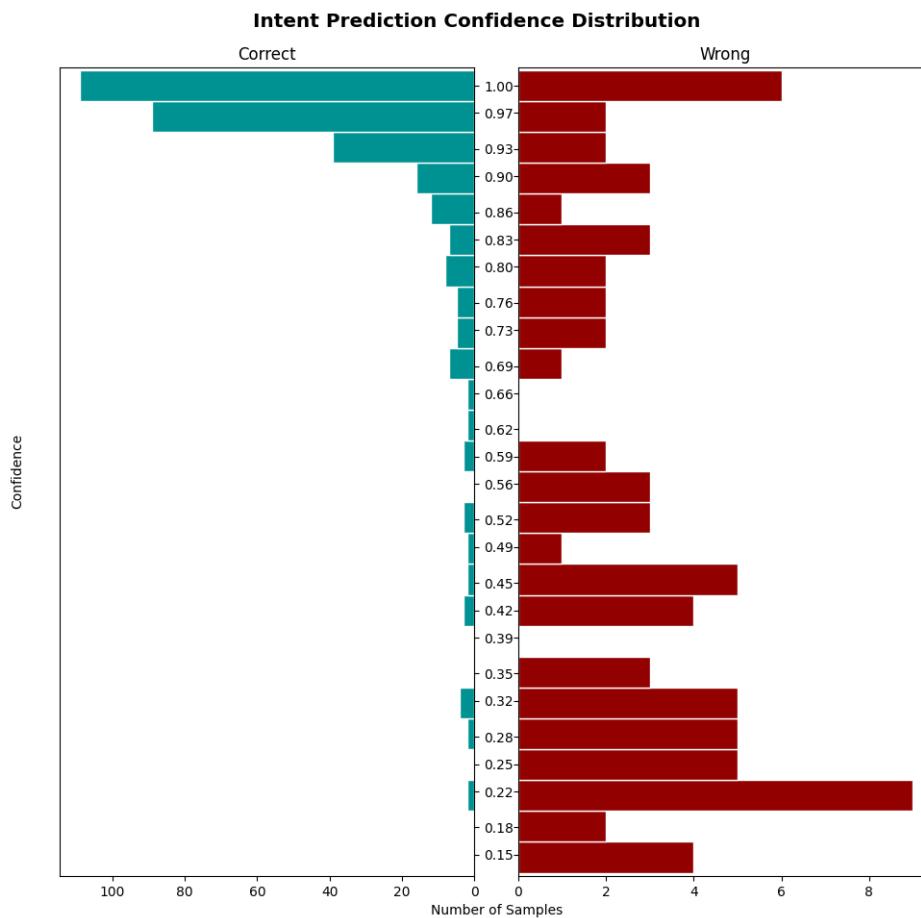


Figure 24: Histogramm für 75% Exklusion

4.2.5 WhitespaceTokenizer

Ein WhitespaceTokenizer ist eine sehr einfache Art eines Tokenizers [1]. Hierbei wird, wie der Name bereits vermuten lässt, der Satz aufgeteilt in verschiedene Tokens. Ein Token kann zum Beispiel also ein Wort sein und beim WhitespaceTokenizer wird der Satz pro Whitespace, also pro Leerzeichen, aufgeteilt.[49, 50, 51]

Es gibt auch komplexere Tokenizer und Tokenizer, die speziell an Sprachen angepasst sind und die besonderen Regeln, die in diesen Sprachen gelten, beachten. Außerdem besteht die Möglichkeit, einen Tokenizer selbst zu implementieren.[49, 50, 51] Tokenizer sollten generell weit am Anfang einer Pipeline 4.2, wenn nicht sogar die erste Komponente, sein.

4.2.6 RegexFeaturizer

Einen RegexFeaturizer kann man verwenden, um die EntityExtraction zu erleichtern. Bei diesem werden Regular Expressions 4.2.6 und Lookup Tables 4.2.6 verwendet und

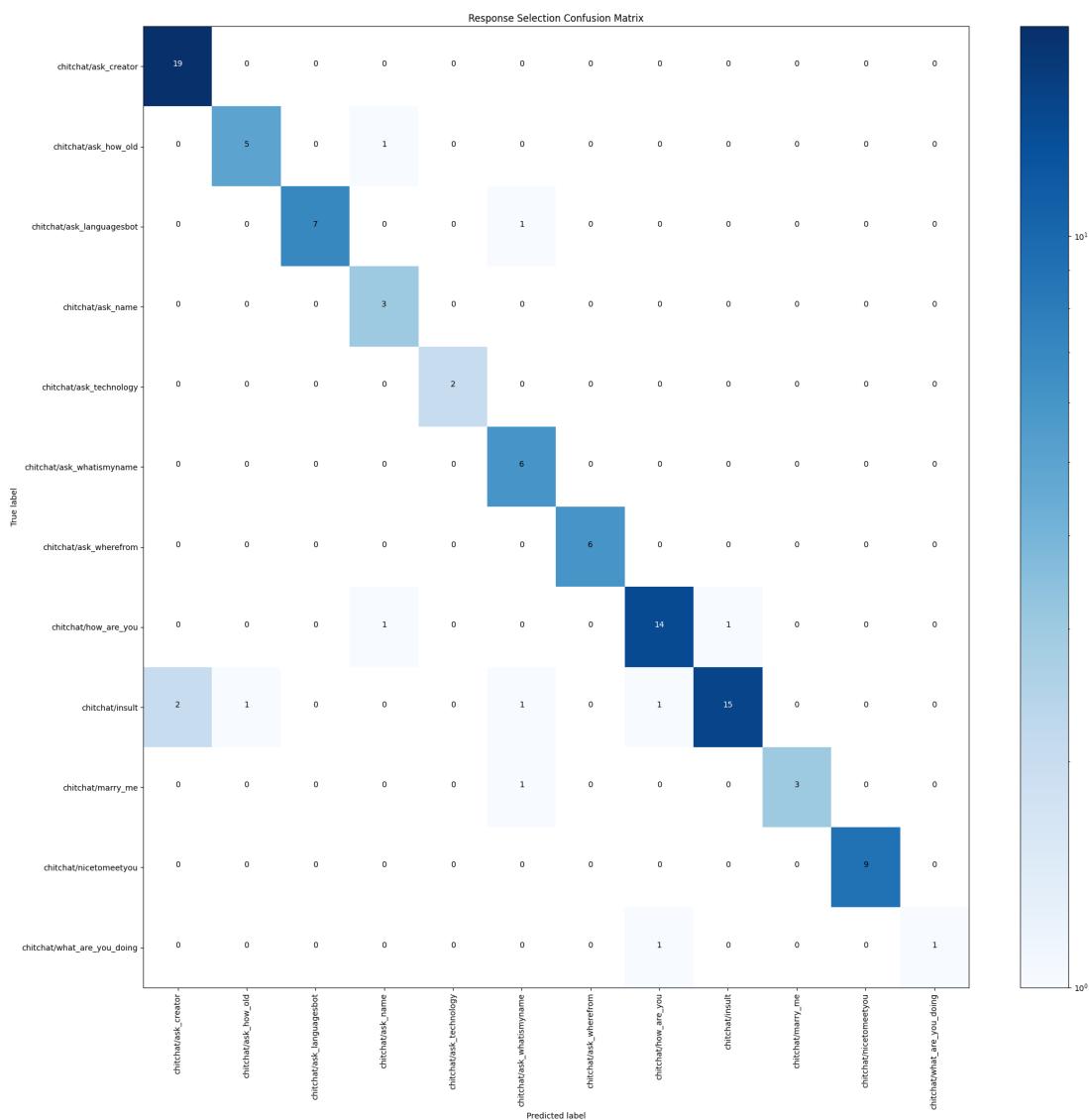


Figure 25: Konfusionsmatrix für chitchat Intents

I'm looking for a hospital

Word Tokenizer

I'm looking for a hospital

Figure 26: Tokens durch einen WhitespaceTokenizer [51]

der RegexFeaturizer gibt an, ob ein Wort von den Regular Expressions oder Lookup Tables erkannt wurde.[52, 51, 53]

Regular Expressions

Regular Expressions können verwendet werden, um Muster eines Strings zu beschreiben und können hier bei dem Beispiel von Chatbots verwendet werden, um Zahlen zu beschreiben, wie in Abbildung 27 zu sehen ist.[52, 51, 53]

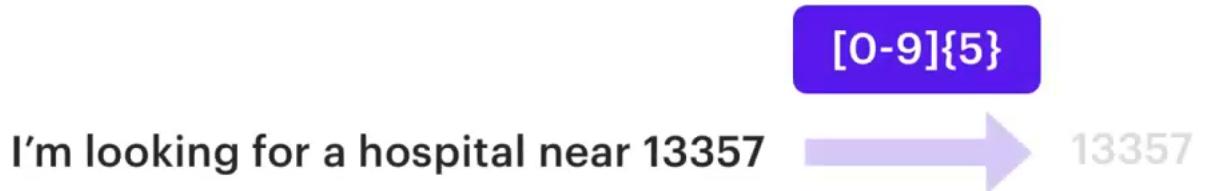


Figure 27: Regular Expression Beispiel [51]

Lookup Tables

Lookup Tables können verwendet werden, wenn eine Entity vordefinierte Werte haben kann. Zum Beispiel kann eine Entity mit dem Namen `country` 194 verschiedene Länder annehmen.[52, 51, 53]



Figure 28: Lookup Table Beispiel [51]

Der RegexFeaturizer muss vor dem EntityExtractor in der Pipeline platziert werden.[52, 51, 53]

4.2.7 CRFEntityExtractor

Eine Möglichkeit, um Entities 4.4.6 aus der Nachricht des Benutzers zu extrahieren ist über den CRFEntityExtractor.[54]

CRF steht hierbei für Conditional Random Field. Dieses Modell lernt, welche Komponenten eines Satzes Entities 4.4.6 sind und welche Entities 4.4.6 diese sind.[54, 51, 53]

Dies macht der CRFEntityExtractor, indem er die Sequenzen der Tokens 1 beobachtet. Es wird also ein Token aus dem Satz herausgepickt, und anschließend wird geschaut, ob

die Wörter danach und davor zum Kontext des Worts beitragen, um zu erkennen, ob es sich hierbei um eine Entity handelt. Dabei schaut der Extractor auf Eigenschaften, wie beispielsweise, ob das Wort groß- oder kleingeschrieben ist, ob es einen Prefix hat, ob es eine Zahl ist oder ob es ein speziell definiertes Wort für die Entities ist. Dies wird auch mit den Wörtern davor und danach gemacht.[54, 51, 53]

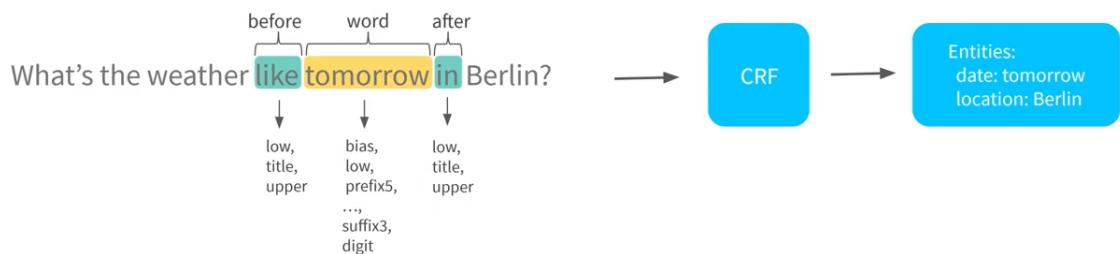


Figure 29: Funktionsweise vom CRFEntityExtractor [51]

Der CRFEntityExtractor produziert anschließend als Output, welche Wörter in einem Satz Entities sind und welche Entities diese sind, also was ihre Labels sind. Außerdem wird noch produziert, wie sicher das Modell war, dass diese Entities korrekt sind, also wird die sogenannte Confidence ausgegeben und welches Modell verwendet wurde.[54, 51, 53]

4.2.8 LexicalSyntacticFeaturizer

Ein Featurizer wird generell verwendet, um Features von den Tokens zu extrahieren. Diese können dann von dem Intent-Klassifikations-Modell genutzt werden, um Muster zu erkennen und anschließend den korrekten Intent 4.4.1 vorherzusagen.[55, 51, 56]

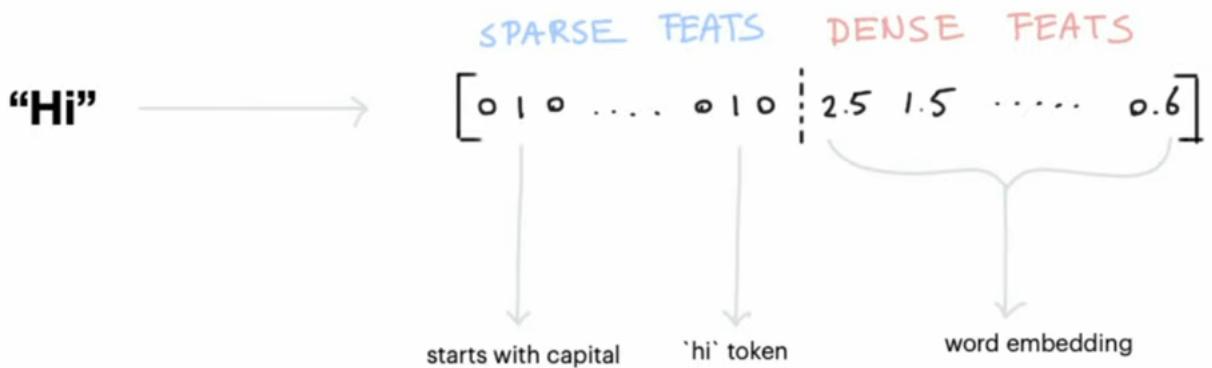


Figure 30: Funktionsweise von einem Featurizer [56]

4.2.9 CountVectorsFeaturizer

Der CountVectorsFeaturizer erstellt Bag-of-Words 2.2.11. Dabei wird gezählt, wie oft ein bestimmtes Wort aus den Trainingsdaten in der Nachricht des Benutzers vorkommt.[57, 56, 51, 58]

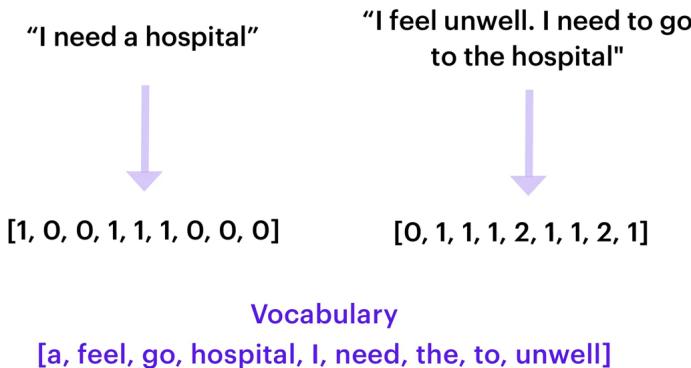


Figure 31: CountVectorsFeaturizer Beispiel [56]

