Automatische Erzeugung von Marginalien

Aaron Lang

Masterarbeit

im Studiengang Angewandte Informatik

der Fakultät Wirtschaftsinformatik und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Prüfer: Prof. Dr. Andreas Henrich

Lehrstuhl für Medieninformatik

Bamberg 2026

URN: urn:nbn:de:bvb:473-irb-112598x

DOI: https://doi.org/10.20378/irb-112598

Inhaltsverzeichnis

1 Einleitung

Zusammenfassungen von Texten sind allgegenwärtig: Romane besitzen eine kurze Inhaltsbeschreibung, Zeitschriftenartikel fassen die wesentlichen Kernaussagen in wenigen Stichpunkten zusammen und bei wissenschaftlichen Publikationen wie Papers bietet der Abstract einen Überblick über die Forschungsmethodik und -ergebnisse. Dabei verfolgen viele der Anwendungsgebiete solcher Zusammenfassungen denselben Zweck: Zeitersparnis bei der Informationssuche. Die benötigte Zeit für die Einschätzung, ob ein Paper für die eigene Forschung relevant ist, kann mithilfe des Abstracts reduziert werden. Ein Roman muss nicht erst angefangen werden, nur um nach etlichen Seiten festzustellen, dass dieser nicht dem eigenen Geschmack entspricht.

Analog dazu können Schlagworte als noch weiter reduzierte Form von Zusammenfassungen ebenfalls die Informationssuche beschleunigen. So können bei Nachrichtenportalen alle Meldungen mit dem Schlagwort „Fußball“ vom Leser ignoriert werden, wenn dieser nicht an Fußball interessiert ist. Darüber hinaus können Schlagworte auch für weitergehende Funktionalitäten wie das Filtern nach bestimmten Schlagworten Verwendung finden. Dadurch können etwa alle Nachrichtenartikel zum Thema „Ukraine“ gesammelt angezeigt werden. Papers nutzen Schlagworte auch für eine bessere Auffindbarkeit via Suchmaschine oder eine automatische Ermittlung thematisch ähnlicher Papers anhand übereinstimmender Schlagworte.

Neben positiven Auswirkungen auf die Informationssuche in Form von Zeitersparnis beim Prüfen eines Textes auf Relevanz oder einer besseren Auffindbarkeit innerhalb einer Suchfunktion können Zusammenfassungen und Schlagworte möglicherweise einen lernpsychologischen Nutzen bieten. Werden diese schon vor dem eigentlichen Text gelesen, wird Vorwissen geschaffen, der Leser weiß bereits, was ihn erwartet. Dieses Prinzip wird mitunter auch in Lehrbüchern verfolgt: Ein Kapitel beginnt mit einer Vorstellung des kommenden Kapitelinhalts („Was werden wir lernen?“) und endet zudem noch in einem kurzen Resümee („Was haben wir gelernt?“), um das Gelernte weiter zu festigen.

Darüber hinaus weisen manche Lehrbücher ein weiteres Konzept auf: Marginalien. Diese befinden sich am Seitenrand neben dem Text und bieten meist eine kurze Beschreibung des Inhalts des nebenstehenden Textabschnitts. Lehrbücher sind wie viele andere Textformen in Kapitel oder Abschnitte unterteilt, die den Text inhaltlich strukturieren. Innerhalb eines solchen Kapitels existiert als kleinste Struktureinheit eine Unterteilung in Sinneinheiten, die durch Absätze dargestellt werden. Während es für Kapitel eine Beschreibung ihrer inhaltlichen Strukturierung in Form einer Kapitelüberschrift gibt, können Marginalien diese Funktion für ihre jeweiligen Absätze übernehmen. Dadurch können Marginalien dem Leser zum einen eine gewisse Vorstrukturierung bieten, wodurch die Bedeutung der Sinneinheit innerhalb der umgebenden anderen Absätze nicht vom Leser selbst erschlossen werden muss. Zum anderen kann der Leser bereits vor dem Lesen des jeweiligen Absatzes anhand der Marginalie auf einen Blick erkennen, welchen Inhalt der Absatz behandelt, was die Suche nach dem Absatz mit einem bestimmten Thema beschleunigen kann. Es ist also plausibel anzu-

nehmen, dass Marginalien insbesondere für Lehrbücher eine sinnvolle Ergänzung darstellen können. Marginalien werden zwar in einigen Lehrbüchern eingesetzt, aber längst nicht in allen. Eine automatische Erzeugung von Marginalien könnte helfen, diese Lücke zu schließen.

Der Forschungsbereich der natürlichen Sprachverarbeitung befasst sich intensiv mit Problemen, die in eine ähnliche Richtung wie die Erzeugung von Marginalien gehen, dazu gehört Keyphrase-Prediction und Text-Summarization. Inwiefern sich diese Verfahren für die Erzeugung von Marginalien eignen, wurde jedoch noch nicht untersucht. Im Folgenden soll daher untersucht werden, inwieweit sich Verfahren aus diesen Bereichen auf die automatische Erzeugung von Marginalien anwenden lassen.

2 Forschungsstand

Da die automatische Erzeugung von Marginalien noch unerforscht ist, soll im Folgenden ein Überblick über Fragestellungen und Methoden ähnlicher Probleme aus der natürlichen Sprachverarbeitung gegeben werden: Text-Summarization und Keyphrase-Prediction.

Während Text-Summarization und Keyphrase-Prediction prinzipiell ähnlich sind, besteht ein wesentlicher Unterschied in deren Ausgabe. Text-Summarization nimmt ein Dokument als Eingabe und erzeugt eine Zusammenfassung in Form eines Fließtexts, während Keyphrase-Prediction einzelne relevante Begriffe (Keyphrases) zurückgibt. Im Verlauf der näheren Beschäftigung mit Marginalien stellte sich heraus, dass Keyphrase-Prediction der Erzeugung von Marginalien konzeptuell nähersteht. Daher soll im Folgenden nur ein kurzer, beispielhafter Einblick in den Bereich der Text-Summarization gegeben werden, bevor der Fokus auf Keyphrase-Prediction gelegt wird.

2.1 Text-Summarization

Der Bereich der Text-Summarization [1] beschäftigt sich mit der Erzeugung einer Zusammenfassung eines Ausgangsdokuments unter Beibehaltung relevanter Informationen. Dabei werden zwei Herangehensweisen unterschieden: extraktive und abstraktive Verfahren.

Extraktive Verfahren identifizieren diejenigen Sätze aus dem Ausgangsdokument, die relevante Informationen beinhalten und konkatenieren diese zu einer Zusammenfassung . Die so erzeugten Zusammenfassungen enthalten daher nur Satzfolgen, die auch so Wort-für-Wort im Ausgangsdokument enthalten sind.

Ein Beispiel für extraktive Verfahren sind graphenbasierte Ansätze, die Dokumente als Graphen $G = (E, V)$ modellieren [2]. Die Knoten E des Graphen bilden die Sätze des Dokumentes und dessen Kanten V stellen die Beziehungen zueinander dar [2]. Ein bekannter Vertreter ist TextRank [3]. TextRank verbindet diejenigen Knoten (Sätze) miteinander, die überlappenden Inhalt aufweisen und somit ähnlich zueinander sind [3]. Diese Ähnlichkeit zwischen zwei Sätzen

S_i und S_j wird definiert als

$$Similarity(S_i, S_j) = \frac{|\{w_k \mid w_k \in S_i \wedge w_k \in S_j\}|}{\log(|S_i|) + \log(|S_j|)} \quad (1)$$

wobei ein Satz bestehend aus n Wörtern durch eine Menge an Wörtern w_1 bis w_n repräsentiert wird [3]. Der Nenner dient dabei als Normierung, damit lange Sätze nicht bevorzugt werden [3]. Dieses Ähnlichkeitsmaß wird zur Gewichtung der Kanten verwendet und ermöglicht schließlich mittels eines Ranking-Algorithmus die n besten Sätze zu ermitteln und zu einer Zusammenfassung zusammenzuführen [3].

Abstraktive Verfahren sind hingegen in der Lage, Zusammenfassungen „in eigenen Worten“ zu formulieren, können demnach die relevanten Informationen des Ausgangsdokuments in einer Weise wiedergeben, die so nicht im Ausgangsdokument existiert.

2.2 Keyphrase-Prediction

Da der Bereich der Keyphrase-Prediction ein weites Forschungsfeld darstellt, kann hier kein vollständiger und umfassender Überblick gegeben werden. Stattdessen werden ausgewählte Verfahren kurz erläutert, sodass ein Einblick in die Vielfältigkeit und auch Gemeinsamkeiten der unterschiedlichen Verfahren möglich ist. State-of-the-Art-Verfahren werden auf Basis der Problemanalyse (Kap. 3) im Rahmen der Lösungsansätze (Kap. 4) besprochen.

Keyphrase-Prediction befasst sich mit der Ermittlung einer oder mehrerer Keyphrases aus einem Textdokument. Bei Keyphrases handelt es sich um n-Gramme, die das zugehörige Textdokument repräsentieren sollen [4]. Ein Unterschied zur Text-Summarization ist also die Granularität der Ergebnisse: Text-Summarization ermittelt relevante oder repräsentative Satzfolgen, Keyphrase-Prediction dagegen Wortfolgen. Analog zur Text-Summarization existiert auch hier die Unterscheidung zwischen extraktiven und abstraktiven Verfahren: Keyphrase-Extraction und Keyphrase-Generation.

Keyphrase-Extraction Keyphrase-Extraction-Verfahren sind in der Lage, N-Gramme eines Textes als Keyphrases zu identifizieren. Somit können nur im Text wortwörtlich enthaltene Keyphrases ermittelt werden, sogenannte Present-Keyphrases.

Keyphrase-Extraction kann mittels überwachter oder unüberwachter Verfahren durchgeführt werden. Die unüberwachten Verfahren können auf statistischen Merkmalen, Graphen oder Deep-Learning basieren [5].

Zu den auf statistischen Merkmalen basierenden Verfahren zählt TF-IDF[1] (Term Frequency – Inverse Document Frequency), das oft als Baseline eingesetzt wird. TF-IDF ist das Produkt aus der Vorkommenshäufigkeit TF eines

[1] Die Erläuterungen zu TF-IDF folgen den Darstellungen in [6]. Das Konzept von TF-IDF wird dort jedoch nicht erstmals beschrieben. Die Bestandteile TF und IDF wurden – wenn auch nicht unter diesen Bezeichnungen – erstmals in [7] bzw. [8] eingeführt.

Keyphrase-Kandidaten und inverser Dokumenthäufigkeit IDF. Dabei ist IDF definiert als

$$IDF = log_2 \frac{N}{1 + DF} \tag{2}$$

mit DF als Dokumentenhäufigkeit, welche die Anzahl der Dokumente innerhalb eines Korpus angibt, die diesen Keyphrase-Kandidaten enthalten, und N als Anzahl aller Dokumente im Korpus.

Ein Beispiel für ein etwas komplexeres statistisches Verfahren ist YAKE!, das neben der Vorkommenshäufigkeit W_{Freq} eines Worts w noch die Groß-/Kleinschreibung W_{Case}, Wortposition $W_{Position}$, Wortähnlichkeit zu Stoppwörtern im Kontext W_{Rel} sowie die Vorkommenshäufigkeit in unterschiedlichen Sätzen $W_{DifSentence}$ als Merkmale heranzieht [9].