Beim CountVectorsFeaturizer kann außerdem angeben werden, dass statt Wörtern sogenannte n-grams 2.2.12 verwendet werden sollen. Dies macht das Modell in der Regel robuster gegen Typos allerdings erhöht sich damit auch die Zeit, die das Modell zum Trainieren benötigt.[57, 56, 51, 58]

Listing 25: CountVectorsFeaturizer

```

1 pipeline:
2   - name: CountVectorsFeaturizer
3     analyzer: char_wb
4     min_ngram: 1
5     max_ngram: 4

```

4.2.10 DIETClassifier

DIET steht für **Dual Intent and Entity Transformer** und kann somit sowohl Intents 4.4.1 als auch Entities 4.4.6 klassifizieren und erkennen. Bei der Klassifizierung wird die Eingabe des Users hergenommen und dabei der passende Intent aus dem NLU.yml File gefunden. Ohne DIET würde zusätzlich zur Intent-Klassifizierung auch noch eine Komponente benötigt werden, die Entities extrahiert, wie beispielsweise den CRFEntityExtractor 4.2.7. Der Parameter `epochs` gibt dabei an, wie oft die Trainingsdaten durchgegangen werden sollen. Standardmäßig ist dieser auf den Wert 300 gesetzt. Je kleiner diese Nummer also ist, desto schneller wird das Modell trainiert.[59]

Listing 26: DietClassifier in Pipeline

```

1  pipeline:
2      - name: DIETClassifier
3      epochs: 100

```

Der Output des DIETClassifier kann wie in Listing 27 aussehen. Dabei wird zunächst der Intent angeführt, der mit der höchsten Wahrscheinlichkeit (Confidence) vorhergesagt wurde. Alle anderen Intents, die das Modell in Betracht gezogen hat, werden nach ihrer Confidence absteigend sortiert ausgegeben. Zusätzlich sind noch alle Entities, die erkannt wurden, mit deren Wert angeführt, inklusive der Stelle in der Eingabe des Users, an der die jeweiligen Entities erkannt wurden.

Listing 27: Output des DIETClassifiers [59]

```

1  {
2      "intent": {"name": "greet", "confidence": 0.8343},
3      "intent_ranking": [
4          {
5              "confidence": 0.385910906220309,
6              "name": "goodbye"
7          },
8          {
9              "confidence": 0.28161531595656784,
10             "name": "restaurant_search"
11         }
12     ],
13     "entities": [
14         {
15             "end": 53,
16             "entity": "time",
17             "start": 48,
18             "value": "2017-04-10T00:00:00.000+02:00",
19             "confidence": 1.0,
20             "extractor": "DIETClassifier"
21         }
22     }
23 }

```

4.2.11 EntitySynonymMapper

Der EntitySynonymMapper erwartet als Input einen Extractor von den verschiedenen Entity Extraktoren und liefert als Ausgabe modifizierte Werte für die Entities, die bereits vorher erkannt wurden. Wenn beim EntitySynonymMapper also eine Entity überliefert wird und in den Trainingsdaten Synonyme für diese Entity vorhanden sind, wird der Wert der Entity auf den Wert, der in den Trainingsdaten angegeben ist, gesetzt. Wenn also beispielsweise in einem Satz des Benutzers New York City oder NYC vorkommt, werden diese beide auf denselben Wert nyc gesetzt.[60]

Listing 28: Entity Synonym Mapper

```

1  [
2      {
3          "text": "I moved to New York City",
4          "intent": "inform_relocation",
5          "entities": [
6              {
7                  "value": "nyc",
8                  "start": 11,
9                  "end": 24,
10                 "entity": "city",
11             }
12         ]
13     }
14 ]

```

```

11     },
12     {
13         "text": "I got a new flat in NYC.",
14         "intent": "inform_relocation",
15         "entities": [
16             {
17                 "value": "nyc",
18                 "start": 20,
19                 "end": 23,
20                 "entity": "city",
21             }
22     ]
23 }
```

4.2.12 ResponseSelector

Ein Selector sagt die korrekte Response 4.4.2 voraus aus einer Menge von möglichen Responses. Beim ResponseSelector wird als Ausgabe ein Key-Value Paar ausgegeben. Als Key wird hierbei der Retrieval Intent ausgegeben und das Value ist die vorhergesagte Response und die Confidence, mit der diese vorhergesagt wurde.[61]

Listing 29: Response Classifier

```

1  {
2     "response_selector": [
3         "faq": [
4             "response": [
5                 {
6                     "id": 1388783286124361986,
7                     "confidence": 0.7,
8                     "intent_response_key": "chitchat/ask_weather",
9                     "responses": [
10                        {
11                            "text": "It's sunny in Berlin today",
12                            "image": "https://i.imgur.com/nGF1K8f.jpg"
13                        },
14                        {
15                            "text": "I think it's about to rain."
16                        }
17                    ],
18                    "utter_action": "utter_chitchat/ask_weather"
19                },
20                "ranking": [
21                    {
22                        "id": 1388783286124361986,
23                        "confidence": 0.7,
24                        "intent_response_key": "chitchat/ask_weather"
25                    },
26                    {
27                        "id": 1388783286124361986,
28                        "confidence": 0.3,
29                        "intent_response_key": "chitchat/ask_name"
30                    }
31                ]
32            }
33        ]
34    }
35 }
```

Bei einem Retrieval Intent handelt es sich um einen speziellen Intent, der weiter aufgeteilt werden kann in sogenannte Sub-Intents. Dies kann man zum Beispiel nutzen, wenn man einen Retrieval Intent für FAQ und für Chitchat hat, bei denen dann individuelle Frage einen Sub-Intent darstellt.[62] Verwendet wird dies, wenn der Bot zum Beispiel bei FAQ oder Chitchat sowieso immer mit derselben Antwort auf die Frage antworten soll, unabhängig von dem, was vorher geschrieben wurde.[63]

Bei der Arbeit wurde dieser Ansatz ebenfalls gewählt, wie in 30 zu sehen ist.

Listing 30: Response Selector für Chitchat und FAQ

```

34 - name: ResponseSelector
35   epochs: 100
36   retrieval_intent: faq
37 - name: ResponseSelector
38   epochs: 100
39   retrieval_intent: chitchat

```

4.2.13 FallbackClassifier

Der FallbackClassifier ist standardmäßig nicht in der Rasa Pipeline enthalten.[44]

Dieser wird verwendet, um mit Nachrichten umzugehen, bei denen nur eine sehr niedrige Confidence vorhergesagt wurde. In diesem Fall wird dann ein Intent mit dem Namen `nlu_fallback` vorhergesagt, welchen man dann behandeln kann, indem man beispielsweise als Antwort immer definiert, dass der User seine Nachricht neu formulieren soll. Die Confidence wird hierbei auf den Wert gesetzt, welcher im `Threshold` angegeben wird.[64, 65] Außerdem gibt es die Option einen sogenannten `ambiguity_threshold` anzugeben. Bei diesem wird der `nlu_fallback` Intent vorhergesagt, wenn die zwei als wahrscheinlichst empfundenen Intents sich um weniger als den `ambiguity_threshold` in der Confidence unterscheiden.[64]

Listing 31: Fallback Classifier

```

1 pipeline:
2 - name: FallbackClassifier
3   threshold: 0.7
4   ambiguity_threshold

```

4.3 Welche Rolle spielen neuronale Netze in Rasa

Rasa selbst definiert 5 Stufen von AI, wie in Abbildung 32 zu sehen ist..[66]

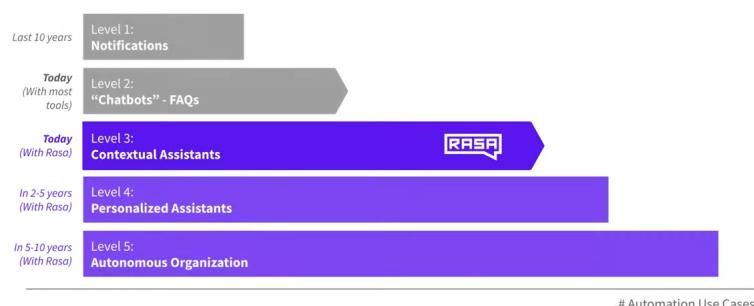


Figure 32: 5 Stufen von AI [67]

Level 1: Notifications

Auf Level 1 geht es darum, simple Aufgaben zu erfüllen, wie man sie möglicherweise von seinem Smartphone kennt. Darunter fällt zum Beispiel das Anlegen von Terminen und anderen Benachrichtigungen, die man dann zu einer eingestellten Uhrzeit bekommt.[68, 66, 67]

Level 2: "Chatbots" & FAQs

Auf Level 2 geht es darum, dass die Benutzerin oder der Benutzer einfache Fragen stellen kann und anschließend der Chatbot auf diese antwortet. Diese Art von Chatbots findet man sehr oft, allerdings sind diese sehr fehleranfällig, weil hier nur eine Menge an Regeln angelegt wird, an die sich der Chatbot hält. Dies wird sehr oft in Form von FAQs genutzt und auch ein paar kleine Follow-up Fragen können hierbei schon definiert sein.[68, 66, 67]

Level 3: Contextual AI Assistants

Dieses Level wird derzeit von Rasa unterstützt. Zusätzlich zu normalen FAQs wird hierbei auch der Kontext beachtet. Es macht nämlich einen Unterschied, wann, wie und in welcher Situation ein Benutzer etwas gesagt hat. Bei Contextual Assistants wird der Kontext also was bereits zuvor gesagt wurde, auch beachtet. Außerdem lenken sie Konversationen in die gewünschte Richtung und werden mit der Zeit immer besser, wenn man sie trainiert.[68, 66, 67]

4.4 Komponenten

In der sogenannten Domain werden alle Intents 4.4.1, Entities 4.4.6, Slots 4.4.5, Responses 4.4.2, Forms 4.4.7 und Actions 4.4.9 angegeben, die der Bot kennt. Diese ganzen Informationen befinden sich in der **domain.yml** Datei.[69]

4.4.1 Intents

Intents sind die Absichten hinter der Nachricht des Benutzers. Als Intents werden also alle möglichen Beispielsätze definiert, die ein Benutzer sagen könnte, um eine bestimmte Absicht auszudrücken.[70]

Intents werden in dem **nlu.yml** File wie in Listing 32 angegeben:

Listing 32: Intents Beispiel

```

1  ## intent:<name des intents>
2  - <phrase 1>
3  - <phrase 2>
4  - <phrase 3>
```

4.4.2 Responses

Responses sind die Antworten, die vom Bot gegeben werden, wenn ein bestimmter Intent erkannt wurde.[71]

Responses fügt man in der **domain.yml** Datei in der Form, wie in Listing 33 gezeigt, ein.

Listing 33: Responses Beispiel

```

1  responses:
2    utter_<name der response>:
3      - text: "<text>"
4      - image: "<img link>"
5      - buttons:
6        - title: "<button title>"
7        - payload: "<payload>"
8      ...
9
10 z.B.
11
12 utter_chitchat/what_are_you_doing:
13   - text: Ach nichts besonderes, nur ein wenig chatten!
14   - text: Ich entspanne gerade ein wenig aber durchl cher mich ruhig mit Fragen
     ich muss sowieso wieder an die Arbeit.
15   - text: Frag mich ruhig etwas ich brauche sowieso eine Ablenkung von all dem
     was ich eigentlich machen sollte!
16   buttons:
17     - title: "Wie geht es dir?"
18     - payload: "Wie geht es dir?"
19     - title: "Willst du mich heiraten?"
20     - payload: "Willst du mich heiraten?"
21     - title: "Welche Technologien wurden f r dich verwendet?"
22     - payload: "Welche Technologien wurden f r dich verwendet?"
23     - title: "Wie sp t ist es?"
24     - payload: "Wie sp t ist es?"
```

4.4.3 Stories

Stories werden als Trainingsdaten verwendet, die zum Trainieren des Modells des Bots verwendet werden. Stories können dabei genutzt werden, um Modelle zu trainieren, bei denen auch unvorhersehbare Konversationspfade behandelt werden und die sich unterscheiden von den Rules 4.4.4.[72]

Bei einer Story wird also die Unterhaltung zwischen einem Benutzer und dem Bot dargestellt. Dabei wird die Eingabe des Benutzers als Intent 4.4.1 angegeben und die Antwort, mit der der Bot antworten soll, als Name der Action 4.4.9.[72]

Stories können wie in 34 aufgezeigt aussehen und sind im **stories.yml** File anzugeben:

Listing 34: Stories Beispiel

```

1 stories:
2 - story: name der story
3 steps:
4 - intent: <name des intents>
5 - action: <name der action>
6
7 z.B.
8 stories:
9 - story: sad path 1
10 steps:
11 - intent: greet
12 - action: utter_greet
13 - intent: mood_unhappy
14 - action: utter_cheer_up
15 - action: utter_did_that_help
16 - intent: affirm
17 - action: utter_happy

```

Checkpoints und OR-Statements

Stories können außerdem mit Checkpoints und OR-Statements versehen werden. Bei diesen sollte aber grundsätzlich aufgepasst werden und diese sollten nur bedacht verwendet werden, weil in den meisten Fällen die gewünschten Resultate besser mit **Rules** 4.4.4 oder einem **ResponseSelector** 4.2.12 zu erzielen sind.[73]

Checkpoints können genutzt werden, um seine Trainingsdaten zu vereinfachen, indem ein Checkpoint in einer Story gesetzt wird und auf diesen in einer anderen Story wieder angesetzt wird. Von diesen sollte man allerdings nicht zu viele machen, weil sonst die Stories sehr leicht schwer zu lesen und unübersichtlich sind und außerdem die Trainingszeit dadurch erhöht wird.[74]

Ein Checkpoint wird am Ende der Story definiert, wenn dieser Teil der Story auch wieder als Voraussetzung für eine weitere Story gesetzt sein soll. In der nächsten Story wird dann mit dem Checkpoint, also dem Punkt, auf den angeknüpft wird, begonnen.

In Listing 35 werden Checkpoints von Stories verwendet, um an anderen Stories anzuknüpfen[74]:

Listing 35: Checkpoints Beispiel

```

1 stories:
2 - story: beginning_of_flow
3 steps:
4 - intent: greet
5 - action: action_ask_user_question
6 - checkpoint: check_asked_question
7
8 - story: handle_user_affirm
9 steps:
10 - checkpoint: check_asked_question
11 - intent: affirm
12 - action: action_handle_affirmation
13 - checkpoint: check_flow_finished
14
15 - story: handle_user_deny
16 steps:

```

```

17     - checkpoint: check_asked_question
18     - intent: deny
19     - action: action_handle_denial
20     - checkpoint: check_flow_finished
21
22   - story: finish flow
23   steps:
24     - checkpoint: check_flow_finished
25     - intent: goodbye
26     - action: utter_goodbye

```

OR-Statements können dafür verwendet werden, wenn man auf mehrere Intents innerhalb einer Story gleich reagieren möchte.[75]

Hierbei wird also anstelle von einem Intent in der Story ein **or** geschrieben und darunter werden alle Intents angegeben, von denen einer eintreffen muss, damit die Story zutrifft.

Listing 36: OR Beispiel

```

1  stories:
2  - story:
3    steps:
4      # ... vorherige schritte
5      - action: utter_ask_confirm
6      - or:
7        - intent: affirm
8        - intent: thankyou
9        - action: action_handle_affirmation

```

4.4.4 Rules

Rules werden angegeben, um kleine Teile von Unterhaltungen anzugeben, die immer wieder gleichbehandelt werden sollen. Diese sollten allerdings nicht allzu häufig verwendet werden, weil nie alle Konversationen vorhergesagt werden können. Um Rules verwenden zu können, muss die **RulePolicy** in der Policy Konfiguration eingetragen sein.[76]

Eine Rule wird wie in Listing 37 beschrieben in der Datei **rules.yml** angegeben.

Listing 37: Rules Beispiel

```

1  rules:
2
3  - rule: Say 'hello' whenever the user sends a message with intent 'greet'
4    steps:
5    - intent: greet
6    - action: utter_greet

```

4.4.5 Slots

Slots sind sozusagen das Gedächtnis des Bots. Diese sind als Key-Value-Paare dargestellt und können dazu verwendet werden, damit Information, die der Benutzer bereitstellt,

gespeichert werden können, ähnlich zu Entities. Diese Informationen können beispielsweise der Name des Benutzers sein oder Informationen, die für den generellen Kontext des Gesprächs wichtig sind.[77]

Listing 38: Slots Beispiel

```

1  slots:
2    slot_name: <slot name>
3      type: <type>
4
5 z.B.
6
7 slots:
8   name:
9     type: rasa.shared.core.slots.TextSlot
10    initial_value: null
11    auto_fill: true
12    influence_conversation: false

```

4.4.6 Entities

Entities sind strukturierte Stücke von Informationen, die sich innerhalb der Nachricht eines Benutzers befinden. Solche Entities können beispielsweise ein Ort, ein Beruf oder ein Name sein.[78]

Entities werden wie in Listing 39 in der Datei `domain.yml` angegeben.

Listing 39: Entities in der Domain

```

1  entities:
2    - <entity name>
3    - <entity name>
4
5 z.B.
6
7 entities:
8   - name
9   - branch
10  - teacher
11  - class
12  - grade
13  - subject

```

Diese Entities müssen dann noch in den Intents angegeben werden, in denen sie vorkommen sollen. Dies wird dadurch erreicht, dass folgende Syntax von Listing 40 bei den Trainingssätzen im `nlu.yml` File ergänzt wird.

Listing 40: Entities Beispiel

```

1 Hallo mein Name ist [Lukas](name).
2 Ich h tte gerne eine [kleine](size) [Pizza](meal)

```

Entity Roles

Entity Roles können sinnvoll in manchen Szenarien sein.

Listing 41: Entity Roles Beispiel

```
1 Buche einen Flug von [Linz](city) nach [London](city).
```

In dem Fall von 41 sind sowohl Linz als auch London zwar richtig gekennzeichnet als Entity mit dem Namen `city`, allerdings reicht diese Information noch nicht aus, damit der Chatbot richtig reagieren kann. Hierbei wäre es praktisch, wenn noch angegeben wird, welche dieser zwei Städte das Ziel und welche der Abflugort ist. Dies wird mit Entity Roles erreicht.[79]

Entity Groups

Entity Groups können genutzt werden, wenn Entities miteinander gruppiert werden sollen.[79]

Im Beispiel von 42 kann dies von Vorteil sein.

Listing 42: Entity Groups Beispiel

```
1 Ich h tte gerne eine kleine [Pizza](meal) mit [Pilzen](topping) und eine
[Salami](topping) [Pizza](meal).
```

Bei der Gruppe muss hier erkannt werden, welche zwei Entities zusammen gehören[79]:

Listing 43: Entity Groups Beispiel 2

```
1 Ich h tte gerne eine kleine [Pizza](meal) mit [Pilzen](topping) und eine
[Salami](topping) [Pizza](meal).
2 Group 1: [Pizza](meal) [Pilzen](topping)
3 Group 2: [Salami](topping) [Pizza](meal)
```

Um Entity Groups oder Entity Roles nutzen zu können, muss in der Pipeline entweder der CRFEntityExtractor 4.2.7 oder der DIETClassifier 4.2.10 verwendet werden. Diese sind die einzigen Entity Extraktoren, die Role und Group Labels erkennen können.[79]

4.4.7 Forms

Um mehrere Informationen von einem Benutzer zu bekommen, eignen sich Forms. Um Forms zu verwenden, muss die **RulePolicy** in der Policy Konfiguration eingetragen sein.[80]

Wenn ein Formular hinzugefügt werden soll, muss dies in der forms Sektion in dem `domain.yml` File angegeben sein.

Listing 44: Forms Beispiel

```

1   forms:
2     restaurant_form:
3       required_slots:
4         cuisine:
5           - type: from_entity
6             entity: cuisine
7       num_people:
8         - type: from_entity
9           entity: number

```

4.4.8 Synonyms

Mithilfe von Synonymen können extrahierten Entities einen anderen Wert annehmen, als sie eigentlich vorher hatten, wenn diese in der Bedeutung gleich sind. Wenn also mit verschiedenen Wörtern dasselbe gemeint ist, können Synonyme zur Hilfe genommen werden.[81]

Ein Beispiel dafür wäre 45 im **nlu.yml** File.

Listing 45: Synonym Beispiel

```

1   - synonym: Medientechnik
2     examples: |
3       - IT-Medientechnik
4       - IT Medientechnik
5       - Medientechnologie

```

4.4.9 Actions

Es gibt 2 verschiedene Arten von Messages:

1. **Static Messages:** Diese sind unabhängig vom User Input und benötigen keinen Action Server[82]
2. **Dynamic Messages:** Diese sind abhängig vom User Input und benötigen einen Action Server 4.4.10[82]

Der Rasa Action Server führt sogenannte Custom Actions 4.4.10 für einen Rasa Open Source Conversation Assistent aus.

Wenn der Assistant eine gewisse Custom Action vorhersagt, sendet der Rasa Server einen POST-Request an den Actionserver mit einer JSON Payload mit dem Namen der vorhergesagten Action, der Conversation ID, den Inhalten des Trackers und den Inhalten der Domain.[83]

4.4.10 Custom Actions

Neben normalen Responses können auch Custom Actions definiert werden. Diese sind in Python zu schreiben und dabei kann eigener Code für Berechnungen oder dergleichen verwendet werden. Hierbei muss eine Klasse implementiert werden, die Kind der Klasse `Action` ist und eine Methode `name` und `run` besitzt. Eine Custom Action wird beim Leobot beispielsweise für die Ausgabe des aktuellen Datums genutzt, wie bei Listing 46 zu sehen ist.

Listing 46: Custom Action für die Ausgabe des aktuellen Datums

```

1  class ActionWhatDateIsIt(Action):
2      def name(self) -> Text:
3          return "action_what_date_is_it"
4
5      def run(self, dispatcher: CollectingDispatcher,
6              tracker: Tracker,
7              domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
8
9          now = datetime.now()
10         weekday = datetime.today().weekday()
11         weekday_string = ("Montag", "Dienstag", "Mittwoch", "Donnerstag",
12                           "Freitag", "Samstag", "Sonntag")[weekday]
13
14         current_date = now.strftime("%d.%m.%Y")
15         print("Current Date =", current_date)
16
17         dispatcher.utter_message(text=f"Heute ist {weekday_string} der
18                                   {current_date}!")
19
20     return []

```

Nun muss die Custom Action in der `domain.yml` Datei definiert werden, wie in Listing 47 zu sehen ist.

Listing 47: Custom Actions in Domain definiert

```

1  actions:
2      - action_what_date_is_it

```

4.5 Initialisieren

4.5.1 Rasa Init

Ein neuer Rasa Assistant kann mithilfe des Init-Befehls in der Konsole erstellt werden. Der Befehl dafür lautet, wie in Listing 48 zu sehen ist.

Listing 48: Befehl fürs Initialisieren

```
1 rasa init
```

Bei diesem Befehl wird man gefragt, ob man direkt ein voreingestelltes Modell trainieren möchte. Dieses Modell wird basierend auf den Demodaten, die von jedem neu erstellten Rasa Projekt zur Verfügung gestellt werden, trainiert.

```
(rasa_env) lukas@lukas-IdeaPad-5-15ALC05:~/Documents/Schule$ rasa init
Welcome to Rasa! 🤖

To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa.
Now let's start! ⏪

? Please enter a path where the project will be created [default: current directory] chatbot-demo
? Path 'chatbot-demo' does not exist. Create path? Yes
Created project directory at '/home/lukas/Documents/Schule/chatbot-demo'.
Finished creating project structure.
? Do you want to train an initial model? 💪 (Y/n) 🚧
```

Figure 33: Rasa Init Konsolenausgabe

Nachdem dieser Befehl ausgeführt wurde, wird eine Struktur, wie in Abbildung 34 erzeugt.

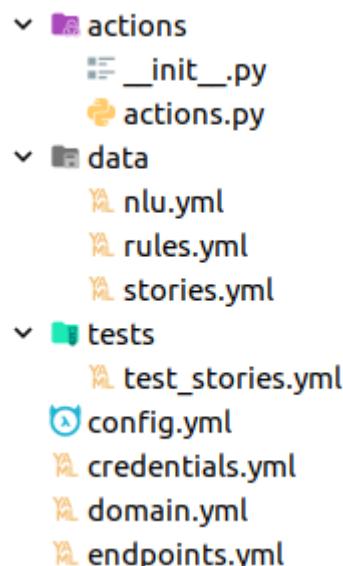


Figure 34: File Tree nach dem Initialisieren

4.6 Trainieren

4.6.1 Rasa Train

Um das Modell nun zu trainieren, wird der `Rasa train` Befehl benutzt. Dabei werden nur die Komponenten (Core 4.1.2 oder NLU 4.1.4) trainiert, die sich auch geändert haben im Vergleich zu dem letzten Modell. Es kann aber auch selbstständig angeben werden, dass nur `core` oder `nlu` trainieren werden soll. Nach dem Trainieren wird das Modell in den Pfad gespeichert, der beim `-out` definiert wurde und standardmäßig

/models ist. Dabei wird als Modellname der aktuelle Zeitstempel gewählt, außer es wird die `-fixed-model-name` Flag verwendet. Wenn andere Pipelines oder Policies verwendet werden sollen, kann auch ein anderes config-File angegeben werden, in dem diese gespeichert sind. Dies wird mit `-c` erreicht. Ab Rasa 2.2 gibt es außerdem die Möglichkeit eine Feinabstimmung an einem bereits bestehenden Modell vorzunehmen. Dies macht man, wenn man neue Trainingsdaten bei bereits bestehenden Intents hinzugefügt hat. Hierbei ergibt sich eine viel kürzere Trainingszeit, als wenn das Modell noch einmal von vorne trainiert würde und dabei wird der Pfad zum Modell angegeben, wenn nicht, wie standardmäßig definiert, das neueste trainiert werden soll.[84]

Listing 49: Rasa Train Befehle

```

1 rasa train
2 rasa train -c config.yml #Das verwendete Config File (default: config.yml)
3 rasa train --out trained-models/ #Der Pfad, in dem das Modell gespeichert wird
  (default: models/)
4 rasa train --fixed-model-name leobot-model #Der Name des Modells (default:
<timestamp>.tar.gz)
5 rasa train core #Rasa Core mit den Stories trainieren
6 rasa train nlu #Rasa NLU mit den NLU-Daten trainieren
7 rasa train --finetune models/20220211-094939.tar.gz #Finetune eines bestehenden
  Modells

```

4.7 Interagieren

4.7.1 Rasa Shell

Wenn man sich nun mit seinem Bot unterhalten möchte, kann man eine ChatSession über die Konsole mit dem `rasa train` Befehl. Normalerweise wird dabei das aktuellste Modell genommen. Allerdings kann auch mit der `-model` Flag auch der Pfad zum richtigen Modell angegeben werden.

Listing 50: Rasa Shell Befehle

```

1 rasa shell
2 rasa shell --model models/20220211-094939.tar.gz #Das Modell, das verwendet werden
  soll
3 rasa shell --debug #Debug-Modus aktivieren

```

Für detailliertere Ausgaben kann außerdem die `-debug` Flag verwendet werden. Die Ausgabe dabei sieht wie in Abbildung 35 aus:

4.7.2 Rasa Interactive

Um eine interaktive Session mit dem Bot zu starten, kann der Befehl `rasa interactive` genutzt werden. Dabei wird zunächst ein Modell trainiert und anschließend startet eine

```

2022-03-30 17:49:47 DEBUG rasa.core.policies.ensemble - Predicted next action using policy_0_MemoizationPolicy.
2022-03-30 17:49:47 DEBUG rasa.core.processor - Predicted next action 'utter_htl_age' with confidence 1.00.
2022-03-30 17:49:47 DEBUG rasa.core.processor - Policy prediction ended with events '[<rasa.shared.core.events.DefinePrevUserUtteredFeaturization object at 0x7f89ade8ed30>]'.
2022-03-30 17:49:47 DEBUG rasa.core.processor - Action 'utter_htl_age' ended with events '[BotUttered('Die HTL Leonding wurde 1984 gegründet und der Zweig für EDV und Organisation entstand 1985.', {"elements": null, "quick_replies": null, "buttons": [{"title": "Wie schwer ist die HTL?", "payload": "/ask_htl_difficulty"}, {"title": "Wie lange dauert die HTL?", "payload": "/htl_duration"}, {"title": "Wie komme ich zur HTL?", "payload": "/public_transport_connection"}, {"title": "Zeig mir ein Bild der HTL", "payload": "/show_school_picture"}], "attachment": null, "image": null, "custom": null}, {"utter_acti
on": "utter_htl_age"}, 1648655387.435862)]'.