Nach der Zerlegung des Eingabetextes in Tokens werden die genannten Merkmale in einer gemeinsamen Funktion $S(w)$ für die einzelnen Keyword-Kandidaten w eingesetzt [9]. $S(w)$ ist nach [9] definiert als

$$S(w) = \frac{\mathbf{W}_{\text{Rel}} * \mathbf{W}_{\text{Position}}}{\mathbf{W}_{\text{Case}} + \frac{\mathbf{W}_{\text{Freq}}}{\mathbf{W}_{\text{Rel}}} + \frac{\mathbf{W}_{\text{DifSentence}}}{\mathbf{W}_{\text{Rel}}}} \tag{3}$$

Relevante Wörter sollen dadurch einen höheren Score $S(w)$ erhalten [9].

Diese Gewichtung berücksichtigt allerdings nur Keywords, d.h. Unigramme[2], statt Keyphrases [9]. Der finale Score $S(kw)$ eines bis zu drei Wörter umfassenden Keyphrase-Kandidaten ergibt sich aus

$$S(\text{kw}) = \frac{\prod_{w \in \text{kw}} S(w)}{\text{TF}(\text{kw}) * \left(1 + \sum_{w \in \text{kw}} S(w)\right)} \tag{4}$$

wobei der Zähler die Scores $S(w)$ der einzelnen Wörter w der Keyphrase kw multipliziert [9]. Damit beispielsweise Keyphrases aus drei Wörtern keinen Vorteil gegenüber Keyphrases aus ein oder zwei Wörtern haben, teilt der Nenner durch die Summe der Scores $S(w)$ [9]. Zusätzlich wird durch die Vorkommenshäufigkeit TF einer Keyphrase kw geteilt, sodass häufiger auftretende Keyphrase-Kandidaten einen besseren Score $S(kw)$ erhalten [9]. Ein besserer Score bedeutet hier einen niedrigeren Wert für $S(kw)$ [9].

Abschließend wird von denjenigen Keyphrase-Kandidaten, die sich ähnlich sind, d.h. deren Levenshtein-Distanz [10] über einem gewissen Schwellwert liegt, der Kandidat mit einem schlechteren (höheren) $S(kw)$ entfernt [9].

Andere Keyphrase-Extraction-Verfahren modellieren Dokumente als Graphen, analog zu den Schilderungen zur Text-Summarization in Kapitel 2.1. Das dort betrachtete Text-Summarization-Verfahren TextRank kann statt auf Satzebene auch auf Wortebene angewandt werden und dadurch Keyphrases extrahieren [3].

[2] In der Literatur werden die Begriffe Keyword und Keyphrase teils synonym gebraucht. Um Ambiguität zu vermeiden, werden die Begriffe in dieser Arbeit im strengeren Wortsinn verwendet: Keywords als Unigramme und Keyphrases als N-Gramme.

PositionRank betrachtet Dokumente ebenfalls als Graphen, berücksichtigt dabei aber die Position der Kandidaten-Keyphrases innerhalb des Dokuments [11].

Im ersten Schritt konstruiert PositionRank einen Graphen, dessen Knoten aus Wörtern des Dokuments und dessen Kanten deren Kookkurrenzen darstellen [11]. Die Kanten werden anhand der Anzahl der Kookkurrenzen innerhalb eines Fensters bestimmter Größe gewichtet [11]. Nun werden die Positionen der Kandidaten-Keyphrases im Text berücksichtigt: Für jedes Auftreten eines Kandidaten-Keywords im Dokument wird dessen inverse Position aufaddiert [11]. Je früher im Dokument und je häufiger die Kandidaten vorkommen, desto stärker werden diese gewichtet [11]. Für jeden Keyword-Kandidaten wird das so ermittelte Gewicht normiert, indem das jeweilige Gewicht durch die Summe aller Gewichte dividiert wird [11].

Nun kann der PageRank $S(v_i)$[12] eines Knoten (Keyword-Kandidaten) v_i durch rekursive Anwendung der Gleichung

$$S(v_i) = (1 - \alpha) \cdot \tilde{p}_i + \alpha \cdot \sum_{v_j \in \mathrm{Adj}(v_i)} \frac{w_{ji}}{O(v_j)} S(v_j) \tag{5}$$

ermittelt werden [11]. Dabei ist $\tilde{p}$ der Vektor aller normierten Positionsgewichte, $\tilde{p}_i$ das normierte Gewicht des Knotens v_i, α ein Faktor zur Zyklenvermeidung, w_{ij} das Gewicht der Kante zwischen Knoten v_j und v_i sowie $O(v_j)$ die Summe der Gewichte aller ausgehenden Kanten des Knotens v_j [11].

Anschließend werden diejenigen Kandidaten-Keywords, die im Dokument aufeinander folgen, zu Keyphrases aus bis zu drei Wörtern konkateniert [11]. Die so erzeugten Keyphrases werden anhand der Summe der PageRank-Scores ihrer Bestandteile bewertet und die besten Keyphrases als Ergebnis zurückgegeben [11].

Die vorhergehenden Verfahren berücksichtigen verschiedene Merkmale, wie die Position, Häufigkeit und auch Kookkurrenzen von Kandidaten-Keyphrases. Deep-Learning-Verfahren zielen darauf ab, semantische Beziehungen innerhalb des Dokuments zu nutzen [13]. Dafür können beispielsweise Einbettungen von Keyphrase-Kandidaten und dem jeweiligen Dokument anhand ihrer Kosinusdistanz miteinander verglichen werden [5]. Dabei ist die Annahme, dass je größer die ermittelte Ähnlichkeit, desto relevanter ist die Keyphrase für das Dokument [5].

Bei KPRank handelt es sich um ein Verfahren, das ähnlich wie zuvor PositionRank graphenbasiert ist und den PageRank bestimmt [14]. Anders als PositionRank verwendet KPRank allerdings zusätzlich Einbettungen, um die Ähnlichkeit zwischen Keyphrase-Kandidaten und dem Dokument zu bestimmen [14].

Das Vorgehen hinsichtlich der Konstruktion des Graphen entspricht weitgehend dem von PositionRank, jedoch stellt der Vektor $\tilde{p}_i$ hier nicht nur Positionsgewichte dar, sondern wird um einen Thema-Score („theme score“) erweitert [14]. Dieser Thema-Score soll für ein Kandidaten-Keyword je höher sein, desto mehr dieses dem Thema des Dokuments ähnelt [14]. Das Embedding des Dokuments, der Thema-Vektor („theme vector“) T_D, wird durch Mitteln über

die SciBERT-Einbettungen [15] aller Adjektive und Nomen des Dokumenttitels gebildet [14]. Der Thema-Score eines Knotens n_i wird anschließend mittels Kosinusdistanz zwischen dem SciBERT-Embedding von n_i und T_D bestimmt [14]. Das Produkt aus den Positionsgewichten analog zu PageRank und den Thema-Scores ergibt schließlich das finale Gewicht w_i für n_i [14]. Die Berechnung des PageRank, die Konkatenation von Kandidaten-Keywords zu Keyphrases und die Bewertung der Keyphrases anhand der Summe der PageRank-Scores erfolgt wie für PositionRank, allerdings können Keyphrases hier aus bis zu vier Wörtern bestehen [14].

Reference Vector Algorithm (RVA) nutzt ebenfalls Einbettungen [16]. Statt vortrainierte Einbettungen zu verwenden, werden hier jedoch lokal auf dem jeweiligen Dokument trainierte GloVe-Einbettungen [17] eingesetzt [16]. Anschließend wird ein Referenz-Vektor („reference vector“) für den Titel und Abstract des Dokuments durch Mitteln der dortigen Wort-Einbettungen erzeugt [16]. Die Keyphrase-Kandidaten sind Uni- bis Trigramme aus Titel und Abstract des Dokuments, nicht aus dem vollständigen Dokument, und durchlaufen Vorverarbeitungsschritte wie etwa die Entfernung von Stoppwörtern [16]. Die finale Bewertung der Kandidaten-Keyphrases erfolgt nun anhand der Summe der Kosinusähnlichkeiten der einzelnen Wörter der Keyphrase mit dem Referenz-Vektor [16]. Dadurch können längere und damit potenziell informativere Keyphrases eine bessere Bewertung erhalten [16].

Bei den überwachten Keyphrase-Extraction-Verfahren sind Deep-Learning-Ansätze verbreitet, die unterschiedliche Modelle nutzen wie Convolutional-Neural-Networks (CNNs), Recurrent-Neural-Networks (RNNs), Generative-Adversarial-Networks (GANs) oder auch die Transformer-Architektur [13].

DivGraphPointer (Diversified Graph Pointer) ist ein hybrides Verfahren, das sowohl graphenbasiert ist als auch ein CNN einsetzt [18]. DivGraphPointer greift auf das Encoder-Decoder-Modell zurück und erstellt zunächst mittels Adjazenzmatrizen der Wörter des Dokuments einen Graphen, der dieses Dokument modelliert [18]. Dieser Graph dient als Eingabe für den Encoder [18], einem Graph-Convolutional-Network (GCN) [19]. Das GCN generiert eine Dokumentrepräsentation als Eingabe für den Decoder [18]. Diese graphenbasierte Dokumentrepräsentation mittels GCN soll Vorteile gegenüber CNNs oder RNNs bieten, wie etwa die explizite Erfassung kurz- und langfristiger Abhängigkeiten zwischen Wörtern [18]. Der Decoder wählt Keyphrases auf Basis der Dokumentrepräsentation aus, wobei die Dokumentrepräsentation während der Dekodierung dynamisch aktualisiert wird, indem die zuvor erzeugten Keyphrases berücksichtigt werden [18]. Zusätzlich wird verfolgt, welche Keyphrases bereits ausgewählt wurden [18]. Auf diese Weise sollen Wiederholungen oder sehr ähnliche Keyphrases vermieden und die Diversität der Keyphrases erhöht werden [18]. DivGraphPointer wurde auf Kp20k trainiert, einem Datensatz mit Papern aus dem Bereich der Informatik und den zugehörigen, von den jeweiligen Autoren erstellten Keyphrases [20].

Keyphrase-Generation Keyphrase-Generation-Verfahren können sowohl Present- als auch Absent-Keyphrases erzeugen [20]. Absent-Keyphrases sind nicht wortwörtlich im zugehörigen Text enthaltene Keyphrases [20]. Keyphrase-Generation ist deshalb von Interesse, da Absent-Keyphrases in Keyphrase-Datensätzen oft einen erheblichen Anteil bilden: Inspec [21] hat einen Anteil von 44,31% an Absent-Keyphrases, für Krapivin [22] sind es 52,26%, für NUS [23] 32,25% und für SemEval-2010 [24] 57,99% [20]. Im Folgenden werden zwei Keyphrase-Generation-Verfahren mit interessanten Ansätzen vorgestellt.

AutoKeyGen ist ein unüberwachtes Deep-Learning-Verfahren, das aus drei Phasen besteht: die Erstellung einer Phrasenbank („phrase bank“) für Absent-Keyphrases, Ranking der Keyphrase-Kandidaten und das Training eines Seq2Seq-Modells [25].

Die Idee hinter der Phrasenbank ist, dass Absent-Keyphrases des einen Dokuments Present-Keyphrases eines anderen Dokuments sein können [25]. Shen et al. zeigen für den Inspec-Datensatz, dass dies für 99% der Keyphrases gilt [25]. Daher werden für alle Dokumente die Present-Keyphrase-Kandidaten, die mittels regulärer Ausdrücke für Nominalphrasen extrahiert wurden, in einer Phrasenbank gesammelt [25]. Außerdem bemerken Shen et al., dass für 56,8% der Absent-Keyphrases alle Tokens dieser Keyphrases verstreut im zugehörigen Text vorkommen [25]. Somit können Absent-Keyphrases erzeugt werden, indem diejenigen Kandidaten-Keyphrases aus der Phrasenbank ausgewählt werden, deren Tokens allesamt im jeweiligen Dokument auftreten [25].