```

Figure 35: Rasa Shell Ausgabe mit Debug Flag

interaktive Shell Session. Dabei hat die Benutzerin oder der Benutzer die Möglichkeit, alle vorhergesagten Aktionen vom Assistant zu korrigieren. Wenn nur das Core Model getestet werden soll, kann dies spezifiziert werden.

Listing 51: Interaktive Trainingssession starten

```

1 rasa interactive
2 rasa interactive --model models/20220211-094939.tar.gz #Das Modell, das verwendet
   werden soll
3 rasa interactive core #Interaktive Session f r das Core Modell starten

```

Bei dieser Trainingssession hat man die Möglichkeit, jedes Mal anzugeben, ob der vorhergesagte Intent und die daraus resultierende Antwort die Richtige sind. Außerdem wird die gesamte Chat History visualisiert und ausgegeben, wie in Abbildung 36 zu sehen ist.

```

? The bot wants to run 'utter_greet', correct? Yes
-----
Chat History

#      Bot                                     You
-----+
1      action_listen
-----+
2          Gibt es M dchen an der HTL?
                  intent: greet 1.00
-----+
3      ... 0.00
      action_listen 0.99
-----+
4          Hallo
                  intent: greet 1.00
-----+
5      utter_greet 0.96
      Hey! How are you?

```

Figure 36: Ausgabe beim Interactive Befehl

5 Implementierung

5.1 Systemarchitektur

Es gibt 3 große Services, das Chat Widget 5.3 auf der Schulhomepage, das Dashboard 5.4 und das Backend 5.2. Das System, das im Zuge der Arbeit entstand, sieht dabei folgendermaßen aus:

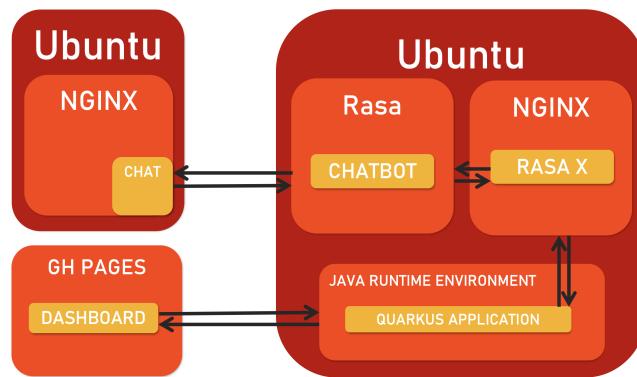


Figure 37: Systemarchitektur

Es gibt zwei virtuelle Ubuntu Maschinen, nämlich eine für den Chat und eine für Rasa und das Backend. Dadurch, dass der Chat auf die Seite der HTL Leonding Seite kommt, wird dadurch nur noch eine VM benötigt.

Der Chat wird dabei gerade mit NGINX gehostet. Dieser Server kommuniziert mit Rasa direkt über REST, wenn die Benutzerin oder der Benutzer eine Nachricht sendet, wird diese an Rasa gesendet und Rasa antwortet mit der passenden Antwort. Das Dashboard 5.4 wird auf GH Pages gehostet, dieses kommuniziert mit dem Backend über REST. Das Backend authentisiert sich dabei bei Rasa X 3.3.1 und holt sich alle Konversationen. Schließlich bereitet es die Konversationen auf und sendet sie an das Dashboard.

5.2 Backend

Das Quarkus 3.2.2 Backend hat die Aufgabe, mit Rasa X zu kommunizieren und die Konversationsdaten für das Dashboard aufzubereiten und zu senden.

Dieses Backend besitzt folgende Endpoints:

- GET /api/conversations
- GET /api/conversations/id
- POST /api/feedback
- GET /api/feedback
- GET /api/file/filename
- PUT /api/file/filename

Bei den beiden `conversations` Endpoints muss man sich zunächst authentisieren und sich einen Bearer Token von Rasa X holen, damit auf die anderen Rasa X Endpoints zugegriffen werden kann.

5.2.1 GET /api/conversations

Der Endpoint ruft zuerst die `getAuth()` Funktion auf, um den Bearer Token zu erhalten.

Danach wird der Rasa X GET Endpoint “/api/conversations”, mit dem Bearer Token im Header aufgerufen. Von diesem Endpoint wird ein JSON Objekt zurückgegeben, das die Konversationen enthält, aber zusätzlich noch viele andere irrelevante Daten, deshalb filtert sich das Backend wirklich nur ID des Senders, die Zeit und die Anzahl von Nachrichten der Unterhaltungen heraus und gibt diese zurück.

5.2.2 GET /api/conversations/id

Der Endpoint `/api/conversations/id` ruft zuerst die `getAuth()` Funktion auf, um den Bearer Token zu erhalten.

Um nur die Nachrichten einer gezielten Unterhaltung zu erhalten, wird der Rasa X Endpoint `/api/conversations/id/messages`, mit dem Bearer Token im Header aufgerufen. Die Response wird auch in dieser Form schon zurückgegeben.

5.2.3 POST /api/feedback

Der Endpoint `/api/feedback` erhält ein JSON Objekt mit den Daten des Feedback-Formulars und speichert diese in die Datenbank.

5.2.4 GET /api/feedback

Der Endpoint `/api/feedback` liest alle Feedbacks aus der Datenbank aus und gibt diese zurück.

5.2.5 GET /api/file/filename

Der Endpoint `/api/file/filename` liest die im URL angegeben Datei aus dem Filesystem aus und gibt diese zurück. Die möglichen Dateinamen sind:

- `nlu.yml`
- `rules.yml`
- `stories.yml`
- `config.yml`
- `domain.yml`

5.2.6 PUT /api/file/filename

Der Endpoint `/api/file/filename` erhält den Inhalt aus dem File, welches auch im URL angegeben wurde und dieses dann im Filesystem überschreibt.

5.3 Chat Widget

Der Chatbot der HTL Leonding sollte auf der Schulhomepage als Chatblase angezeigt werden und verschiedene Elemente, wie Buttons und Links unterstützen.

5.3.1 Konzept

Während den Anfängen der vorliegenden Arbeit wurde ein Konzept erstellt, um das Design des Chatbots festzulegen. Lange Zeit wurde der Chatbot, während der Entwicklung, unter dem Namen "Leon" geführt. Dies wurde jedoch im späteren Verlauf geändert und "Leon" wurde Teil des langjährigen "Leonie Projektes" der HTL Leonding.

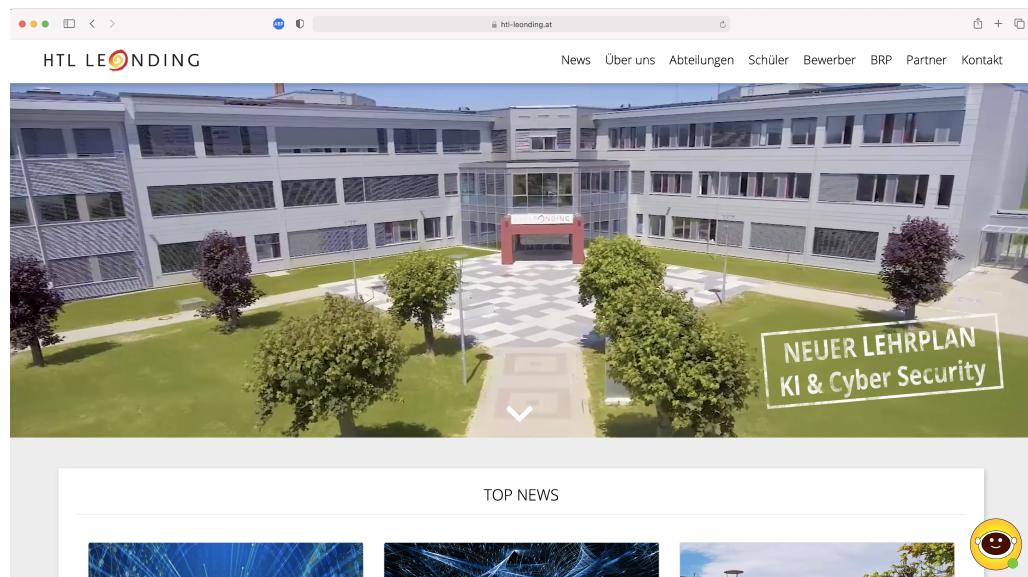


Figure 38: Konzept Chatbot geschlossen

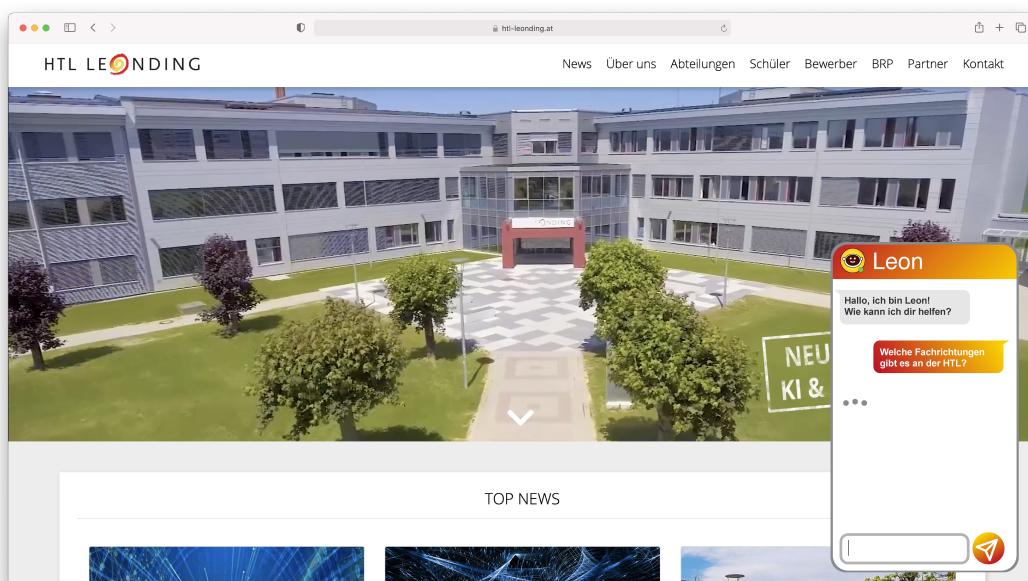


Figure 39: Konzept Chatbot geöffnet

5.3.2 Umsetzung

Umgesetzt wurde das Frontend mithilfe von Angular. Die Chatblase ist eine eigene Komponente, die durch CSS immer rechts unten fixiert ist. Die Farben des Chatbots sollten dabei an die HTL Leonding erinnern, weshalb ein Farbverlauf aus Farben des HTL-Logos erstellt wurde.

Jedoch begann der Chatbot ganz anders und zu Beginn wurde der Bot als ganze Seite entwickelt und nicht nur als Chatblase. Dies ist auf der Abbildung 40 zu sehen.

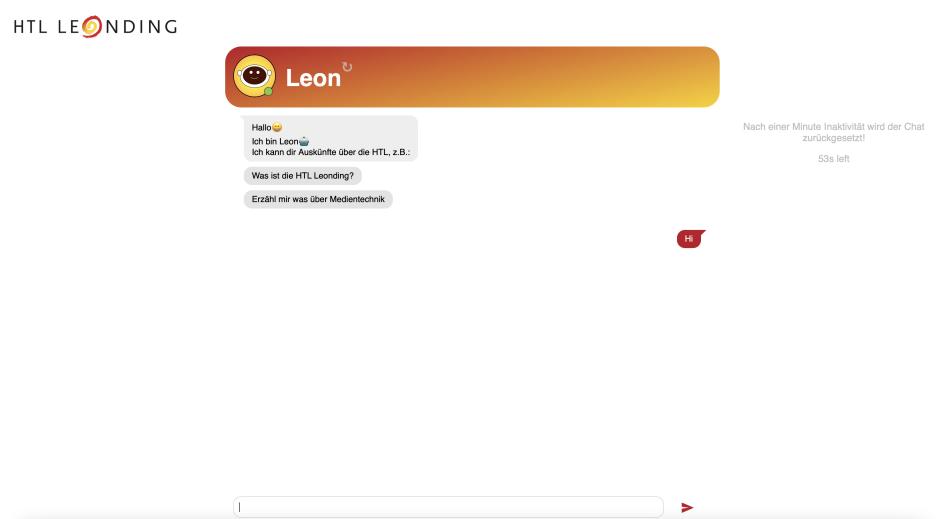


Figure 40: Chatbot auf einer ganzen Seite

Dabei war dies nicht das endgültige Ziel und so kam es dazu, dass der Bot schnell zur Chatblase umgewandelt wurde. Zum Testen war die HTL Leonding Seite mithilfe eines IFrame eingebunden und zusätzlich die Chatblasen Komponente.

Um das Gespräch in eine Richtung zu lenken, wurden Buttons eingeführt, die nach fast jeder Antwort mögliche Folgefragen vorschlagen. Solche Vorschläge sind auf der Abbildung 41 zu sehen.

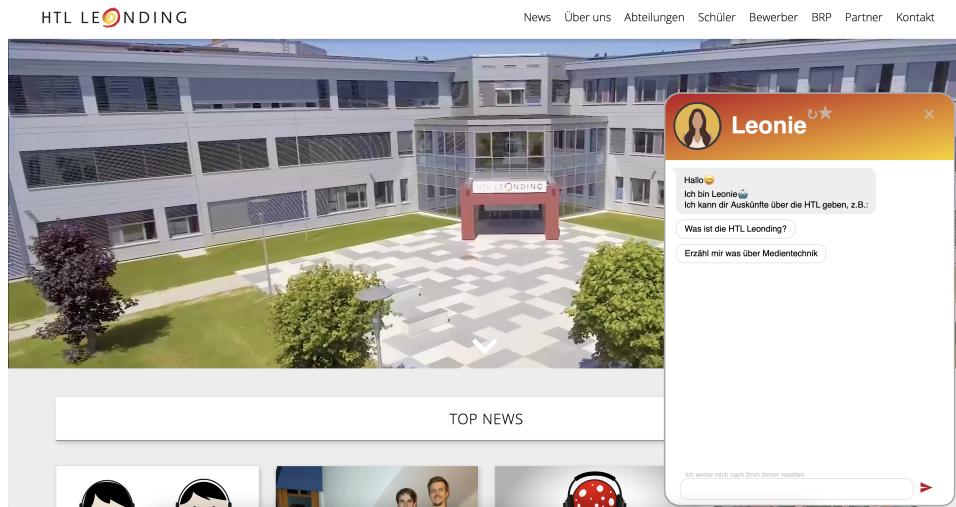


Figure 41: Chatbot

Die Buttons werden von Rasa mit der Antwort auf die Frage mitgeschickt. Wenn eine Benutzerin oder ein Benutzer auf einen der Buttons drückt, wird nicht der Text des Buttons an Rasa geschickt, sondern direkt die Bezeichnung des Intents mit einem "/" davor, da Rasa dies auch erkennt.