Die so ermittelten Keyphrase-Kandidaten müssen nun in eine Rangfolge gebracht werden [25]. Dafür wird das geometrische Mittel aus dem TF-IDF der Kandidaten-Keyphrases und der Kosinusähnlichkeit zwischen Einbettungen der Kandidaten und dem Dokument gebildet [25]. Für die Einbettungen wird Doc2Vec [26] eingesetzt [25].

Mit den auf diese Weise ermittelten Keyphrase-Kandidaten wird ein Seq2Seq-Modell bestehend aus einem BiLSTM-Encoder und einem LSTM-Decoder trainiert [25]. Die fünf bestbewerteten Present- und die fünf bestbewerteten Absent-Keyphrases eines Dokuments dienen als „Silberstandard“ („silver labels“) für das Training [25]. Um die Qualität der generierten Keyphrases zu verbessern, wird eine angepasste Dekodierstrategie eingesetzt: Wörter aus dem Eingabedokument werden mit doppelter Wahrscheinlichkeit generiert, Absent-Keyphrases können aber dennoch erzeugt werden [25]. Mit dem trainierten Modell können nun Keyphrases für ungesehene Dokumente erzeugt werden [25].

Wu et al. beschreiben ein Keyphrase-Generation-Verfahren, das mittels prompt-basiertem Lernen arbeitet [27]. Sie argumentieren, dass die gemeinsame Erzeugung von Present- und Absent-Keyphrases wie etwa bei Seq2Seq-Modellen, Absent-Keyphrases erzeugen kann, die für das zugehörige Dokument irrelevant sind [27]. Der Ansatz von Wu et al. nutzt vordefinierte Eingabe-Prompts, um die Generierung der Absent-Keyphrases gezielt zu steuern [27]. Die dahinterliegende Idee ist, dass viele Absent-Keyphrases einzelne Wörter enthalten, die auch im zugehörigen Dokument vorkommen [27].

Im ersten Schritt werden daher die überlappenden Wörter als Keywords extrahiert [27]. Basierend auf diesen Keywords werden Prompts erzeugt, die die Generierung der vollständigen Absent-Keyphrases steuern [27]. Die Prompts können entweder die Form „`phrase of kw is [MASK] [MASK] kw [MASK] [MASK]`“ oder „`other phrases are [MASK] [MASK] [MASK] [MASK]`“ mit `kw` als überlappendem Keyword annehmen [27]. Die `[MASK]`-Tokens werden zu einer gemeinsamen Absent-Keyphrase zusammengefügt [27]. Die erste Variante sorgt dafür, dass die so erzeugte Absent-Keyphrase das Keyword `kw` enthalten muss, während die zweite Variante auch die Generierung von Absent-Keyphrases erlaubt, die keine mit dem zugehörigen Dokument überlappenden Bestandteile hat [27]. Das Ersetzen der `[MASK]`-Tokens erfolgt parallel, somit werden die Tokens nicht autoregressiv nacheinander von links nach rechts, sondern in einem Schritt ersetzt, was für kürzere Laufzeiten sorgt [27]. Für diesen Ansatz wird Multi-Task-Training eingesetzt, sodass die Extraktion der überlappenden Keywords sowie die Erzeugung von Present- und Absent-Keyphrases gleichzeitig trainiert werden kann [27].

3 Problemanalyse

Um besser beurteilen zu können, welche Verfahren für die Erzeugung von Marginalien geeignet sein könnten und wie deren Parameter am besten gewählt werden sollten, sollen im Folgenden die Eigenschaften von Marginalien untersucht werden.

Der Duden beschreibt den Begriff der Marginalie als „handschriftliche Glosse, kritische Anmerkung o. Ä. in Handschriften, Akten oder Büchern“, sowie als „auf den Rand einer [Buch]seite gedruckter Verweis (mit Quellen, Zahlen, Erläuterungen o. Ä. zum Text)“ [28]. Marginalien können also recht verschiedenartig sein, denn zwischen einer kritischen Anmerkung, einer Quellenangabe und einer Erläuterung bestehen große Unterschiede, die potenziell auch unterschiedliche Lösungsansätze erforderlich machen. Es kann somit nicht allgemein von einer medien- und fachübergreifenden Uniformität innerhalb der Marginalien ausgegangen werden.

Für den Zweck dieser Arbeit soll der Fokus allerdings auf der für Lehrbücher erfahrungsgemäß typischen Form der Marginalien liegen, nämlich kurzen, stichwortartigen Beschreibungen des Gegenstandes des zugehörigen Textabschnitts[3]. Dafür sollen Lehrbücher aus dem Bereich der Informatik herangezogen werden, um die dort vorliegenden Eigenschaften von Marginalien untersuchen zu können und schließlich für diesen Anwendungsbereich passende Marginalien automatisch erzeugen zu können.

[3] Wenn im Folgenden von typischen Marginalien die Rede ist, sind Marginalien wie hier beschrieben gemeint. Ob diese tatsächlich „typisch“ sind, d.h. die überwiegende Mehrheit der Marginalien bilden, kann nur auf Basis anekdotischer Evidenz beantwortet werden, da hierfür keine entsprechende Statistik bekannt ist.

3.1 Erstellung eines Marginalien-Datensatzes

Die Erstellung des Marginalien-Datensatzes umfasst fünf Schritte:

1. Auffinden von Lehrbüchern mit Marginalien
2. Extraktion der Marginalien und zugehöriger Textabschnitte
3. Untersuchung der Marginalien-Arten
4. Bereinigen der extrahierten Daten
5. Vorbereitung der Analyse

Für das Auffinden von Lehrbüchern mit Marginalien hat sich das Verlagsprogramm der dpunkt.verlag GmbH als gute Quelle für zahlreiche Lehrbücher mit Marginalien herausgestellt. Diese sind zudem als HTML verfügbar, was für die Extraktion hilfreich ist. Über die Suche des O'Reilly-Verlages mit einer Beschränkung der Suchergebnisse auf Bücher des dpunkt.verlag und Deutsch als Sprache konnten so 39 deutschsprachige Lehrbücher aus dem Bereich der Informatik gefunden werden.

Für die Extraktion der Marginalien und zugehöriger Textabschnitte wurden Python-Skripte eingesetzt, welche in `extractFromHTML.ipynb` zu finden sind. Während die gesammelten Lehrbücher zwar insgesamt eine ähnliche HTML-Struktur aufweisen, sind die HTML-Elemente, die Marginalien enthalten, unterschiedlich hinsichtlich ihrer HTML-Tags (z.B. `<p>` oder `<span>`) und hinsichtlich der Attributwerte (z.B. `class="marginalia"`). Für die automatische Extraktion muss demnach manuell für jedes zu erfassende Buch HTML-Tag, Attributname und Attributwert der Marginalien erfasst werden. Diese Informationen werden für jedes Buch jeweils in einer Datei namens `marginaliaStructure.json` hinterlegt. Anschließend erfolgt die Extraktion von Marginalien, zugehöriger Textabschnitte sowie weiterer Informationen wie Kapitelnamen automatisch.

Die Extraktion der zugehörigen Textabschnitte folgt der Annahme, dass sämtlicher Text nach einer Marginalie zu dieser gehört, bis hin zur nächsten Marginalie oder Kapitelüberschrift. Dies birgt den möglichen Nachteil, dass gewisse Textabschnitte fälschlicherweise zu einer Marginalie gezählt werden, der Autor dies aber nicht so beabsichtigt hat. Die alternative Annahme, dass nur der auf eine Marginalie unmittelbar folgende Absatz zu dieser gehört, birgt wiederum den möglichen Nachteil, dass weniger Text als vom Autor beabsichtigt zur Marginalie gezählt wird. Aus der HTML-Struktur der Lehrbücher geht dies nicht hervor. Erstere Annahme wurde getroffen, sodass die Lösungsansätze eher auf „zu viele“ - möglicherweise aber auch irrelevante - Informationen zurückgreifen können, als auf „zu wenige“, mit möglicherweise fehlenden relevanten Informationen.

Eine weitere Annahme hinsichtlich der Extraktion ist das Ausschließen von Textabschnitten mit mehr als einer zugehörigen Marginalie. Diese Annahme soll potenziellen Lösungsansätzen zur Erzeugung von Marginalien die Entscheidung abnehmen, wie viele Marginalien erzeugt werden sollen. Da solch ein Fall ohnehin nur selten auftritt, werden diese nicht in den Datensatz mit aufgenommen.

Im nächsten Schritt wurden die unterschiedlichen Arten von Marginalien in den Lehrbüchern untersucht, damit diese Informationen für die Bereinigung des extrahierten Marginalien-Datensatzes eingesetzt werden können.

Bei der Untersuchung fällt auf, dass sowohl zwischen als auch innerhalb der verschiedenen Lehrbücher verschiedene Arten von Marginalien auftreten, die je nach Lehrbuch und Autor mal mehr und mal weniger einheitlich sein können. Im Folgenden werden Beispiele für die verschiedenen im Marginalien-Datensatz beobachteten Marginalien gegeben.

Die beobachteten Marginalien lassen sich in inhaltsbezogene und zweckbezogene Marginalien unterteilen. Die für Lehrbücher vermutlich typischste Art der inhaltsbezogenen Marginalie dient als Hinweis, wovon der zugehörige Textabschnitt handelt, ähnlich den Keyphrases. Dabei kann die Marginalie als Present-Marginalie als Ganzes wortwörtlich im Textabschnitt enthalten sein oder als Absent-Marginalie nicht im gleichen Wortlaut im Textabschnitt enthalten sein, sondern in anderen Worten formuliert (vgl. Tabelle 1).

Tabelle 1: Beispiele für Present- bzw. Absent-Marginalien

Text	**Marginalie und Typ**
Als ein nützliches Konzept bei der Beschreibung von Zugriffskontrollmechanismen hat sich die sogenannte Zugriffsmatrix erwiesen, in der die Zeilen die in einem System vorhandenen Subjekte und die Spalten die Objekte repräsentieren [Amo94]. Jede Zelle dieser Matrix enthält die Zugriffsrechte, die das durch die Zeile adressierte Subjekt auf das durch die Spalte adressierte Objekt hat (siehe hierzu auch Abbildung 9.2). *Aus :[29]*	**Zugriffsmatrix** *→Present-Marginalie*
Obwohl die eigentliche Durchführung eines Audits nur in seltenen Fällen dem Projektmanager obliegt, wird er praktisch immer direkt als Teilnehmer oder indirekt als Informationslieferant involviert sein. Im Projekt hat der Projektmanager die Aufgabe, dem Team Prozesse, Methoden und Werkzeuge zu vermitteln und deren Einhaltung bzw. korrekte Verwendung zu überwachen. Damit steht und fällt das Ergebnis der Auditdurchführung. *Aus: [30]*	**Rolle des Projektmanagers im Audit** *→Absent-Marginalie*

Tabelle 2 zeigt weitere Arten inhaltsbezogener Marginalien. Manche der Marginalien können Informationen enthalten, die über den zugehörigen Textabschnitt hinausgehen. In der ersten Zeile in Tabelle 2 wird das Akronym „PRM“ in der Marginalie erklärt, nicht jedoch im Textabschnitt. Die Marginalie in der zweiten Zeile fasst den Inhalt oder einen Teil des Inhalts des zugehörigen Textabschnitts ähnlich der Text-Summarization in Form eines ganzen Satzes zusammen.

Außerdem können inhaltsbezogene Marginalien auch in Form einer Frage wie in der dritten Zeile auftreten. Der zugehörige Textabschnitt kann die passende Antwort geben oder sich näher damit befassen, warum die Frage von Bedeutung ist. Im Gegensatz zu den vorherigen Arten der inhaltsbezogenen Marginalien, die sich nur auf den jeweils zugehörigen Textabschnitt beziehen, existieren zudem Marginalien, die Querbezüge zu vorherigen Marginalien oder einem größeren Kontext aufweisen, wie es in der vierten Zeile der Fall ist.

Tabelle 2: Beispiele für eine Marginalie mit Zusatzinformation, eine Marginalie als Zusammenfassung, eine Marginalie in Form einer Frage und eine abhängige Marginalie mit Querbezügen zu anderen Marginalien. Fortsetzung auf Seite 13.

Text	**Marginalie und Typ**
PRM und PAM gehören immer zusammen. Das PRM beschreibt die umzusetzenden Prozesse (Prozessgruppe, Prozessname, Ziel, Ergebnisse). Es kann sich dabei beispielsweise um einen bestehenden Standard wie z. B. ISO 12207 »Software Lifecycle Processes« handeln. Genau dieser Standard wird übrigens in ISO/IEC 15504 als Beispiel für ein PRM angeführt. *Aus: [30]*	**Prozess-Referenz-Modell (PRM)** *→Marginalie mit Zusatzinformation*
Oft wird unter Testen die (im Allgemeinen stichprobenartige) Ausführung der zu prüfenden Software (Testobjekt) auf einem Rechner verstanden. Dazu werden einzelne Testfälle ausgeführt, d.h., das Testobjekt wird mit Testdaten versehen und ausgeführt. Die anschließende Bewertung prüft, ob das Testobjekt die geforderten Eigenschaften erfüllt und sich konform zu den Anforderungen verhält. *Aus: [31]*	**Testen ist eine stichprobenhafte Prüfung** *→Marginalie als Zusammenfassung*

Text	Marginalie und Typ
Ein Fehler ist somit die Nichterfüllung einer festgelegten Anforderung, eine Abweichung zwischen dem Istverhalten (während der Ausführung der Tests oder des Betriebs festgestellt) und dem Sollverhalten (in der Spezifikation, den Anforderungen oder den User Stories festgelegt). Wann liegt aber ein nicht anforderungskonformes Verhalten des Systems vor? Im Gegensatz zu physischen Systemen entstehen Fehler in einem Softwaresystem nicht durch Alterung oder Verschleiß. Jeder Fehler ist seit dem Zeitpunkt der Entwicklung in der Software vorhanden. Er kommt jedoch erst bei der Ausführung der Software zum Tragen. *Aus: [31]*	**Was gilt als Fehler?** *→Marginalie als Frage*
Diese Befehle fügen an das Ende der Konfigurationsdatei des sogenannten Internet-Superdaemons ≫inetd≪ eine Zeile an, die auf dem TCP-Port 4545 eine passwortlose Root-Shell zur Verfügung stellt. Da nach jeder Konfigurationsänderung der inetd neu gestartet werden muss, startet der Angreifer diesen Dienst abschließend mit einem Kill-Signal neu. *Aus: [32]*	**5. Installation einer Root-Shell** *→Abhängige Marginalie*

Neben den inhaltsbezogenen Marginalien beschreiben zweckbezogene Marginalien den Zweck ihres zugehörigen Textabschnittes oder die strukturelle Rolle, die dieser innerhalb des Kapitels einnimmt. Die Marginalie „Beispiel“ in der ersten Zeile von Tabelle 3 gibt keinen Hinweis darauf, wovon der Textabschnitt handelt, sondern wozu dieser dient. In diesem Fall soll ein Beispiel für das im vorhergehenden Text besprochene Konzept gegeben werden. Weitere Beispiele für zweckbezogene Marginalien sind „Problem“ oder „Lösung“. Letztlich schließen sich inhalts- und zweckbezogene Marginalien nicht gegenseitig aus, sondern können auch kombiniert werden. Die Marginalie in der zweiten Zeile führt sowohl den Zweck (die Darstellung eines Problems) als auch den Inhalt/Gegenstand des Problems (die Änderung der GUI) auf.

Tabelle 3: Beispiele für eine rein zweckbezogene Marginalie und eine sowohl inhalts- als auch zweckbezogene Marginalie

Text	**Marginalie und Typ**
Angenommen Sie haben das Ziel, den durchschnittlichen Gewinn je Kunde bis Ende des Jahres von 10 Euro auf 15 Euro zu erhöhen. Ein CSF kann beispielsweise die Vermarktung eines neuen Produkts sein, das hoffentlich dazu führt, dass die Kunden mehr Geld im Unternehmen lassen. Der KPI kann in diesem Beispiel direkt den Erfüllungsgrad des Ziels adressieren. Er wäre der durchschnittliche Gewinn je Kunde. *Aus: [31]*	**Beispiel** →*Zweckbezogene Marginalie*
Eine Schwierigkeit besteht allerdings: Ändert sich im Zuge einer Programmkorrektur oder Programmerweiterung die Bedienoberfläche des Testobjekts zwischen zwei Testläufen, so kann es passieren, dass das ursprünglich aufgezeichnete Skript nicht mehr zur neuen Bedienoberfläche »passt«. Das Skript bleibt dann unter Umständen stehen, der automatisierte Testlauf bricht ab. [...] *Aus: [31]*	**Problem: Änderung der GUI** →*Inhalts- und zweckbezogene Marginalie*

Nach der Extraktion enthält der unbereinigte Datensatz 13.118 Elemente. Die Bereinigung umfasst einerseits die Entfernung von fehlerhaften Elementen, wie Duplikaten oder Elementen mit fehlenden Attributen und andererseits die Entfernung einiger zweckbezogener Marginalien (vgl. Tabelle 3). Zweckbezogene Marginalien haben zwar auch ihren Nutzen, es ist aber davon auszugehen, dass deren Erzeugung eine andere Herangehensweise bzw. Lösungsansätze erfordert, als es für inhaltsbezogene Marginalien der Fall ist. Zusätzlich ist anzunehmen, dass überwachte Verfahren durch homogenere Ground-Truths bessere Ergebnisse erzeugen können. Zweckbezogene Marginalien können jedoch nicht vollständig entfernt werden, da die Einordnung in inhalts- oder zweckbezogen manuell erfolgt. Die Marginalien werden daher anhand einer Liste häufiger zweckbezogener Marginalien entfernt.

Marginalien, die sowohl inhalts- als auch zweckbezogen sind (vgl. Tabelle 3), werden hingegen nicht entfernt. Das Entfernen anhand der Übereinstimmung mit den Einträgen einer Liste könnte aufgrund der wesentlich höheren Diversität nur wenige Marginalien stichprobenartig entfernen. Würden alle Marginalien entfernt, die Sub-Strings der Listeneinträge enthalten, könnten wiederum zu viele Marginalien entfernt werden, obwohl diese gar keinen Zweckbezug aufweisen. Zusätzlich haben bei inhalts- und zweckbezogenen Marginalien Lösungsansätze,

die inhaltsbezogene Marginalien erzeugen können, zumindest die Möglichkeit, den inhaltsbezogenen Teil der Marginalie zu generieren.

Des Weiteren werden Marginalien entfernt, die Zusammenfassungen des zugehörigen Textabschnitts darstellen (vgl. Tabelle 2). Diese gehören nicht zu den zuvor als für Lehrbücher typisch bezeichneten Marginalien und werden auf Basis derselben Überlegung wie bei den zweckbezogenen Marginalien entfernt. Da Zusammenfassungen üblicherweise ganze Sätze bilden, dient die Überprüfung, ob eine Marginalie mit einem Punkt endet, als einfache Heuristik für die Erkennung ganzer Sätze und stellt einen ersten, groben Filter dar. Möglicherweise werden dadurch allerdings mehr Marginalien als nötig entfernt. Ausrufe- und Fragezeichen werden dagegen nicht als Indikator für ganze Sätze verwendet, da Ausrufezeichen beispielsweise zur Betonung wichtiger Sachverhalte dienen können ("Datenleck!") und Fragezeichen bei Fragen verwendet werden, die im zugehörigen Textabschnitt beantwortet werden („Warum verschlüsseln?“). Diese Marginalien in Form von Fragen (vgl. Tabelle 2) werden nicht aus dem Datensatz entfernt, da sie zwar als Frage formuliert, aber dennoch inhaltsbezogen sind und zu den typischen Marginalien gezählt werden können. Als zweiter, präziserer Filter dient die Erkennung der Flexionsform von Verben in Marginalien mittels spaCy [4]. Von den Marginalien, die Verben enthalten, passieren nur die Marginalien mit Verben im Infinitiv den Filter. Unter der Annahme, dass ganze Sätze immer flektierte Verben enthalten, können dadurch ganze Sätze aussortiert werden, ohne Marginalien mit Verben pauschal zu entfernen.

Marginalien, die Zusatzinformationen enthalten (vgl. Tabelle 2) können ohne dahinterliegende Wissensbasis nicht von Lösungsansätzen erzeugt werden und können das Training überwachter Verfahren beeinträchtigen. Da diese Art Marginalien nicht ohne Weiteres automatisch als solche erkannt werden kann, können diese jedoch nicht aus dem Marginalien-Datensatz aussortiert werden. Abhängige Marginalien (vgl. Tabelle 2) enthalten zwar auch Zusatzinformationen in Form von Querbezügen zu anderen Marginalien oder Textabschnitten, diese sind Teil des Dokuments und erfordern in der Regel keine zusätzliche Wissensbasis.

Darüber hinaus wurden 500 Marginalien manuell daraufhin überprüft, ob diese überhaupt aus dem zugehörigen Textabschnitt hervorgehen können. Diejenigen, die unmöglich aus dem Textabschnitt hervorgehen, etwa weil die Marginalie externe Zusatzinformationen enthält oder einen persönlichen Kommentar des Autors darstellt, wurden entfernt und sind in `exluded_marginalia.txt` aufgelistet. Marginalien, die nicht aus dem Text hervorgehen können, würden beispielsweise das Training von Modellen zur automatischen Erzeugung von Marginalien erschweren. Eine manuelle Überprüfung des gesamten Datensatzes wäre wünschenswert, übersteigt jedoch den Zeitrahmen dieser Arbeit.

Im letzten Schritt werden die extrahierten und bereinigten Daten um weitere Informationen für die anschließende Analyse der Marginalien angereichert. Das bedeutet, dass für die Analyse nötige Informationen bereits im Datensatz

[4] https://spacy.io/
Diese und alle nachfolgenden URLs in dieser Arbeit wurden zuletzt am 1. Februar 2025 abgerufen.

hinterlegt werden und diese Informationen während der Analyse lediglich abgerufen und in Form entsprechender Diagramme dargestellt werden müssen. Dazu gehört die Bestimmung der Parts-of-Speech (POS) der Marginalien mittels spaCy, sodass deren Verteilung über den Datensatz untersucht werden kann. Dafür werden einige Annahmen und Vereinfachungen getroffen.