Um Bewertungen von echten Benutzern einzuholen, wurde außerdem eine Feedback-Seite eingeführt. Auf dieser kann die Benutzerin oder der Benutzer eine Bewertung zwischen 1 und 5 Sternen abgeben und auch einen Text absenden.

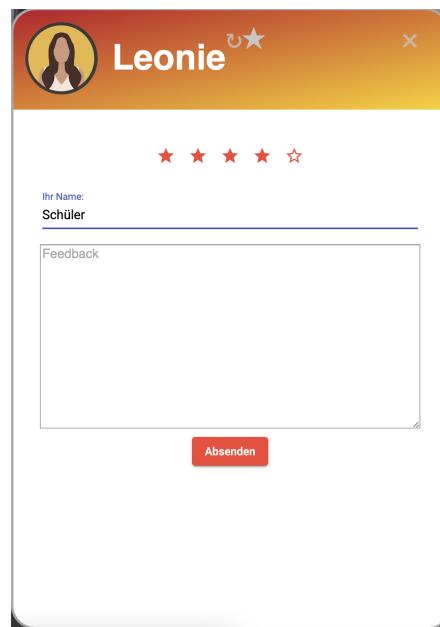


Figure 42: Feedback Fenster

Das Chatfenster wurde in wesentliche 3 Bereiche geteilt, die in Abbildung 43 zu sehen sind.



Figure 43: Aufbau vom Chat Fenster

Avatar

Wie bereits erwähnt, existiert der virtuelle Avatar der Leonie in Form einer 3D Pyramide und einer Website mit einem 3D Modell. Um Verwirrung zu vermeiden, wurde entschieden, dass der Chatbot auch in “Leonie” umbenannt werden soll. Bei der ursprünglichen Leonie handelt es sich um ein 3D Model (Abbildung 44). Da dies aber nicht ganz zu einem Profilbild in einem Chat passen würde, wurde mit “Adobe Illustrator” eine 2D-Grafik erstellt. Diese besteht aus einer farbigen Silhouette, welche die 3D-Leonie darstellt (Abbildung 45).



Figure 44: 3D Leonie

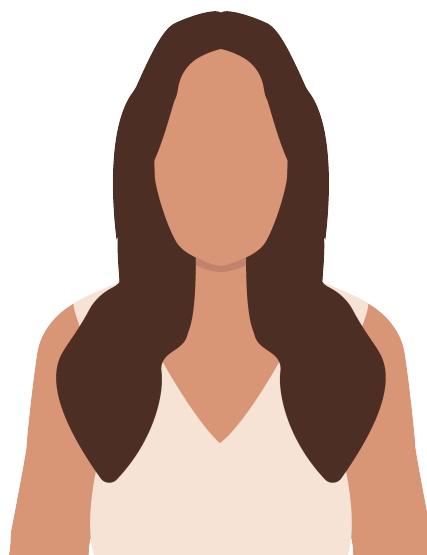


Figure 45: 2D Leonie

5.4 Dashboard

Um alle Gespräche, sowie die Bewertungen der Benutzerinnen und Benutzer anzuzeigen, wurde ein Dashboard eingeführt, wo nur diese Funktionalitäten gegeben sind. Im Laufe der Arbeit wurde dieses dann erweitert, um für den Leobot Conversation Cycle 5.4.2 als Seite zu dienen.

5.4.1 Conversation-Driven Development

Rasa empfiehlt für die Entwicklung von Chatbots “Conversation-Driven Development”, kurz CDD.[85] CDD beschreibt dabei eine Entwicklungsstrategie, in der der Chatbot nach Konversationen mit echten Usern verbessert wird.

Bei CDD wird empfohlen, dass man seinen Chatbot möglichst früh Personen zur Verfügung stehlt und so beobachten kann, welche Intents die echten Benutzerinnen und Benutzer erwarten und wie sie ihre Anfragen formulieren. Als Entwicklerin oder Entwickler kann man diese nun bei Bedarf hinzufügen.

5.4.2 Leobot Conversation Cycle



Figure 46: Leobot Conversation Cycle

CDD 5.4.1 wird mithilfe des Leobot Conversation Cycle umgesetzt. Der Leobot Conversation Cycle ist ein Workflow, welcher dazu dient, den Leobot zu überprüfen und zu erweitern. Der Workflow besteht dabei aus folgenden Schritten:

- Unterhalten
- Überprüfen
- Verbessern
- Trainieren

Diese 4 Schritte werden immer wieder durchgeführt, um den Chatbot dauerhaft zu verbessern.

Unterhalten

Das Unterhalten ist der erste Schritt des Leobot Conversation Cycles. In diesem Schritt werden dem Bot Fragen über die HTL Leonding und deren Produktangebot gestellt, die er anschließend beantworten soll. Dies passiert während Unterhaltungen im Chat auf der Schulhomepage mit echten Personen. Diese Unterhaltungen werden schließlich alle gespeichert.

Überprüfen

Das Überprüfen stellt den zweiten Schritt des Leobot Conversation Cycles dar. In diesem Schritt wird vom Schulpersonal oder Verantwortlichen überprüft, ob der Bot die Fragen richtig beantwortet und erkannt hat oder ob Verbesserungspotenzial besteht

Verbessern

Das Verbessern ist der dritte Schritt des Leobot Conversation Cycles. In diesem Schritt wird der Bot verbessert, das bedeutet, dass versucht wird, dass die Fehler verbessert werden. Je nachdem, wo der Fehler des Bots liegt, muss hier anders gearbeitet werden. Wenn es sich um einen eigentlich bekannten Intent handelt, jedoch die Formulierung des Benutzers nicht erkannt wurde, wird die Eingabe des Users zu den Trainingsdaten hinzugefügt.

Falls jemand aber eine Frage stellt, zu der es noch keinen passenden Intent gibt, so wird ein ganz neuer Intent zu den Trainingsdaten hinzugefügt.

Trainieren

Das Trainieren ist der vierte Schritt des Leobot Conversation Cycles. In diesem Schritt werden die Trainingsdaten an den Bot übergeben und dieser wird trainiert. Nachdem er fertig trainiert wurde, wechselt er auf das neu trainierte Model.

Und nun beginnt der Leobot Conversation Cycle wieder von vorne.

Regelkreis

Dieser Vorgang erinnert dabei stark an einen Regelkreis. Ein solcher Regelkreis ist in Abbildung 47 zu sehen und dieser führt eine Regelgröße auf einen gewünschten Sollwert. Die Hauptkomponenten des Regelkreises sind der Regler und die Regelstrecke. Beispielhaft ist die Regelstrecke ein Auto und die Fahrerin oder der Fahrer der Regler. Dabei nimmt die Lenkerin oder der Lenker gewisse Parameter zur Hilfe, durch die entschieden wird, ob das Auto gelenkt, gebremst oder beschleunigt werden soll. Diese Signale sind beispielsweise die aktuelle Geschwindigkeit, die Position des Autos oder die Fahrbahnverhältnisse. Beim Regelkreis wird also eine Regelgröße über ein Eingangssignal der Regelstrecke zurückgeführt und auf einen gewünschten Wert gebracht. Der Regelkreis wird oft bei der Regelung von Heizungen eingesetzt [86, 87]

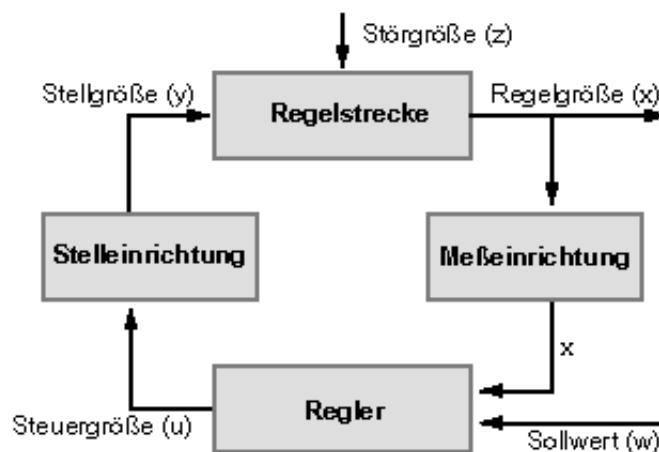


Figure 47: Allgemeiner Regelkreis [86]

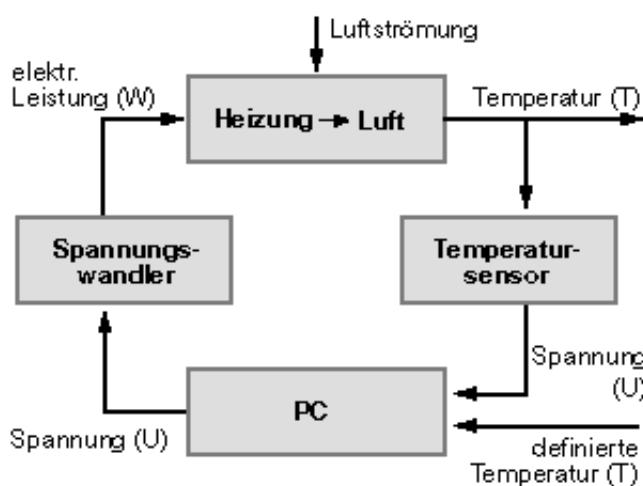


Figure 48: Regelkreis am Beispiel einer Heizung [86]

Ein Regelkreis wird dabei entweder durch eine Änderung der Sollgröße oder durch Auftreten einer Störung ausgelöst. Beim Leobot Conversation Cycle wird der Regelkreis

demzufolge ausgelöst, wenn eine Interaktion mit einem User stattfand, bei der der Bot eine falsche Antwort gegeben hat. Die Messeinrichtung ist dabei das Dashboard 5.4, das alle Unterhaltungen auflistet und diese grafisch veranschaulicht. Durch den eingebauten Editor ist die Möglichkeit gegeben, dass direkt die Files bearbeitet werden, sodass der Bot in Zukunft auch auf diese Fragestellung die richtige Antwort parat hat. Der Sollwert wird also in den Regler eingegeben und anschließend wird das Modell basierend auf den neuen Dateien neu trainiert. Dies stellt dabei die Stelleinrichtung des Regelkreises dar und der Kreislauf kann von vorne beginnen, sollte erneut eine Störgröße auftreten oder die Sollgröße verändert werden.

5.4.3 Warum das Dashboard

Die Oberfläche von Rasa X bietet Möglichkeiten Sachen zu verändern, die man als normale Verwaltungsperson nicht unbedingt benötigt, wie zum Beispiel Zugriff auf die Pipeline, Git und die ganzen trainierten Modelle. Das sind alles Funktionen, die man als Verwaltungsperson nicht benötigt, man sollte wirklich nur das sehen, was man auch verändern muss.

5.4.4 Umsetzung

Authentifizierung

Da nicht jeder auf das “Gehirn” des Bots zugreifen soll, ist das Dashboard Benutzer- und Passwortgeschützt. Außerdem soll in der Zukunft die Möglichkeit offen sein, dass manche Userrollen nur auf gewisse Inhalte Zugriff haben.

Die Authentifizierung wird über das Backend mit property file based authentication [88] durchgeführt.

Man kann sich alle vergangenen Unterhaltungen ansehen, sowie die nlu.yml, stories.yml, rules.yml, domain.yml Dateien direkt im Monaco Editor bearbeiten und speichern.

Die Unterhaltungen kann man zur Übersicht auch nach Datum filtern, wie in Abbildung 51 zu sehen ist.

Im Editor wurden auch Code-Vorschläge benutzt um das Einarbeiten von neuen Inhalten für die Benutzerinnen und Benutzer zu vereinfachen.

Dabei wurde sich für einen Editor entschieden, weil die Implementierung eines eigenen Formulars zu aufwendig im Zusammenhang für die Diplomarbeit gewesen wäre.

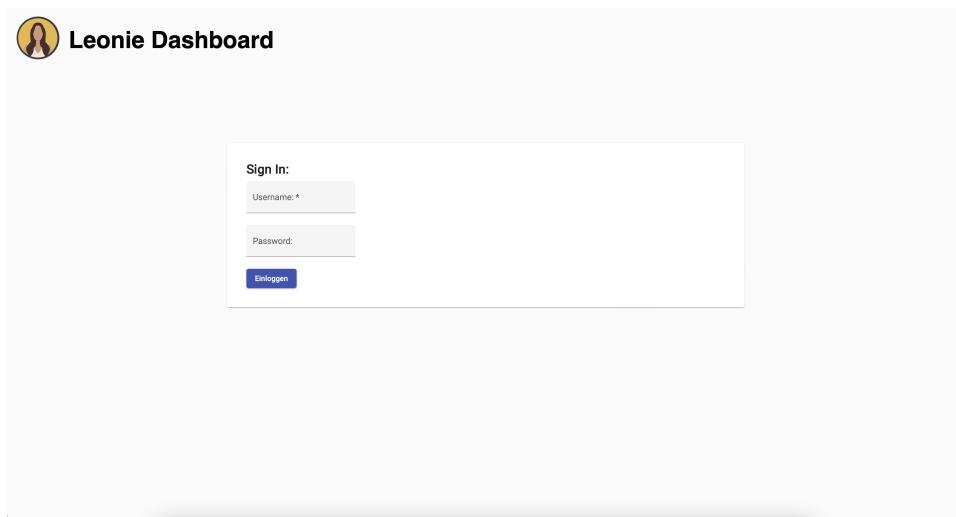


Figure 49: Anmeldefenster

Overview:
0:0 Nachrichten schreiben die Louts im Durchschnitt und insgesamt wurden in 521 Unterhaltungen bereits 1155 Nachrichten geschrieben.
Falls es irgendwelche Probleme geben sollte kannst du gerne einen Issue auf [GitHub](#) hinzufügen

Feedbacks:

Kategorie	Anzahl	Inhalt
Schüler	5/5	Der Bot hat mir sehr geholfen, ich denke ich werde mich für den Medientechnik Zweig anmelden

Figure 50: Dashboard

Zeitraum der Unterhaltungen wählen: 3/1/2022 – 3/2/2022

ID	Datum	Anzahl Nachrichten
FE-S-1646129395067	1. Mar. 2022 11:09	4
FE-S-1646129382971	1. Mar. 2022 11:09	1
FE-S-1646128446792	1. Mar. 2022 11:09	1
172.225.38.77+FE-S-1646128380143	1. Mar. 2022 10:53	6
47ad6181658049d193665a82cb8b910f	1. Mar. 2022 10:52	1