Zum einen wird nicht zwischen Eigennamen, Nomen und dem POS-Tag „X“, das für sonstige POS steht, differenziert. Die Unterscheidung zwischen Eigennamen und Nomen ist für die statistische Untersuchung der POS im Rahmen dieser Arbeit spezifischer als benötigt. Das POS-Tag „X“ wird zur Vereinfachung als Nomen behandelt, da es im Marginalien-Datensatz oft für englischsprachige Nomen auftritt. Im Bereich der Informatik ist viel englische Terminologie geläufig, für die deutschsprachigen Lehrbücher wird allerdings ein spaCy-Modell für die deutsche Sprache. Der Nachteil dieser Vereinfachung ist eine mögliche Fehlkategorisierung von Wörtern mit dem POS-Tag „X“ als Nomen, bei denen es sich aber eigentlich nicht um Nomen handelt. Andererseits ist das Ziel der Analyse der Marginalien hinsichtlich POS nicht hundertprozentige Genauigkeit, sondern ein Überblick über die Verteilung der POS. Eine stichprobenartige Überprüfung von Wörtern mit dem POS-Tag „X“ zeigte nur seltene Fehlkategorisierungen. Dadurch ist anzunehmen, dass dies keine maßgebliche Verzerrung der POS-Verteilung bewirkt.

Analog zu Eigennamen und Nomen wird außerdem nicht zwischen neben- und unterordnenden Konjunktionen unterschieden. Zusätzlich werden Zeichensetzung und Leerzeichen ignoriert.

Für die spätere Analyse wird zudem überprüft, wie viele Marginalien im zugehörigen Text selbst enthalten sind und an welcher Stelle im Textabschnitt sie auftreten. Dafür werden die Marginalien und die Textabschnitte mittels spaCy lemmatisiert, nach vollständigen Treffern gesucht und die Trefferpositionen im Marginalien-Datensatz persistiert.

3.2 Analyse des Marginalien-Datensatzes

Im Folgenden werden Marginalien und zugehörige Textabschnitte aus dem Marginalien-Datensatz nach verschiedenen Aspekten untersucht. Die dafür eingesetzten Python-Skripte sind in `marginalia_dataset_scripts/marginalia Stats.ipynb` zu finden.

Länge der Marginalien Abb. 1 zeigt die Längenverteilung der vorliegenden Marginalien. Hierbei fällt auf, dass die Marginalien mehrheitlich aus wenigen Wörtern bestehen. Bei den Present-Marginalien ist eine deutlich stärkere Tendenz zu Marginalien, die nur aus einem Wort bestehen, ersichtlich (Abb. 2). Ein möglicher Faktor für diesen Unterschied in der Verteilung zwischen Marginalien allgemein und Present-Marginalien könnte in der Ermittlung der Present-Marginalien liegen: Die Wahrscheinlichkeit, dass Marginalien aus nur einem Wort im Textabschnitt enthalten sind, dürfte größer sein, als dass Marginalien aus z.B. fünf Wörtern in genau dieser Wortreihenfolge und ununterbrochen im Textabschnitt enthalten sind.

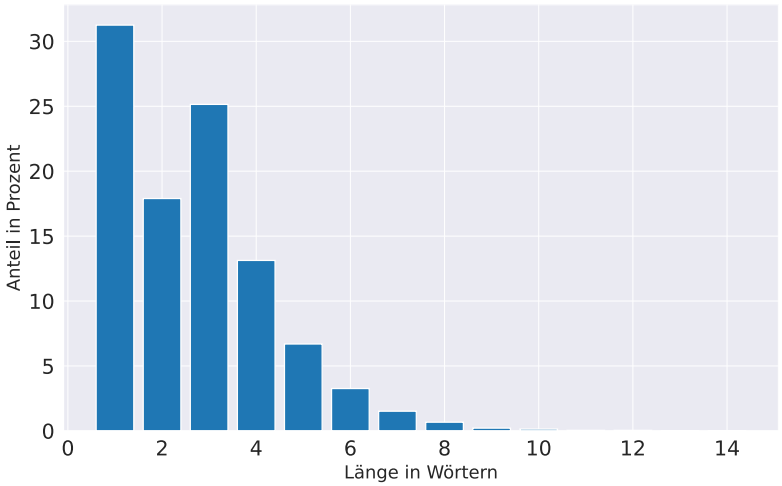

Abb. 1: Relative Häufigkeit der Länge aller Marginalien in Wörtern

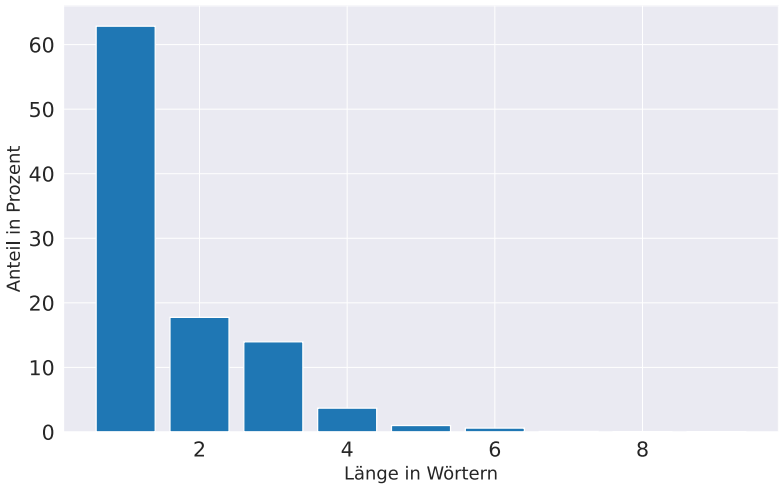

Abb. 2: Relative Häufigkeit der Länge der Present-Marginalien in Wörtern

Länge der Textabschnitte Die den Marginalien zugehörigen Textabschnitte weisen eine durchschnittliche Länge von 133,32 Wörtern auf, mit einem Median von 100 Wörtern (Abb. 3). Dabei ist die in Kapitel 3.1 erwähnte Annahme zur Bestimmung der zugehörigen Textabschnitte zu beachten, die potenziell die wahre Länge der zugehörigen Textabschnitte überschätzt.

Die Länge der Textabschnitte ist für die Auswahl der Lösungsverfahren von Interesse, da verschiedene Ansätze der Keyphrase-Prediction für Datensätze unterschiedlicher Dokumentenlängen evaluiert werden. Die Annahme ist dabei, dass ein Verfahren, welches für kurze Dokumente gute Ergebnisse erzielt, besser für den ebenfalls vornehmlich aus kurzen Dokumenten bestehenden Marginalien-Datensatz geeignet ist, als ein Verfahren, welches für lange Dokumente gut abschneidet.

Der Inspec-Datensatz beispielsweise besteht aus Paper-Abstracts mit einer durchschnittlichen Länge von 128,2 Wörtern [13], bewegt sich also in einem dem Marginalien-Datensatz ähnlichen Bereich. Der Krapivin-Datensatz dagegen besteht aus vollständigen Papers, mit einer durchschnittlichen Länge von 8040,74 Wörtern [13]. Eine Übersicht häufig verwendeter Datensätze für Keyphrase-Prediction findet sich z.B. in [13,5].

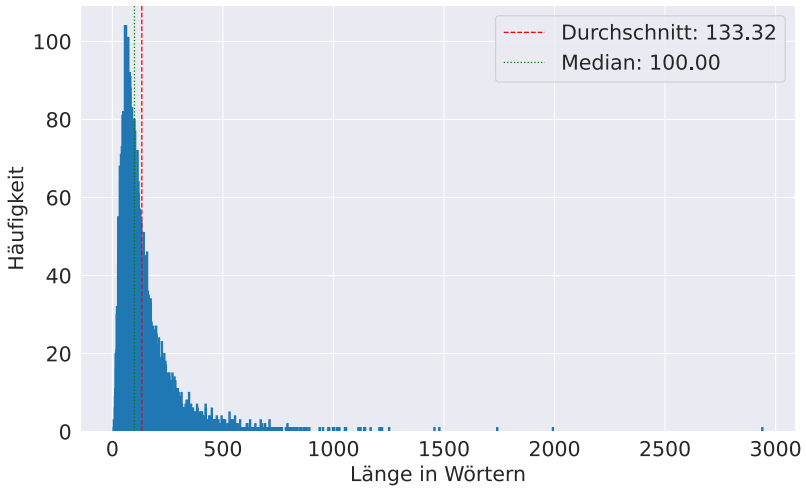

Abb. 3: Längenverteilung der den Marginalien zugehörigen Textabschnitte

POS der Marginalien Abb. 4 zeigt die relative Häufigkeit verschiedener POS-Kombinationen über alle Marginalien. Eine POS-Kombination bezeichnet dabei die zu den Wörtern einer Marginalie gehörigen POS. Die POS-Kombination für „Zugriff auf das SUT“ wäre demnach „NOUN ADP DET NOUN“.

Dabei ist ersichtlich, dass 31,5% der Marginalien aus nur einem Nomen bestehen, während 24,6% der Marginalien seltene POS-Kombinationen mit jeweils weniger als 1% aufweisen. Betrachtet man dagegen die Verteilung der POS-Kombinationen unter den Present-Marginalien in Abb. 5, verdoppelt sich der Anteil an Marginalien aus nur einem Nomen fast, während die seltenen POS-Kombinationen auf 8,5% sinken. Unter den Present-Marginalien gibt es also deutlich weniger Varianz hinsichtlich der POS-Kombinationen.

Die Häufigkeit verschiedener POS-Kombinationen ist für die Auswahl eines Lösungsverfahrens relevant, da manche Verfahren eine Kandidatenauswahl anhand der POS vornehmen. Aufgrund der unterschiedlichen Verteilung der POS-Kombinationen zwischen allen Marginalien und Present-Marginalien wären beispielsweise für Keyphrase-Extraction andere POS-Kombinationen für die Kandidatenauswahl vorteilhaft als für Keyphrase-Generation.

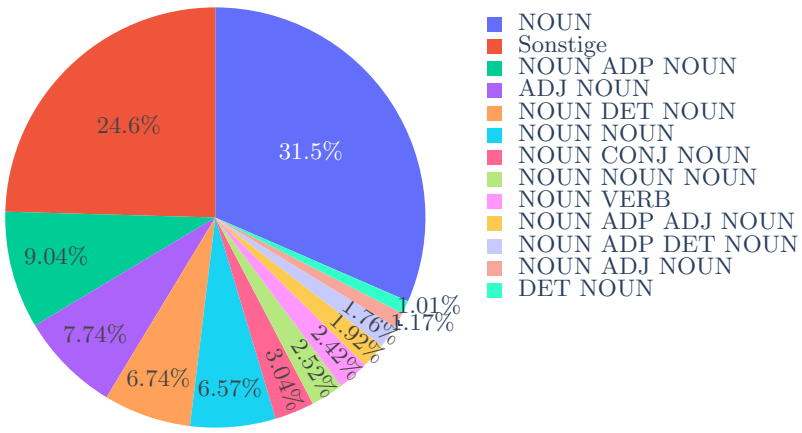

Abb. 4: Verteilung der POS-Kombinationen der Marginalien über den gesamten Datensatz. Die Kategorie „Sonstige“ subsumiert die POS-Kombinationen, die einen Anteil von jeweils weniger als 1% besitzen.

Anteil der Present-Marginalien Etwa ein Drittel der Marginalien sind Present-Marginalien (Abb. 6). Dies deutet darauf hin, dass zumindest für den Marginalien-Datensatz Keyphrase-Extraction in einem Drittel der Fälle erfolgreich sein kann. Erfolgreich in dem Sinne, dass die vom Verfahren erzeugte Marginalie mit der vom Autor erstellten Marginalie übereinstimmt. Allerdings bedeutet dies nicht

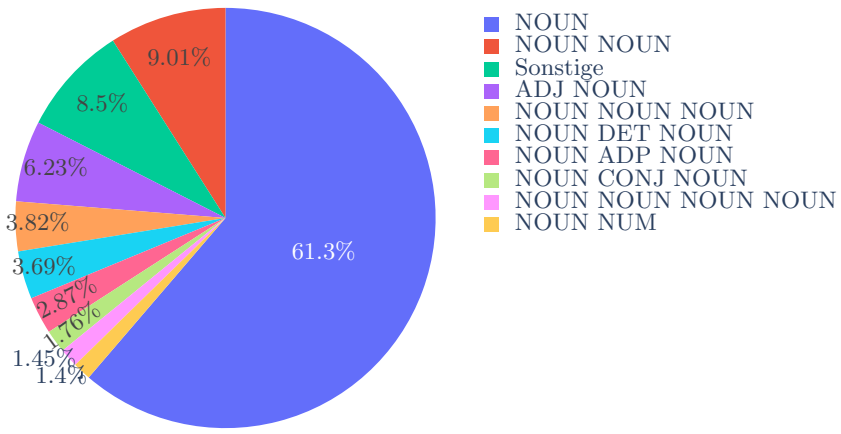

Abb. 5: Verteilung der POS-Kombinationen der Present-Marginalien. Die Kategorie „Sonstige“ subsumiert die POS-Kombinationen, die einen Anteil von jeweils weniger als 1% besitzen.

zwangsläufig, dass in zwei Dritteln der Fälle Keyphrase-Extraction nicht eingesetzt werden kann. Es ist möglich, dass Keyphrase-Extraction in mehr als nur einem Drittel der Fälle gute Marginalien erzeugt, auch wenn diese nicht mit den Marginalien der Autoren übereinstimmen.

Position der Present-Marginalien Aus Abb. 7 geht hervor, dass Present-Marginalien besonders häufig in den ersten 5% des Textabschnittes auftreten. Dies ist für diejenigen Verfahren der Keyphrase Prediction relevant, welche die Position der Kandidaten in deren Bewertung einbeziehen.

Auch hier ist die Auswahl eines Lösungsverfahrens ratsam, dessen Evaluierung für einen Datensatz gute Ergebnisse erzielt, dessen Keyphrases ebenfalls vermehrt am Anfang des Dokuments auftreten. Bei einem Datensatz wie Krapivin, der vollständige Paper enthält, ist mit einer höheren Auftretenswahrscheinlichkeit von Keyphrases sowohl am Anfang (Abstract und Introduction) als auch gegen Ende des Dokuments (Conclusion) zu rechnen. Im Marginalien-Datensatz hingegen treten die Present-Marginalien ausschließlich zu Beginn des Textabschnittes besonders häufig auf.

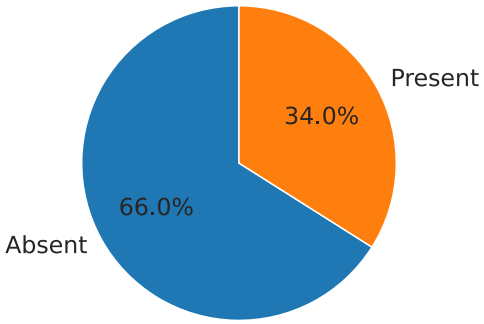

Abb. 6: Anteil der Present- und Absent-Marginalien

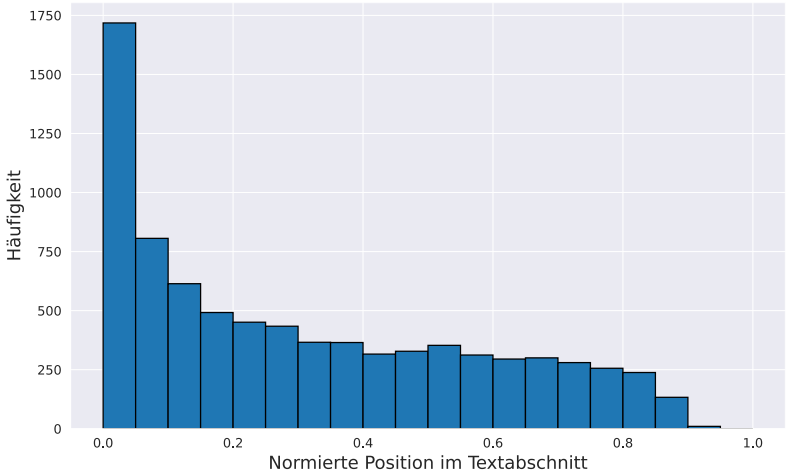

Abb. 7: Häufigkeitsverteilung der Positionen der Present-Marginalien in den Textabschnitten. Die Position wird in Anzahl an Wörtern ab Beginn des Textabschnittes gemessen und mittels Division durch die Länge des Textabschnittes normiert. Jeder Balken umfasst 5% des Textabschnitts, somit zeigt der zweite Balken von links die Häufigkeit der Present-Marginalien innerhalb der ersten 5% bis exklusive 10%.

4 Lösungsansätze

Gemäß der Analyse in Kapitel 3.2 besteht ein Unterschied hinsichtlich der POS-Kombinationen zwischen Marginalien allgemein und Present-Marginalien im Speziellen. Zusätzlich stellen Present-Marginalien nur etwas mehr als ein Drittel der Marginalien dar. Beides legt den Einsatz sowohl von extraktiven (Keyphrase-Extraction) als auch generativen (Keyphrase-Generation) Verfahren nahe.

Während es denkbar ist, dass auch ein generatives Verfahren alleine ausreichend ist, um Present- und Absent-Marginalien zu erzeugen, soll im Folgenden dennoch beides, extraktive und generative Verfahren, ausgewählt werden. Dadurch soll zum einen untersucht werden, ob extraktive Verfahren im Falle des Marginalien-Datensatzes tatsächlich nur in etwa einem Drittel der Fälle gute Marginalien erzeugen können und generative Verfahren zusätzlich oder anstelle dessen zwingend notwendig sind. Zum anderen sind insbesondere die traditionellen extraktiven Verfahren sehr leichtgewichtig und benötigen kein Training.

Xie et al. untersuchten in einer 2023 veröffentlichten Studie [5] zahlreiche Verfahren der extraktiven und generativen Keyphrase-Prediction. Unter den generativen Verfahren erzielte WR-SetTrans für Present-Keyphrases in der Mehrheit der Fälle die besten F_1@5- und F_1@M-Werte über fünf verschiedenen Datensätzen [5]. Für Absent-Keyphrases erreicht WR-SetTrans ebenfalls mehrheitlich die besten Werte, wobei allerdings KG-KE-KR-M [33] im NUS- und KP20k-Datensatz bessere Werte erreicht [5]. Während WR-SetTrans für den hinsichtlich der Länge der Dokumente mit dem Marginalien-Datensatz vergleichbaren Inspec-Datensatz die besten Werte erreicht, übertrifft wiederum KG-KE-KR-M für den ebenfalls längenmäßig ähnlichen KP20k-Datensatz alle anderen generativen Verfahren [5].

Vorab ist nicht abzusehen, welches der beiden Verfahren für den Marginalien-Datensatz besser geeignet ist, insbesondere, da die Aufgabe der Keyphrase-Prediction nicht zwangsläufig mit der Erzeugung von Marginalien gleichzusetzen ist. Zudem ist das F_1-Maß als rein lexikalische Messgröße nur bedingt für die Bewertung der Qualität der erzeugten Keyphrases geeignet, wie in Kapitel 6 näher betrachtet wird. Da WR-SetTrans allerdings hinsichtlich der Vorhersage von Present-Keyphrases in den meisten Fällen besser ist als KG-KE-KR-M [5], wird WR-SetTrans als generatives Verfahren ausgewählt.

Hinsichtlich der extraktiven Verfahren erzielten Xie et al. zufolge die auf Deep-Learning basierenden die besten Ergebnisse [5]. Als Vergleich zum generativen WR-SetTrans soll SIFRank+ [34] als bestes unüberwachtes Deep-Learning-Verfahren eingesetzt werden [5]. Im Gegensatz zum überwachten WR-SetTrans ist demnach kein Training notwendig, wodurch SIFRank+ einfacher einsatzbereit ist und die Ergebnisse der Evaluierung potenziell besser übertragbar auf domänenfremde Datensätze sind als bei WR-SetTrans [5]. Zusätzlich verwendet SIFRank+ eine Gewichtung der Kandidaten anhand ihrer Position im Text, sodass Kandidaten zu Beginn des Textes stärker gewichtet werden [34]. Diese Gewichtung ist auch für den Marginalien-Datensatz aufgrund der in Kapitel 3.2 festgestellten Position der Present-Marginalien in den Textabschnitten plausibel.

Abseits der speziell für Keyphrase-Prediction entwickelten Verfahren können Large-Language-Models (LLMs) im Bereich der NLP inzwischen große Fortschritte verzeichnen. Daher sollen zwei LLMs für die Erzeugung von Marginalien eingesetzt werden: GPT-4o[5] und mT5 [35]. GPT-4o gilt oft als sehr mächtiges LLM, die Inferenz ist aber programmatisch nur kostenpflichtig über eine API möglich. mT5 ist dagegen frei verfügbar. mT5 und GPT-4o sollen einen beispielhaften Einblick geben, inwieweit LLMs im Bereich der Erzeugung von Marginalien gegenüber den für Keyphrase-Prediction konzipierten Verfahren abschneiden. Da der Schwerpunkt der Betrachtung hier auf Keyphrase-Prediction-Verfahren liegt, soll jeweils nur ein knapper Überblick über mT5 und GPT-4o gegeben werden.

Der Ansatz mT5 für die Erzeugung von Marginalien zu verwenden, basiert auf Transfer Learning. Für Transfer Learning wird ein Modell auf eine bestimmte Aufgabe mit einem großen Datensatz vortrainiert, um anschließend Fine-Tuning anhand eines eigenen, üblicherweise kleineren Datensatzes durchzuführen. Es existieren frei verfügbare, auf Text-Summarization vortrainierte mT5-Modelle, selbst für deutsche Texte. Text-Summarization wird deshalb gewählt, damit das LLM auf eine Aufgabe vortrainiert ist, die eine möglichst große Ähnlichkeit zum Problem der Erzeugung von Marginalien hat (vgl. Kapitel 2). Zwar wäre ein Training auf einem deutschen Keyphrase-Datensatz wahrscheinlich noch besser geeignet, jedoch konnte weder ein solcher Datensatz noch ein entsprechendes bereits vortrainiertes Modell gefunden werden. Da Text-Summarization ganze Sätze für die einzelnen Textabschnitte erzeugt, soll Fine-Tuning mit dem Marginalien-Datensatz erfolgen. Auf diese Weise soll mT5 lernen, Ergebnisse statt in ganzen Sätzen im Format der Marginalien auszugeben. mT5 ist anders als T5 [36] nicht bereits auf einen Downstream-Task wie Text-Summarization trainiert, wurde allerdings im Gegensatz zu T5 für die deutsche Sprache sowie 100 weitere Sprachen vortrainiert [35]. Daher muss für diesen Ansatz ein durch Dritte auf Text-Summarization deutscher Texte vortrainiertes mT5 verwendet werden, welches anschließend Fine-Tuning durchläuft.

Alle vorherigen Lösungsansätze beziehen für die Erzeugung der Marginalien ausschließlich den jeweiligen Textabschnitt mit ein, für den die Marginalie erzeugt werden soll. Da die Textabschnitte jedoch nicht abgeschlossen für sich alleine stehen, sondern sich in einem Kontext zueinander (z.B. in einem gemeinsamen Kapitel) befinden, könnte der Einbezug vorheriger und nachfolgender Textabschnitte für die Erzeugung guter Marginalien hilfreich sein. Dies soll über GPT-4o mit einfacher Retrieval-Augmented-Generation (RAG) erfolgen. Dafür soll für jeden Textabschnitt eine Anfrage über die API gestellt werden, die neben dem Textabschnitt selbst auch den im Kapitel vorhergehenden Textabschnitt samt Marginalie und den nachfolgenden Textabschnitt enthält.