Figure 51: Dashboard nach Datum gefiltert

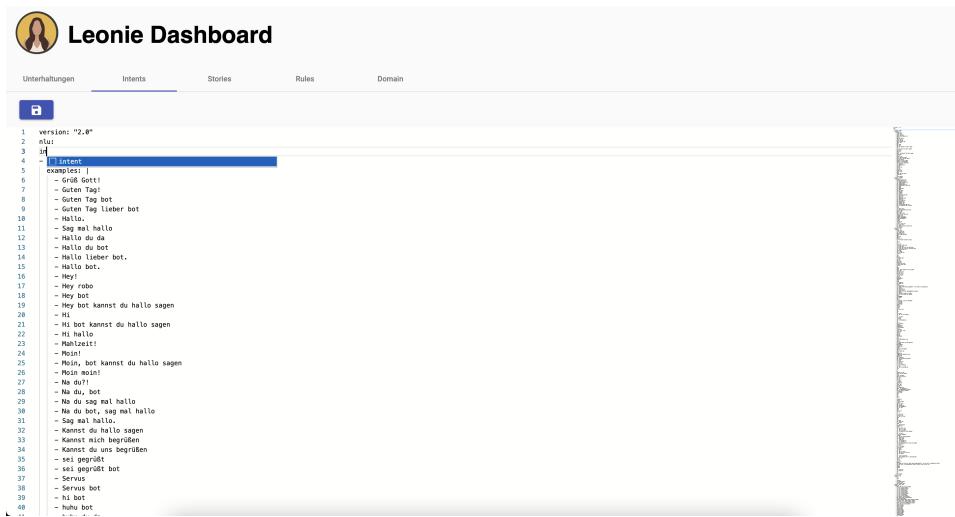


Figure 52: Vorschläge

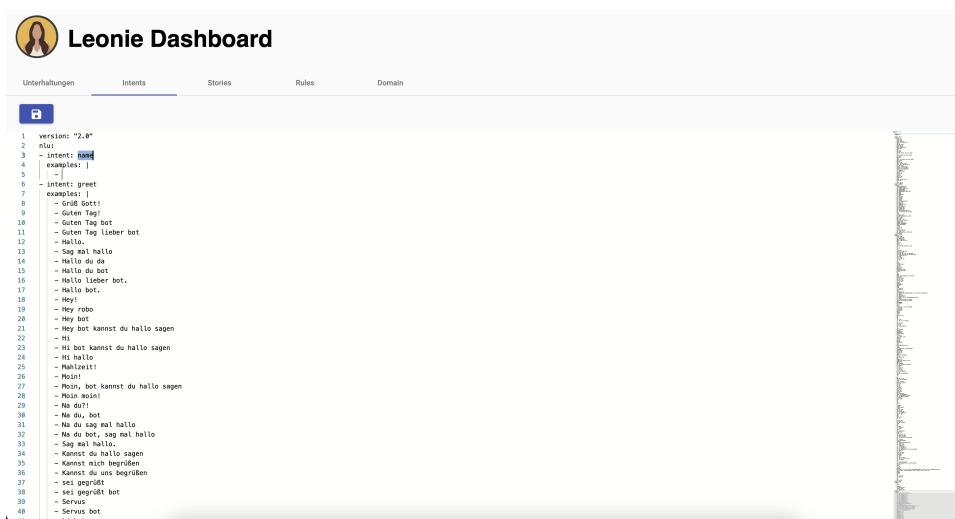


Figure 53: Vorschlag benutzt

5.5 Einbindung in Wordpress

Die HTML Leonding Website ist eine Wordpress Seite, deshalb musste der Chatbot als Webkomponente mit Angular Elements 3.2.4 exportiert werden, dass er dort eingebunden werden kann.

Auf WordPress wurde das Plugin “Shortcoder” [89] installiert, dieses ermöglicht es Code Teile zu speichern. Im Shortcoder Menü wurde ein neuer Shortcode mit der von uns implementierten Webkomponente erstellt.

Auf der WordPress Seite wird nun der Shortcode eingefügt und somit der Chatbot eingebunden.

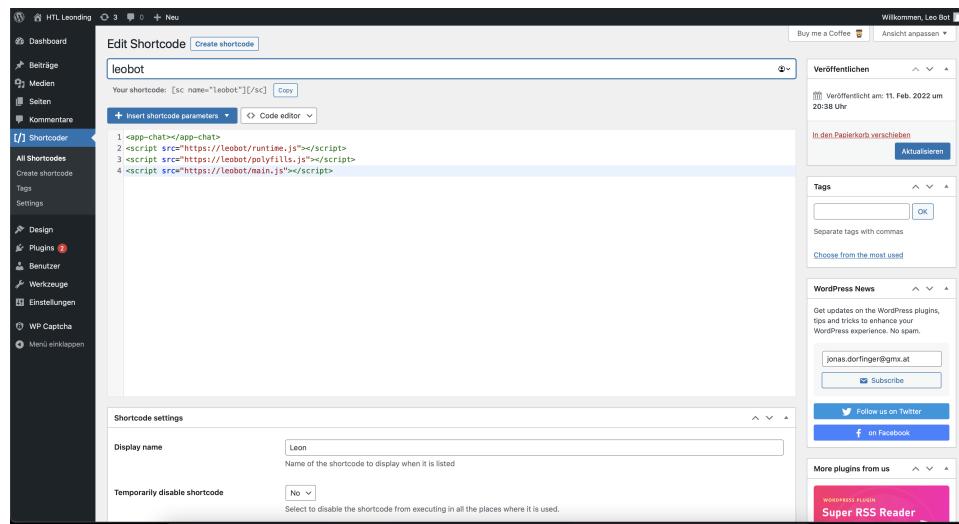


Figure 54: Shortcoder in WordPress Plugin Manager

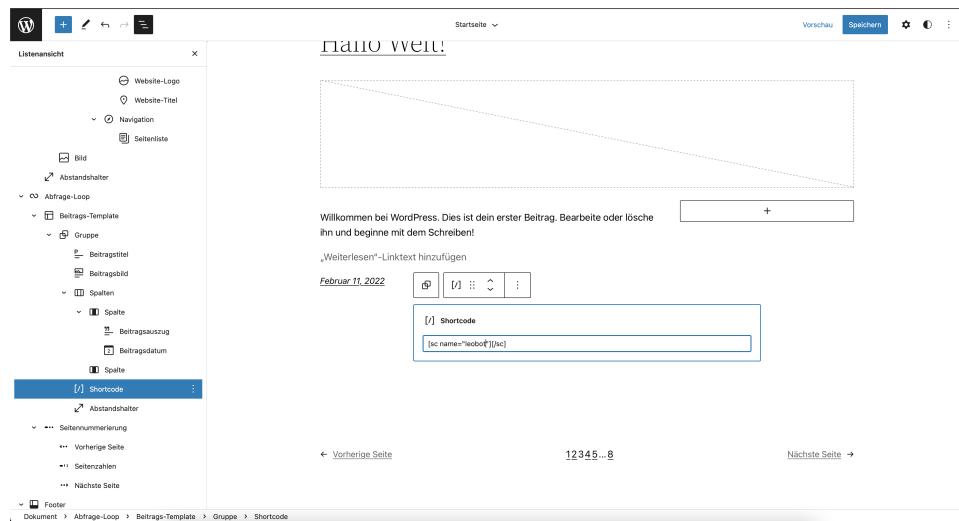


Figure 55: WordPress Editor

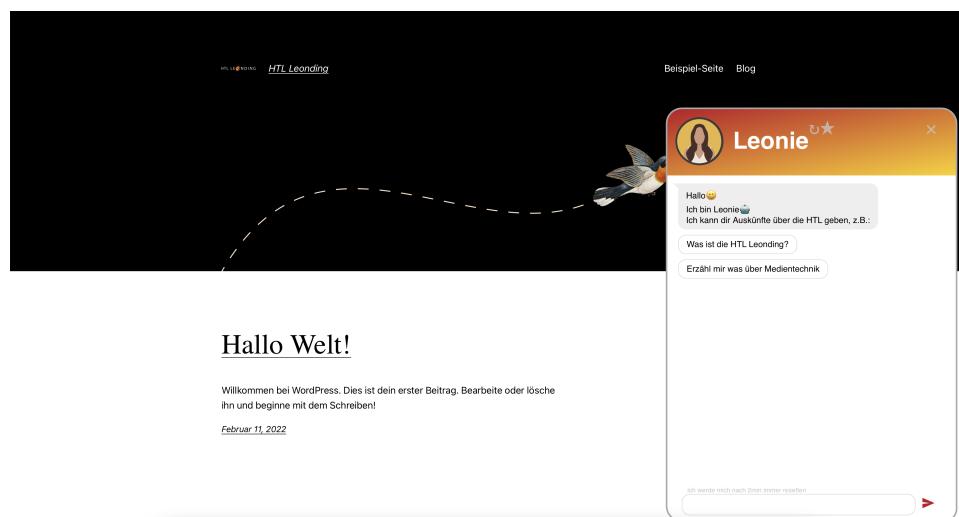


Figure 56: WordPress Seite mit Chatbot

5.6 Deployment

5.6.1 GitHub Actions

Die praktische Arbeit besteht aus sehr vielen einzelnen Projekten, wie dem Chat, Dashboard und Rasa selbst, diese haben alle verschiedenen Funktionen, bei vielen der GitHub Projekte wurde mithilfe von GitHub Actions 3.3.4 das Deployment automatisiert.

Action Server

Der Rasa Custom Action Server wird mithilfe von GitHub Actions auf das LeoCloud Docker Registry geladen.

Listing 52: action_server.yml

```

1  on:
2    push:
3      branches:
4        - main
5      paths:
6        - 'rasa-docker-prototype/actions/**'
7
8  jobs:
9    build_and_deploy:
10       runs-on: ubuntu-latest
11       name: Build Action Server image and upgrade Rasa X deployment
12       steps:
13         - name: Checkout repository
14           uses: actions/checkout@v2
15
16         - id: action_server
17           name: Build an action server with a custom actions
18           uses: RasaHQ/action-server-gha@main
19           # Full list of parameters:
20             https://github.com/RasaHQ/action-server-gha/tree/master#input-arguments
21           with:
22             actions_directory: 'rasa-docker-prototype/actions/'
23             docker_registry: registry.cloud.htl-leonding.ac.at
24             requirements_file: rasa-docker-prototype/actions/requirements.txt
25             docker_image_name: 'f.dumfarth/leobot'
26             docker_registry_login: ${{ secrets.LEO_LOGIN }}
27             docker_registry_password: ${{ secrets.LEO_PASS }}
28             # More details about github context:
29             #
30             https://docs.github.com/en/actions/reference/context-and-expression-syntax-for-github-actions
31             #
32             # github.sha - The commit SHA that triggered the workflow run
33             docker_image_tag: 'leonie'
34             docker_registry_push: true

```

Um den Action Server zu bauen, wird die “RasaHQ/action-server-gha@main” Action [90] benutzt.

Die genutzten Argumente dabei sind:

- actions_directory: Der Ordner in dem sich die Actions befinden.
- docker_registry: Das Docker Registry wohin der Action Server hochgeladen werden soll.

- requirements_file: Der Pfad zur requirements.txt Datei.
- docker_image_name: Der Name des Docker Images.
- docker_registry_login: Dein Username für das Docker Registry, am besten in einem Secret.
- docker_registry_password: Dein Passwort für das Docker Registry, am besten in einem Secret.
- docker_image_tag: der Tag des Docker Images.
- docker_registry_push: True oder False ob das Docker Image auf das Docker Registry hochgeladen werden soll.

Auf der virtuellen Maschine in der `docker-compose.yml` Datei muss noch das Image angegeben werden. Dies wird in Listing 53 beschrieben ist.

Listing 53: docker-compose.yml

```

1 app:
2   restart: always
3   image: "registry.cloud.htl-leonding.ac.at/f.dumfarth/leobot:leon"
4   expose:
5     - "5055"
6   depends_on:
7     - rasa-production

```

Dabei bekommt das Image den Namen “leobot” und den Tag “leon”.

Backend

Das Backend wird mithilfe von GitHub Actions automatisiert gebaut, der Workflow baut das .jar file und dieses wird dann über SSH auf die virtuelle Maschine geladen.

Listing 54: ci.yml

```

1 name: Quarkus Codestart CI
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10   build:
11     runs-on: ubuntu-latest
12     steps:
13       - uses: actions/checkout@v2
14       - name: Set up JDK 17
15         uses: actions/setup-java@v2
16         with:
17           distribution: 'temurin'
18           java-version: '17'
19       - name: Build
20         run: ./mvnw clean package -Dquarkus.package.type=uber-jar -B
21       - name: install ssh key
22         uses: webfactory/ssh-agent@v0.5.3

```

```

23     with:
24         ssh-private-key: ${ secrets.SSH_SERVER_PRIVATE_KEY }
25 - name: create .ssh/known_hosts
26     run: |
27         ssh-keyscan -H -t rsa -v ${ secrets.SERVER } >> ~/.ssh/known_hosts
28 - name: copy binaries to vm
29     run: |
30         echo "Hallo ich bin hier"
31         ls -l target/
32         scp target/leon-1.0.0-SNAPSHOT-runner.jar ${ secrets.SERVER_USER }@${
33             secrets.SERVER }:~/

```

Frontend

Die Chat-Seite 5.3 sowie das Dashboard 5.4 werden mithilfe von GitHub Actions automatisiert gebaut und auf GH Pages veröffentlicht. Dies passiert mit dem Workflow in der `build-and-deploy.yml` Datei:

Listing 55: build-and-deploy.yml

```

1  name: deploy to gh-pages
2
3  on:
4      push:
5          branches:
6              - 'main'
7
8  jobs:
9      build:
10         name: Build
11         runs-on: ubuntu-latest
12         steps:
13             - name: Checkout
14                 uses: actions/checkout@v2
15             - name: Use Node 12.x
16                 uses: actions/setup-node@v1
17                 with:
18                     node-version: '12.x'
19             - name: Install dependencies
20                 run: npm i
21             - name: Build
22                 run: npm run build
23             - name: Archive build
24                 if: success()
25                 uses: actions/upload-artifact@v1
26                 with:
27                     name: dist
28                     path: dist
29
30         deploy:
31             name: Deploy
32             runs-on: ubuntu-latest
33             needs: build
34             steps:
35                 - name: Checkout
36                     uses: actions/checkout@v1
37                 - name: Download build
38                     uses: actions/download-artifact@v1
39                     with:
40                         name: dist
41                 - name: Deploy to GitHub Pages
42                     uses: JamesIves/github-pages-deploy-action@releases/v3
43                     with:
44                         GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
45                         BRANCH: gh-pages
46                         FOLDER: dist/LeoBotHtlLeonding

```

Diese Action hat zwei Jobs, die “build” und “deploy” genannt werden.