Im Folgenden werden die ausgewählten Lösungsansätze näher vorgestellt.

[5] https://openai.com/index/hello-gpt-4o/

4.1 SIFRank+

SIFRank ist ein unüberwachtes Keyphrase-Extraction-Verfahren, das SIF [37] als Modell für Einbettungen von Sätzen mit ELMo [38] als vortrainiertes Sprachmodell kombiniert [34]. SIFRank+ stellt die Erweiterung um die Gewichtung nach der Position der Kandidaten dar [34]. Mit Ausnahme der Gewichtung sind SIFRank und SIFRank+ gleich [34], weshalb der Übersichtlichkeit halber im Folgenden ausschließlich von SIFRank+ die Rede ist.

Die Vorverarbeitung von SIFRank+ umfasst die Tokenisierung des Eingabedokuments sowie POS-Tagging [34]. Anhand der POS-Informationen werden anschließend Nominalphrasen als Kandidaten ausgewählt, während das vortrainierte Sprachmodell ELMo aus den Tokens Worteinbettungen erstellt [34].

Das Ziel von SIFRank+ ist es, die Ähnlichkeit der mittels SIF erzeugten Einbettungen der Kandidaten und des Dokuments zu vergleichen, da dies unter den Annahmen von SIF letztlich der Ähnlichkeit zwischen der Kandidaten-Keyphrase und dem Thema des Dokuments entspricht [34].

SIF steht dabei für „Smooth Inverse Frequency“, da die Worteinbettungen mittels $a/(a + p(w))$ gewichtet werden, wobei a ein Parameter ist und $p(w)$ die geschätzte Häufigkeit des Wortes w [37]. Die dahinterliegende Idee ist, dass häufige Wörter wie „the“ weniger Gewicht erhalten sollten, da sie wenig zur Bedeutung beitragen, sondern unabhängig vom Gegenstand des Dokuments oder Satzes auftreten [37]. Für die Kandidaten und die Sätze des Dokuments wird jeweils ein auf diese Weise gewichteter Durchschnittsvektor bestimmt und anschließend die Projektionen dieses Durchschnittsvektors auf dessen ersten singulären Vektor entfernt [37]. Dieser erste singuläre Vektor ist Ergebnis einer Singulärwertzerlegung, die eine Matrix (hier aus Satzeinbettungen) in Komponenten zerlegt. Die erste Komponente bzw. der erste singuläre Vektor repräsentiert hier einen gemeinsamen Diskursvektor c_0, der mit Syntax oder häufig auftretenden Wörtern wie „just“ oder „but“ in Verbindung zu stehen scheint [37].

Die mittels SIF erzeugten Einbettungen der Nominalphrasen und des Dokuments werden nun per Kosinusdistanz auf ihre Ähnlichkeit untersucht und erhalten einen Score von 0 bis 1 [34]. Je näher der Score der Nominalphrasen an der 1 ist, desto relevanter werden diese hinsichtlich des Themas des Dokuments eingestuft [34]. Der finale Score für eine Nominalphrase als Kandidat für eine Keyphrase ergibt sich durch die Multiplikation der Kosinusdistanz mit einer Gewichtung $p(NP_i)$ nach der Position der ersten Nennung einer Nominalphrase NP_i im Dokument [34]. $p(NP_i) = 1/(p_1 + \mu)$, wobei p_i die Position der Nominalphrase in der Liste aller Kandidaten ist (relative Position der ersten Nennung) und μ ein Hyperparameter zur Optimierung des Positionsgewichts [34]. Anschließend wird eine Softmax-Funktion eingesetzt, um die Gewichtungsunterschiede zwischen benachbarten Kandidaten zu vermindern [34].

Zusätzlich beachtet SIFRank+, dass die Wahrscheinlichkeitsverteilung von Wörtern abhängig von der Domäne ist, in der sie auftreten [34]. Es ist anzunehmen, dass der Begriff „Algorithmus“ in Texten aus der Informatik eine höhere Auftretenswahrscheinlichkeit besitzt als in Texten der Literaturgeschichte. Daher wird bei der eingangs erwähnten Gewichtung der Wörter nach ihrer inversen

Häufigkeit $a/(a + p(w))$ zwischen der Gewichtung basierend auf Wikipedia als allgemeinem Korpus und der Gewichtung basierend auf dem spezifischen Korpus der jeweiligen Domäne unterschieden [34]. Die gewichtete Summe beider ergibt letztlich die finale Gewichtung [34].

Die N besten (ähnlichsten) Kandidaten-Keyphrases werden als Ergebnis ausgegeben [34].

4.2 WR-SetTrans

WR-ONE2SET ist ein Trainings-Paradigma für Keyphrase-Generation, dessen Implementierung WR-SetTrans ein transformerbasiertes Verfahren darstellt [39]. Da WR-ONE2SET eine Weiterentwicklung von ONE2SET [40] darstellt, wird zum besseren Verständnis im Folgenden zunächst ONE2SET vorgestellt, um anschließend die darauf aufbauenden Verbesserungen seitens WR-ONE2SET zu erläutern.

ONE2SET ist eines der drei vorherrschenden Trainings-Paradigmas für Keyphrase-Generation und soll Probleme des ONE2SEQ-Paradigmas beheben, welches Keyphrase-Generation als Sequence-Generation-Problem modelliert [5]. Gemäß ONE2SEQ existiert für jeden Eingabetext eine nach ihrem ersten Auftreten im Text geordnete Sequenz an Present Keyphrases [41]. Absent Keyphrases werden am Ende der Sequenz angehängt [41]. Diese Ordnung innerhalb der Sequenz an Keyphrases kann jedoch zu einem Problem führen: Ermittelt ein Modell während des Trainings zwar die korrekten Keyphrases, aber in einer anderen Reihenfolge als in der Target-Sequenz, kann dies dennoch zu einem großen Training Loss führen [40].

Diese und weitere Nachteile von ONE2SEQ soll ONE2SET beheben, indem zu jedem Dokument statt einer geordneten Sequenz eine (ungeordnete) Menge an Keyphrases gehört [40]. ONE2SEQ verwendet eine Transformer-Architektur, wobei N gelernte Steuer-Codes („control codes“) als zusätzliche Eingabe für den Decoder dienen [40]. Jeder dieser Steuer-Codes ist für die Erzeugung einer Keyphrase oder eines speziellen $\emptyset$-Tokens verantwortlich [40]. Das $\emptyset$-Token bedeutet, dass keine zugehörige Keyphrase existiert [40]. Die Steuer-Codes ermöglichen somit einerseits eine parallele Generierung einer Menge an Keyphrases ohne vorbestimmte Reihenfolge und andererseits kann eine dynamische Anzahl an Keyphrases erzeugt werden, da die entsprechenden Steuer-Codes statt „überzähliger“ Keyphrases $\emptyset$-Tokens generieren können [40].

Für das Training von ONE2SET stellen die Steuer-Codes eine Herausforderung dar: Jeder Steuer-Code soll parallel eine Keyphrase erzeugen, es ist jedoch nicht im Voraus bekannt, welcher Steuer-Code für welche konkrete Keyphrase zuständig ist [40]. Die Kreuzentropie-Funktion setzt eine solche eindeutige Zuordnung zwischen Vorhersage und Target jedoch voraus. Um dies zu lösen, setzt ONE2SET einen Mechanismus namens K-Step Target Assignment ein [40].

Das K-Step Target Assignment versucht, die optimale Zuordnung zwischen den generierten und den Target-Keyphrases zu finden, bevor der eigentliche Loss berechnet wird [40]. Zu Beginn erzeugt das Modell für jeden der Steuer-Codes

K Keyphrases [40]. Anschließend wird bipartites Matching zwischen den erzeugten Keyphrases und den Target-Keyphrases durchgeführt [40]. Beim bipartiten Matching wird jedem Steuer-Code aus einer Menge an erzeugten Keyphrases genau eine (optimale) Target-Keyphrase aus der Menge an Target-Keyphrases zugeordnet [40].

Ye et al. gehen davon aus, dass die Vorhersage von Present- und Absent-Keyphrases unterschiedliche Fähigkeiten erfordert [40]. Daher wird das bipartite Matching für die Hälfte der Steuer-Codes mit Present-Keyphrases als Target und die andere Hälfte mit Absent-Keyphrases als Target durchgeführt [40]. Daraus resultiert jeweils eine separate Loss-Funktion für Present- und für Absent-Keyphrases [40].

Neben den Vorteilen von ONE2SET wie etwa der Vermeidung von fälschlich hohem Training Loss aufgrund einer abweichenden Reihenfolge der Keyphrases, einer höheren Diversität der erzeugten Keyphrases oder einer höheren Effizienz aufgrund paralleler Generierung von Keyphrases, ist ein wesentlicher Nachteil die Überschätzung des $\emptyset$-Tokens [39]. Das $\emptyset$-Token wird also übermäßig oft erzeugt [39]. Xie et. al identifizieren zwei Ursachen: Übermäßige Verwendung des $\emptyset$-Token als Padding und Target-Keyphrases der Steuer-Codes, sowie die Instabilität des K-Step Target Assignment [39]. Diese Instabilität führt dazu, dass denselben Steuer-Codes in unterschiedlichen Iterationen abwechselnd Target-Keyphrases und $\emptyset$-Tokens zugewiesen werden, wodurch die Überschätzung des $\emptyset$-Token noch weiter verstärkt wird [39].

WR-ONE2SET ist eine Weiterentwicklung von ONE2SET und zielt darauf ab, die Wahrscheinlichkeiten der Keyphrase Generation besser zu kalibrieren, sodass das $\emptyset$-Token nicht überschätzt wird [39]. Dies soll durch Adaptive Instance-Level Cost Weighting und Target Re-Assignment erreicht werden [39].

Adaptive Instance-Level Cost Weighting führt einen adaptiven Gewichtungsfaktor λ_{adp} ein, der dynamisch für jede Trainingsinstanz skaliert wird [39]. λ_{adp} bestimmt sich aus dem Verhältnis zwischen der Wahrscheinlichkeit des ersten Tokens des zugewiesenen Targets $\hat{p}^i\left(y_0^{m(i)}\right)$ und der Wahrscheinlichkeit $\hat{p}^i(\emptyset)$ des $\emptyset$-Tokens für jeden Steuer-Code i in einer Instanz, der eine Keyphrase als Target hat [39]. Ist es wahrscheinlicher, dass ein Steuer-Code eine Keyphrase statt eines $\emptyset$-Tokens erzeugt ($\hat{p}^i\left(y_0^{m(i)}\right) > \hat{p}^i(\emptyset)$), wird das Verhältnis auf 1 begrenzt, um zu verhindern, dass die Kosten der nicht-überschätzten Steuer-Codes zu stark reduziert werden [39]. Die Hyperparameter λ_{pre} und λ_{abs} aus ONE2SET, die den Training-Loss für Present- und Absent-Keyphrases anpassen sollen, werden nun in WR-ONE2SET jeweils mit λ_{adp} multipliziert [39]. Auf diese Weise können im Gegensatz zu den (konstanten) Hyperparametern λ_{pre} und λ_{abs} aus ONE2SET dynamisch während des Trainings Anpassungen erfolgen [39].

Das Target Re-Assignment soll die Instabilität des K-Step Target Assignment beheben [39]. Für alle Steuer-Codes werden drei Keyphrase-Arten unterschieden: Die Target-Keyphrase, die K Token-Vorhersagen, sowie die K nicht-$\emptyset$-Vorhersagen [39]. Die Steuer-Codes werden anhand dieser Keyphrase-Arten in zwei Mengen eingeordnet: potenzielle und unwichtige Steuer-Codes [39].