Die verschiedenen Tasks in Build laden zuerst eine Node Version herunter, installieren die Dependencies, bilden das Angular Projekt und schließlich wird dist in die Github Actions Artifacts gespeichert.

Beim Deploy Task wird dist von den Artefakten heruntergeladen und wird in den Zweig “gh-pages” des Repositories gespeichert.

Rasa

Wenn in das GH Repo etwas gepushed wird, wird mithilfe einer GitHub Action Rasa trainiert und das Model auf die VM geladen.

_deploy.yml

Listing 56: rasa_deploy.yml

```

1  name: Deploy Rasa to vm
2
3  on:
4    push:
5      branches:
6        - 'main'
7      paths:
8        - 'rasa-docker-prototype/**'
9    pull_request:
10       branches:
11         - 'main'
12
13   workflow_dispatch:
14
15 jobs:
16   branch:
17     name: Deploy
18     runs-on: ubuntu-latest
19     steps:
20       - name: Checkout
21         uses: actions/checkout@v1
22       - name: Deploy to Branch
23         uses: JamesIves/github-pages-deploy-action@releases/v3
24         with:
25           GITHUB_TOKEN: ${{ secrets.TOKEN }}
26           BRANCH: rasa-chatbot
27           FOLDER: rasa-docker-prototype
28
29 tests:
30   name: Train and Test Rasa
31   runs-on: ubuntu-latest
32   needs: branch
33   steps:
34     - uses: actions/checkout@v2
35       with:
36         ref: rasa-chatbot
37     - name: Train and Test Rasa
38       uses: RasaHQ/rasa-train-test-gha@main
39       with:
40         test_type: all
41         rasa_version: 2.8.12-full
42         data_validate: true
43         test_args: --fail-on-prediction-errors
44         rasa_train: true
45         rasa_test: true
46     - name: Upload model
47       uses: actions/upload-artifact@master
48       with:
49         name: model
50         path: models
51
52 deploy:
53   name: "Deploy to vm"
54   runs-on: ubuntu-latest
55   needs: tests

```

```

54     steps:
55       - name: Download model
56         uses: actions/download-artifact@v2
57         with:
58           name: model
59           path: models
60       - name: Copy model to vm
61         uses: garygrossgarten/github-action-scp@release
62         with:
63           local: models
64           remote: /home/chatadm/leobot/models
65           host: ${secrets.SSH_HOST}
66           username: ${secrets.SSH_USER}
67           privateKey: ${secrets.SSH_KEY}
68       - name: Configure SSH
69         run: |
70           mkdir -p ~/.ssh/
71           echo "$SSH_KEY" > ~/.ssh/staging.key
72           chmod 600 ~/.ssh/staging.key
73           cat >>~/.ssh/config <<END
74             Host staging
75               HostName $SSH_HOST
76               User $SSH_USER
77               IdentityFile ~/.ssh/staging.key
78               StrictHostKeyChecking no
79             END
80         env:
81           SSH_USER: ${secrets.SSH_USER}
82           SSH_KEY: ${secrets.SSH_KEY}
83           SSH_HOST: ${secrets.SSH_HOST}
84       - name: Copy model to Rasa X
85         run: ssh staging 'cd /home/chatadm/leobot/models &&
86               ./upload-latest-model.sh'

```

Es wird zuerst geschaut, ob der Push wirklich auf den Branch “main” geschickt wurde und dort im Ordner “rasa-docker-prototype” sich etwas verändert hat. Wenn dies zutrifft, wird der Workflow ausgeführt. Im Job `branch` wird zuerst der Ordner, der mithilfe des “FOLDER:” Arguments angegeben wurde, also “rasa-docker-prototype” auf den BRANCH, der mit “BRANCH:” angegeben wurde, also “rasa-chatbot” geladen. Im Job `tests` wird zuerst definiert, dass der Job “branch” benötigt wird bevor “tests” ausgeführt werden kann. Dies wird mit dem “needs:” Argument definiert. Nun wird die “rasa-train-test-gha@main” Action geladen. Diese Action trainiert das Modell und testet dieses dann direkt. Nachdem das Model trainiert und getestet wurde, wird das Model mit dem Job “upload-artifact” in die Artefakte hochgeladen.

Nun wird im `deploy` Job zuerst das Modell heruntergeladen. Danach wird mit der Action “garygrossgarten/github-action-scp@release” das Modell über SSH auf die virtuelle Maschine geladen. Mit dem “remote” Argument wird der Pfad auf die VM angegeben, hierbei ist dieser “/home/chatadm/leobot/models”. Außerdem werden Secret Variablen “SSH_HOST”, “SSH_USER” und “SSH_KEY” definiert, diese werden mit “secrets.SSH_HOST”, “secrets.SSH_USER” und “secrets.SSH_KEY” angegeben. Danach wird der Host “staging” mit dem User “chatadm” und der Private Key “staging.key” angelegt. Mit diesem Host wird dann in dem Task “Copy model to Rasa X” wird in den /home/chatadm/leobot/models Ordner gewechselt, wo zuvor das Model hochgeladen wurde. Dann wird das Shell-Skript “upload-latest-model.sh”

ausgeführt. Dieses Shell-Skript beinhaltet eine veränderte Version des Befehls, der das Modell auf Rasa X über die API¹ lädt.

Der Befehl zum Hochladen des Modells auf Rasa X, kann in der Rasa X Oberfläche, wie in Abbildung 57 aufgezeigt, gefunden werden.

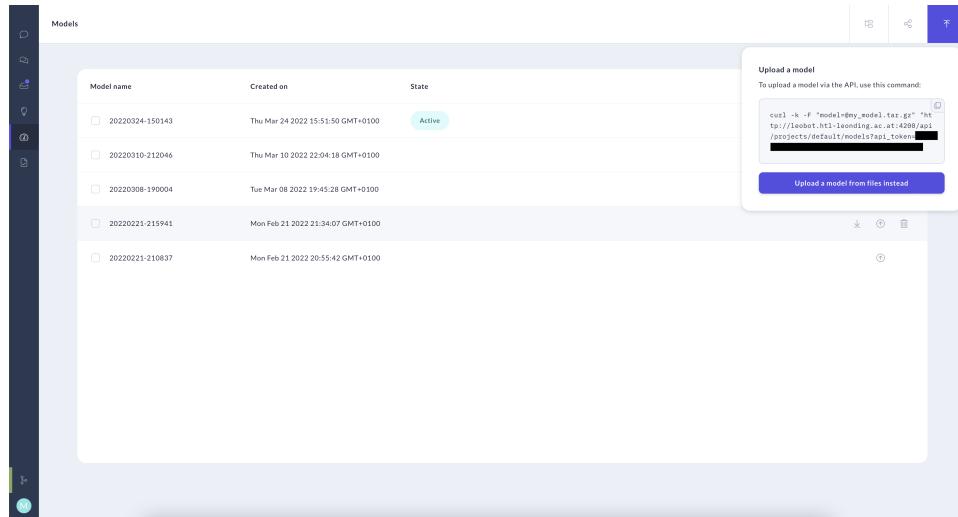


Figure 57: Rasa X Oberfläche mit dem Befehl zum Model upload

Listing 57: Befehl zum Model Upload von Rasa X

```
1 curl -k -F "model=@my_model.tar.gz"
      "http://leobot.html-leonding.ac.at:4200/api/projects/default/models?api_token=TOKEN"
```

Dieser Befehl wurde angepasst, sodass er immer das neuste Model hochlädt, wie in Listing 58 zu sehen ist.

Listing 58: upload-latest-model.sh

```
1 curl -k -F "model=@$(ls -t *tar.gz | head -1)"
      "localhost:4200/api/projects/default/models?api_token=TOKEN"
```

¹API Token zur Sicherheit im Kapitel entfernt

6 Evaluation

6.1 Schwierigkeiten in der Umsetzung

6.1.1 Backend

Beim Backend war eine große Schwierigkeit, dass die Rasa X API nicht sehr gut dokumentiert ist und deshalb sehr viele der Endpoints nur durch Ausprobieren und sehr genaues recherchieren ermittelt werden konnten.

6.1.2 Frontend

Beim Frontend war die größte Schwierigkeit der Export der Chatkomponente, da Angular Materials nicht mit exportiert werden konnte, dies jedoch für die Feedback-Ansicht notwendig ist.

6.2 Fazit

Das geplante Ergebnis der Diplomarbeit war laut dem Diplomarbeitsantrag:

”Ein webbasierter grafischer Avatar, der durch eine modulare Architektur leicht in Webseiten integrierbar ist. Wichtig dabei ist, dass dieser selbstlernend ist und nicht rule-based. Das Wartungspersonal soll dabei Logs des Chatbots erhalten und die Möglichkeit haben, die Wissensbasis zu manipulieren, also neue Inhalte bereitzustellen oder nicht mehr relevante Inhalte zu entfernen.”

In der Arbeit kann man erkennen, dass ein Großteil dieser Punkte erfüllt worden ist.

- Da das Chat-Widget als Webkomponente exportiert wurde, ist es sehr leicht den Chat in Webseiten zu integrieren.
- Selbstlernend ist er nicht, da der Chatbot zwar Machine Learning verwendet, um Muster zu erkennen, damit der Text auch erkannt wird, wenn er nicht wie in den

Trainingsdaten eingegeben wurde. Dabei lernt der Bot allerdings nicht von alleine neue Intents.

- Im Dashboard erhält das Wartungspersonal alle Unterhaltungen.
- Im Dashboard kann das Wartungspersonal die Wissensbasis manipulieren.

Angefangen hat diese Arbeit mit Recherchen über die Geschichte der Chatbots, von Eliza zum Google Assistant, sodass herausgefunden werden kann, wie die Chatbots in der Vergangenheit gearbeitet haben. Es wurde unter anderem mit Rasa, Dialogflow und Keras herumexperimentiert und kleine Prototypen angefertigt, um herauszufinden welche Technologie benutzt werden sollte. Es hat eine lange Zeit gedauert bis der richtige Weg für diese Arbeit gefunden wurde, doch als er gefunden wurde, hat sich unser Chatbot rasend schnell entwickelt. Beim Frontend wurden Prototypen angefertigt und mit Farben und Design gespielt, bis das passende gefunden wurde. Mit dem Endprodukt dieser Arbeit kann noch sehr viel erweitert werden, es könnten Rollen beim Dashboard eingeführt werden oder eine englische Version des Chatbots für eine englische Version der Schulhomepage. Das Erweiterungspotenzial ist sehr groß, nichtsdestotrotz ist der Chatbot und das Dashboard jetzt auch schon ein gut funktionierendes Produkt. Wenn der Chatbot auf der Schulhomepage sein permanentes Zuhause findet, wird er sicher vielen interessierten Personen mit seinem Wissen unterstützen können.

Bibliography

- [1] MonkeyLearn Inc., „What is Text Analysis? A Beginner’s Guide,” o.D., letzter Zugriff am 16.03.2022. Online verfügbar: <https://monkeylearn.com/text-analysis/>
- [2] Yulia Gavrilova, „Machine Learning Text Analysis,” 2020, letzter Zugriff am 16.03.2022. Online verfügbar: <https://serokell.io/blog/machine-learning-text-analysis>
- [3] IBM Cloud Education, „Natural Language Processing (NLP),” 2020, letzter Zugriff am 17.03.2022. Online verfügbar: <https://www.ibm.com/cloud/learn/natural-language-processing>
- [4] tutorialspoint, „NLP - Linguistic Resources,” 2020, letzter Zugriff am 17.03.2022. Online verfügbar: https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_linguistic_resources.htm
- [5] Christopher Kipp, „Einstieg in Natural Language Processing – Teil 2: Preprocessing von Rohtext mit Python,” 2018, letzter Zugriff am 17.03.2022. Online verfügbar: <https://data-science-blog.com/blog/2018/10/18/einstieg-in-natural-language-processing-teil-2-preprocessing-von-rohtext-mit-python>
- [6] Susan Li, „Named Entity Recognition with NLTK and SpaCy,” 2018, letzter Zugriff am 24.03.2022. Online verfügbar: <https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da>
- [7] ICHI.PRO, „Einführung in NLP - Teil 2: Unterschied zwischen Lemmatisierung und Stemming,” 2018, letzter Zugriff am 24.03.2022. Online verfügbar: <https://ichi.pro/de/einfuhrung-in-nlp-teil-2-unterschied-zwischen-lemmatisierung-und-stemming-61064739575228>
- [8] Shivanee Jaiswal, „Natural Language Processing — Dependency Parsing,” 2021, letzter Zugriff am 26.03.2022. Online verfügbar: <https://towardsdatascience.com/natural-language-processing-dependency-parsing-cf094bbbe3f7>
- [9] Francesco Elia, „Constituency Parsing vs Dependency Parsing,” 2020, letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.baeldung.com/cs/constituency-vs-dependency-parsing>
- [10] Ranks.nl, „German Stopwords,” o.D., letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.ranks.nl/stopwords/german>
- [11] Dennis Rudolph, „Skalarprodukt berechnen: Vektoren, Formel und Definition,” 2020, letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.gut-erklaert.de/mathematik/skalarprodukt-berechnen-vektoren-formel-definition.html>
- [12] Bruno Stecanella, „Understanding TF-IDF: A Simple Introduction,” 2019, letzter Zugriff am 27.03.2022. Online verfügbar: <https://monkeylearn.com/blog/what-is-tf-idf/>

- [13] Kinder Chen, „Introduction to Natural Language Processing — TF-IDF,” 2021, letzter Zugriff am 27.03.2022. Online verfügbar: <https://kinder-chen.medium.com/introduction-to-natural-language-processing-tf-idf-1507e907c19>
- [14] C. H. Hobson Lane, Hannes Hapke, *Natural Language Processing in Action: Understanding, analyzing, and generating text with Python*, 1. Aufl. United States of America: Manning, 2019.
- [15] Koaning, „WhatLies,” o.D., letzter Zugriff am 27.03.2022. Online verfügbar: <https://koaning.github.io/whatlies/>
- [16] Thomas Wood, „What is F-score?” o.D., letzter Zugriff am 31.03.2022. Online verfügbar: <https://deepai.org/machine-learning-glossary-and-terms/f-score>
- [17] Karen White, „What is F-score?” 2020, letzter Zugriff am 31.03.2022. Online verfügbar: <https://deepai.org/machine-learning-glossary-and-terms/f-score>
- [18] Wikipedia, „Java (Programmiersprache),” 2022, letzter Zugriff am 20.02.2022. Online verfügbar: [https://de.wikipedia.org/wiki/Java_\(Programmiersprache\)](https://de.wikipedia.org/wiki/Java_(Programmiersprache))
- [19] Tiobe, „Tiobe Index,” 2022, letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.tiobe.com/tiobe-index/>
- [20] Mozilla, „Web Components,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: https://developer.mozilla.org/de/docs/Web/Web_Components
- [21] Rasa Technologies GmbH, „Introduction to Rasa X,” 2021, letzter Zugriff am 31.03.2022. Online verfügbar: <https://rasa.com/docs/rasa-x/>
- [22] ——, „Docker Compose Installation,” 2021, letzter Zugriff am 31.03.2022. Online verfügbar: <https://rasa.com/docs/rasa-x/installation-and-setup/install/docker-compose/>
- [23] EFF, „Certbot,” o.D., letzter Zugriff am 01.04.2022. Online verfügbar: <https://certbot.eff.org>
- [24] Daniel Rösch, „Rasa Core,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://botfriends.de/blog/botwiki/rasa-core/>
- [25] S. S. Abhishek Singh, Karthik Ramasubramanian, *Introduction to Microsoft Bot, RASA, and Google Dialogflow*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [26] Rasa Technologies GmbH, „Rasa Policies,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/policies/>
- [27] ——, „Policy Priority,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/policies#policy-priority>
- [28] ——, „TED Policy,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/policies#ted-policy>
- [29] Daniel Rösch, „Rasa NLU,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://botfriends.de/blog/botwiki/rasa-nlu/>
- [30] Tobias Wochinger, „Rasa NLU in Depth: Part 1 – Intent Classification,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/blog/rasa-nlu-in-depth-part-1-intent-classification/>

- [31] Rasa Technologies GmbH, „How to choose a pipeline,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#how-to-choose-a-pipeline>
- [32] ——, „Tuning Your Model,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/>
- [33] ——, „Component Lifecycle,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#component-lifecycle>
- [34] ——, „Component Lifecycle,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#component-lifecycle>
- [35] Bhavan Ravi, „NLP Behind Chatbots — Demystifying RasaNLU — 1 — Training,” 2018, letzter Zugriff am 03.03.2022. Online verfügbar: <https://medium.com/bhavaniravi/demystifying-rasa-nlu-1-training-91a08429c9fb>
- [36] Sociopath auf Stackoverflow, „Having a combination of pre trained and supervised embeddings in rasa nlu pipeline,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://stackoverflow.com/questions/63683812/having-a-combination-of-pre-trained-and-supervised-embeddings-in-rasa-nlu-pipeli>
- [37] R. T. I. Justine Petraityte, *The Rasa Masterclass Ebook Pre Configured Pipelines*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [38] Anran Jiao, „Components of Spacy Sklearn Pipeline,” 2020, letzter Zugriff am 07.03.2022. Online verfügbar: https://www.researchgate.net/figure/The-list-of-components-which-are-equal-to-pipeline-spacy-sklearn_fig2_340534832
- [39] Rasa Technologies GmbH, „SpacyNLP,” 2022, letzter Zugriff am 07.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components#spacynlp>
- [40] ——, „MiteNLP,” 2022, letzter Zugriff am 07.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components#mitienlp>
- [41] Intellify Solutions, „Open Source Conversational AI Rasa,” o.D., letzter Zugriff am 07.03.2022. Online verfügbar: <https://intellifysolutions.com/blog/know-open-source-conversational-ai-rasa/>
- [42] Alan Nichol, „Supervised Word Vectors from Scratch in Rasa NLU,” 2018, letzter Zugriff am 03.03.2022. Online verfügbar: <https://medium.com/rasa-blog/supervised-word-vectors-from-scratch-in-rasa-nlu-6daf794efcd8>
- [43] Rasa Technologies GmbH, „How to Choose a Pipeline,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#how-to-choose-a-pipeline>
- [44] ——, „Sensible Starting Pipelines,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#sensible-starting-pipelines>
- [45] ——, „Components,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/>
- [46] ——, „Rasa NLU Examples,” 2020, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasahq.github.io/rasa-nlu-examples/>

- [47] ——, „Comparing NLU Pipelines,” 2022, letzter Zugriff am 31.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/testing-your-assistant/#comparing-nlu-pipelines>
- [48] ——, „Interpreting the Output,” 2022, letzter Zugriff am 31.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/testing-your-assistant/#interpreting-the-output>
- [49] ——, „WhitespaceTokenizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#whitespacetokenizer>
- [50] R. T. I. Justina Petraityte, *The Rasa Masterclass Ebook WhitespaceTokenizer*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [51] J. P. Rasa. (Ep 4 - Rasa Masterclass) Training the NLU models: understanding pipeline components | Rasa 1.8.0. Youtube. Online verfügbar: <https://www.youtube.com/watch?v=ET1k9OrsfYQ&list=PL75e0qA87dlHQny7z43NduZHPo6qd-cRc&index=6>
- [52] R. T. I. Justina Petraityte, *The Rasa Masterclass Ebook RegexFeaturizer*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [53] Rasa Technologies GmbH, „RegexFeaturizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#regexfeaturizer>
- [54] ——, „CRFEntityExtractor,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components#crfentityextractor>
- [55] ——, „LexicalSyntacticFeaturizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#lexicalsyntacticfeaturizer>
- [56] Rasa, Justina Petraityte, „(Ep 4 - Rasa Masterclass) Training the NLU models: understanding pipeline components | Rasa 1.8.0,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://www.youtube.com/watch?v=YxMzz6NF6Zw&list=PL75e0qA87dlEjGAc9j9v3a5h1mxI2Z9fi&index=9>
- [57] Rasa Technologies GmbH, „CountVectorsFeaturizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#countvectorsfeaturizer>
- [58] R. T. I. Justina Petraityte, *The Rasa Masterclass Ebook Count Vectors Featurizer*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [59] Rasa Technologies GmbH, „DietClassifier,” 2022, letzter Zugriff am 01.04.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#dietclassifier>
- [60] ——, „EntitySynonymMapper,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#entitysynonymmapper>
- [61] ——, „ResponseSelector,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#responseselector>
- [62] ——, „Retrieval Intent,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/glossary#retrieval-intent>
- [63] ——, „Chitchat and FAQs,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/chitchat-faqs>

- [64] ——, „FallbackClassifier,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#fallbackclassifier>
- [65] ——, „NLU Fallback,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/fallback-handoff/#nlu-fallback>
- [66] Alex Weidauer, „Conversational AI: Your Guide to Five Levels of AI Assistants in Enterprise,” 2018, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/blog/conversational-ai-your-guide-to-five-levels-of-ai-assistants-in-enterprise/>
- [67] Rasa, „(Ep 1 - Rasa Masterclass) Intro to conversational AI and Rasa | Rasa 1.8.0,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://www.youtube.com/watch?v=-F6h43DRpcU&list=PL75e0qA87dlHQny7z43NduZHPo6qd-cRc&index=3&t=69>
- [68] R. T. I. Justine Petraityte, *The Rasa Masterclass Ebook*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [69] Rasa Technologies GmbH, „Domain,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain/>
- [70] ——, „Intents,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#intents>
- [71] ——, „Responses,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#responses>
- [72] ——, „Stories,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/>
- [73] ——, „Checkpoints und OR-Statements,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/#checkpoints-and-or-statements>
- [74] ——, „Checkpoints,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/#checkpoints>
- [75] ——, „Or Statements,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/#or-statements>
- [76] ——, „Rules,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/rules>
- [77] ——, „Slots,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#slots>
- [78] ——, „Entities,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#entities>
- [79] Mady Mantha, „Entity Roles and Groups,” 2020, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/blog/introducing-entity-roles-and-groups/>
- [80] Rasa Technologies GmbH, „Forms,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#forms>
- [81] ——, „Synonyms,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/nlu-training-data/#synonyms>

- [82] Rasa, „(Ep 8 - Rasa Masterclass)Implementing custom actions, forms and fallback | Rasa 1.8.0,” 2019, letzter Zugriff am 02.03.2022. Online verfügbar: https://www.youtube.com/watch?v=9POI7LiKH_8
- [83] Rasa Technologies GmbH, „Actions,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#actions>
- [84] ——, „Rasa Train,” 2022, letzter Zugriff am 30.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/command-line-interface/#rasa-train>
- [85] ——, „Conversation-Driven Development,” 2022, letzter Zugriff am 31.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/conversation-driven-development/>
- [86] Helmut Papp, Ralf Moros, Frank Luft, „Grundlagen Regelung,” o.D., letzter Zugriff am 01.04.2022. Online verfügbar: <https://deepai.org/machine-learning-glossary-and-terms/f-score>
- [87] Manuel Gräber, „Regelkreis einfach erklärt,” 2021, letzter Zugriff am 01.04.2022. Online verfügbar: <https://tlk-energy.de/blog/regelkreis-intuitiv-verstehen>
- [88] Quarkus, „USING SECURITY WITH .PROPERTIES FILE,” o. D., letzter Zugriff am 02.04.2022. Online verfügbar: <https://quarkus.io/guides/security-properties>
- [89] Aakash Chakravarthy, „Shortcoder — Create Shortcodes for Anything,” 2022, letzter Zugriff am 01.04.2022. Online verfügbar: <https://de.wordpress.org/plugins/shortcoder/>
- [90] Tomasz Czekajlo Tobias Wochinger, „RasaHQ/rasa-action-server-gha,” 2021, letzter Zugriff am 01.04.2022. Online verfügbar: <https://github.com/RasaHQ/rasa-action-server-gha>

List of Figures

1	Use-Case Diagramm	3
2	Beispiel für die Beziehung zweier Wörter beim Dependency Parsing [8]	9
3	Darstellung des Dependency Parsing [9]	9
4	Darstellung des Constituency Parsing [9]	10
5	Skalarprodukt von zwei Vektoren [11]	12
6	Berechnung von TF-IDF [13]	13
7	WhatLies Illustration [15]	15
8	Whatlies Illustration [15]	16
9	Whatlies Illustration [15]	16
10	F-Score Berechnung [16]	17
11	Precision Berechnung [17]	17
12	Recall Berechnung [17]	17
13	Java Logo[18]	19
14	Beispiel eines Workflows	23
15	Startseite Wordpress	25
16	Wordpress Liste der Blogeinträge	25
17	Wordpress Blopost Editor	25
18	Rasa Core Aufbau [25]	27
19	Component Lifecycle [33]	28
20	Component Lifecycle [35]	29
21	Pretrained oder Supervised Embeddings [30, 37]	31
22	Comparision Graph bei unserer Pipeline	34
23	Histogramm für 0% Exklusion	35
24	Histogramm für 75% Exklusion	36
25	Konfusionsmatrix für chitchat Intents	37
26	Tokens durch einen WhitespaceTokenizer [51]	37
27	Regular Expression Beispiel [51]	38
28	Lookup Table Beispiel [51]	38
29	Funktionsweise vom CRFEntityExtractor [51]	39
30	Funktionsweise von einem Featurizer [56]	39
31	CountVectorsFeaturizer Beispiel [56]	40
32	5 Stufen von AI [67]	43
33	Rasa Init Konsolenausgabe	52
34	File Tree nach dem Initialisieren	52
35	Rasa Shell Ausgabe mit Debug Flag	54
36	Ausgabe beim Interactive Befehl	54
37	Systemarchitektur	55
38	Konzept Chatbot geschlossen	58
39	Konzept Chatbot geöffnet	58
40	Chatbot auf einer ganzen Seite	59
41	Chatbot	59
42	Feedback Fenster	60

43	Aufbau vom Chat Fenster	60
44	3D Leonie	61
45	2D Leonie	61
46	Leobot Conversation Cycle	62
47	Allgemeiner Regelkreis [86]	64
48	Regelkreis am Beispiel einer Heizung [86]	64
49	Anmeldefenster	66
50	Dashboard	66
51	Dashboard nach Datum gefiltert	66
52	Vorschläge	67
53	Vorschlag benutzt	67
54	Shortcoder in WordPress Plugin Manager	68
55	WordPress Editor	68
56	WordPress Seite mit Chatbot	68
57	Rasa X Oberfläche mit dem Befehl zum Model upload	74

List of Tables

Quellcodeverzeichnis

1	Beispiel für die Tokenization	6
2	Ausnahme für bestimmte Tokens	6
3	Beispiel für POS-Tagging	7
4	Beispiele für Entities	7
5	Stemming von deutschen Wörtern	7
6	Lemmatisierung von deutschen Wörtern	8
7	Bag of Words	12
8	Bag of n-grams	12
9	TF-IDF Beispiel	13
10	HTML File	21
11	credentials.yml	22
12	Install certbot and create certificates	22
13	rasax.nginx.template	22
14	ssl.conf.template	23
15	In der Action	24
16	SpaCy Sklearn Pipeline	29
17	MITIE Sklearn Pipeline	30
18	Tensorflow Embedding Pipeline	30
19	Supervised Embedding Pipeline	30
20	Spacy Startpipeline	31
21	Default Pipeline	32
22	Unsere Pipeline	32
25	CountVectorsFeaturizer	40
26	DietClassifier in Pipeline	40
27	Output des DIETClassifiers [59]	41
28	Entity Synonym Mapper	41
29	Response Classifier	42
31	Fallback Classifier	43
32	Intents Beispiel	44
33	Responses Beispiel	45
34	Stories Beispiel	46
35	Checkpoints Beispiel	46
36	OR Beispiel	47
37	Rules Beispiel	47
38	Slots Beispiel	48
39	Entities in der Domain	48
40	Entities Beispiel	48
41	Entity Roles Beispiel	49
42	Entity Groups Beispiel	49
43	Entity Groups Beispiel 2	49
44	Forms Beispiel	50
45	Synonym Beispiel	50
46	Custom Action für die Ausgabe des aktuellen Datums	51

47	Custom Actions in Domain definiert	51
48	Befehl fürs Initialisieren	51
49	Rasa Train Befehle	53
50	Rasa Shell Befehle	53
51	Interaktive Trainingssession starten	54
52	action_server.yml	69
53	docker-compose.yml	70
54	ci.yml	70
55	build-and-deploy.yml	71
56	rasa_deploy.yml	72
57	Befehl zum Model Upload von Rasa X	74
58	upload-latest-model.sh	74

Anhang