Potenzielle Steuer-Codes besitzen das Potenzial, eine neue Keyphrase zu generieren [39]. Dabei handelt es sich um Steuer-Codes, denen das $\emptyset$-Token als Target zugewiesen wurde, deren nicht-$\emptyset$-Vorhersage jedoch nicht mit den K Vorhersagen aller anderen Steuer-Codes übereinstimmt [39]. Unwichtige Steuer-Codes sind dagegen diejenigen, denen das $\emptyset$-Token zugewiesen wurde, deren nicht-$\emptyset$-Vorhersage jedoch bereits in den K Vorhersagen mindestens eines anderen Steuer-Codes vorkommt [39].

Das Target Re-Assignment weist nun jedem potenziellen Steuer-Code die beste passende Target-Keyphrase zu, während den unwichtigen Steuer-Codes keine Target-Keyphrase zugewiesen wird [39].

4.3 mT5 mit Fine-Tuning

mT5 ist eine multilinguale Variante des Text-to-Text Transfer Transformers T5 und modelliert ebenfalls alle (NLP-) Aufgaben als Text-zu-Text-Problem [35]. Dadurch kann sowohl maschinelle Übersetzung als auch Sentimentanalyse auf dieselbe Weise trainiert werden [35].

mT5 basiert auf einem Encoder-Decoder-Transformer [35]. Als Trainingsdatensatz dient mC4, ein 101 Sprachen umfassender Datensatz erstellt aus Common Crawl Web Scrapes [35].

mT5 ist vortrainiert auf Span-Corruption, wofür zufällige Sequenzen an Eingabetokens maskiert werden und mT5 die Rekonstruktion dieser maskierten Tokens lernen soll [35]. Span-Corruption ist allerdings auch die einzige Aufgabe, auf die mT5 vortrainiert wurde [35]. Das auf Englisch beschränkte T5 dagegen wurde zusätzlich auf verschiedene Aufgaben wie Summarization oder Übersetzung vortrainiert [36] und kann ohne weiteres Training oder Fine-Tuning für diese Aufgaben eingesetzt werden. mT5 muss vor der Verwendung für Text-Summarization erst dafür vortrainiert werden.

Liegt ein für Text-Summarization vortrainiertes mT5 vor, kann Fine-Tuning mit dem Marginalien-Datensatz durchgeführt oder ein Adapter genutzt werden. Die Entscheidung zwischen beiden Alternativen hängt maßgeblich von der Größe des Datensatzes ab [42]. Zhao und Chen untersuchen in ihrer 2022 veröffentlichten Studie [42], ab welcher Größe des Trainingsdatensatzes Fine-Tuning oder die Verwendung eines Adapters bessere Ergebnisse erzielt. Für den XL-Sum-Datensatz unter Verwendung eines BART-Modells ist ab etwa 30.000 Trainingsbeispielen Fine-Tuning signifikant besser [42]. Für kleinere Datensätze zwischen 300-3.000 Trainingsbeispielen sind die beiden Alternativen ähnlich gut, für noch kleinere Datensätze erzielen Adapter bessere Ergebnisse [42]. Für den Marginalien-Datensatz mit seinen 11.808 Trainingsbeispielen ist Fine-Tuning daher wahrscheinlich eine geeignete Wahl.

4.4 GPT-4o mit RAG

GPT-4o ist ein kommerzielles, multimodales LLM der Firma OpenAI. Von OpenAI zum Veröffentlichungszeitpunkt selbst als Flaggschiff des firmeneigenen Repertoires bezeichnet, ermöglicht GPT-4o nicht nur die Ein- und Ausgabe von

Text, sondern auch von Ton und Bildern [43]. Details zur Architektur, Training u.ä. nennt OpenAI nicht. Mögliche Gründe dafür stehen vermutlich - ähnlich wie für GPT-4 argumentiert wurde [44] - im Zusammenhang mit Wettbewerbs- und Sicherheitsaspekten.

RAG kann eingesetzt werden, um LLMs bei Anfragen zu unterstützen, die über die Trainingsdaten hinausgehen und dadurch das Auftreten von faktisch inkorrekten Antworten („Halluzinationen“) vermindern [45]. Dazu wird die Anfrage an das LLM um relevante Informationen einer externen Wissensbasis erweitert [45]. Dies birgt in der Praxis oft Herausforderungen wie effizientes Auffinden relevanter Informationen aus der Datenbasis oder sinnvolle Integration der externen Informationen in die Anfrage [45]. Zur Vereinfachung wird für den Anwendungsfall der Erzeugung von Marginalien naives RAG eingesetzt. Um die Erzeugung der Marginalie zu unterstützen, wird die Anfrage für einen bestimmten Textabschnitt um zusätzlichen Kontext in Form des vorhergehenden und nachfolgenden Textabschnitts erweitert. Falls vorhanden, wird für den vorhergehenden Textabschnitt die zugehörige Marginalie angegeben. Dadurch soll die Konsistenz in der Formulierung und der Art der Marginalien verbessert werden. Vorhergehende Marginalien können somit auf diese Weise ihre Nachfolger beeinflussen. Das birgt allerdings auch den möglichen Nachteil, dass schlechte Marginalien ihre Nachfolger negativ beeinflussen.

5 Implementierung

Im Folgenden wird die Implementierung der Lösungsansätze aus Kapitel 4 beschrieben. Für SIFRank+ und WR-SetTrans stehen Implementierungen ihrer Autoren zur Verfügung, sind aber auf englische Texte (vor-)trainiert und erfordern Anpassungen und Training für die deutsche Sprache und den Marginalien-Datensatz. mT5 erfordert Fine-Tuning mit dem Marginalien-Datensatz und GPT-4o erfordert die Erstellung entsprechender Prompts mit RAG und Skripte für die Verwendung der API. Außerdem sollen statistische Verfahren als Baselines eingesetzt werden. Für diese Baseline-Verfahren sind ebenfalls kleinere Anpassungen für die deutsche Sprache und den Marginalien-Datensatz nötig.

5.1 Baseline-Verfahren

Die Python-Bibliothek pke[6] ermöglicht die einfache Verwendung verschiedener Keyphrase-Extraction-Verfahren, die als Baseline für die vorgestellten Verfahren dienen sollen. Dazu gehören TF-IDF, YAKE!, SingleRank, TextRank, PositionRank, TopicRank und MultipartiteRank (fortan: MPRank). Bis auf TF-IDF werden diese bereits im Rahmen des Codes zu SIFRank+ von Giarelis et al. [13] zur Verfügung gestellt.

TF-IDF benötigt eine Auflistung der Dokumentenhäufigkeit für alle Wörter des Korpus, d.h. alle Wörter des Marginalien-Datensatzes. Dies geschieht in

[6] https://github.com/boudinfl/pke

`create_document_frequency_file.py`. Für dieses und die übrigen Baseline-Verfahren wird eine durch pke bereitgestellte deutsche Stoppwortliste verwendet. Die TF-IDF-Implementierung von pke erlaubt die Auswahl von Uni- bis Trigrammen als Keyphrase-Kandidaten, sodass angesichts der Längenverteilung der Marginalien im Marginalien-Datensatz (vgl. Abb. 1) Trigramme gewählt wurden.

SingleRank, TopicRank, TextRank und MPRank ermöglichen es, Keyphrase-Kandidaten auf bestimmte POS zu beschränken. Dafür werden die im Rahmen der Analyse des Marginalien-Datensatzes identifizierten POS der Marginalien (vgl. Abb. 5) spezifiziert, die 90,1% der Present-Marginalien umfassen. Ohne diese Anpassung wären die Keyphrase-Kandidaten auf Nomen, Eigennamen und Adjektive beschränkt.

Für PositionRank können die Keyphrase-Kandidaten mittels einer Grammatik ausgewählt werden, die ebenfalls anhand der in Abb. 5 identifizierten POS-Kombinationen der Present-Marginalien erstellt wurde. Im Gegensatz zu den anderen Baseline-Verfahren können somit nicht nur die einzelnen Bestandteile der Keyphrase-Kandidaten in Form von POS, sondern auch die Struktur der Keyphrase-Kandidaten spezifiziert werden.

Für YAKE! ist außer der Angabe der Sprache und der Länge der Keyphrase-Kandidaten keine Anpassung nötig. Die Länge wurde auf vier Wörter begrenzt, da dies 98,25% der Present-Marginalien abdeckt.

5.2 SIFRank+

Die Autoren von SIFRank+ stellen ihren Code auf Github[7] zur Verfügung. Für diese Arbeit wird allerdings die im Rahmen der 2024 von Giarelis et al. veröffentlichten Studie verwendete, leicht angepasste Implementierung eingesetzt. Giarelis et al. vergleichen in [13] verschiedene State-of-the-Art Keyphrase-Extraction-Verfahren, zu denen auch SIFRank+ zählt. Der ebenfalls auf Github[8] veröffentlichte Code von Giarelis et al. umfasst neben SIFRank+ auch die zuvor genannten Baseline-Verfahren.

Die Änderungen der SIFRank+ Implementierung seitens Giarelis et al. beschränken sich gemäß der Readme des SIFRank-Verzeichnisses auf das Entfernen hartkodierter Dateipfade, sowie die Verwendung der CPU für das Ausführen von SIFRank+.

Für die Ausführung von SIFRank+ sind einige Vorbereitungen nötig: Installation von Bibliotheken, Anpassen von Dateipfaden, Erstellen eines Vokabulars und das Hinzufügen des Marginalien-Datensatzes.

SIFRank+ erfordert Python 3.6, eine schon ältere Python-Version aus dem Jahr 2016. Durch Verwendung eines Docker-Containers können diese und weitere Abhängigkeiten auf einfache und replizierbare Weise gehandhabt werden. Ein entsprechendes Dockerfile wurde der Implementierung hinzugefügt.

[7] https://github.com/sunyilgdx/SIFRank
[8] https://github.com/NC0DER/KeyphraseExtraction

Darüber hinaus wird eine Options- und Weights-Datei für ELMo benötigt, die separat heruntergeladen werden muss. Verlinkt werden von den SIFRank-Autoren die Dateien für die englische Sprache, für die Marginalien wird aber ein deutsches ELMo benötigt. In der Readme findet sich eine URL zum Download der entsprechenden deutschen ELMo-Dateien.

SIFRank+ verwendet wie viele andere für die englische Sprache konzipierte Keyphrase-Prediction-Verfahren den Porter-Stemmer, für die deutsche Sprache müssen daher Anpassungen vorgenommen werden. Anstelle des Porter-Stemmers wird Lemmatisierung mittels spaCy durchgeführt, sowie eine deutsche Stoppwortliste verwendet.

Für die in Kapitel 4.1 beschriebene Gewichtung der Wörter sind zwei Wortlisten erforderlich, die Wörter samt ihrer absoluten Auftretenshäufigkeit enthalten. Die eine Wortliste soll domänenspezifisch sein und somit die Häufigkeit der Wörter des Marginalien-Datensatzes aufführen, die andere soll die generischen Häufigkeiten der Wörter der deutschen Sprache enthalten. `term_frequencies.ipynb` erstellt beide Wortlisten. Da SIFRank+ eine englische generische Wortliste auf Basis der englischen Wikipedia verwendet, wird die deutsche generische Wortliste anhand der Auftretenshäufigkeit der Wörter der deutschen Wikipedia [9] erstellt.

Der Marginalien-Datensatz befindet sich unter SIFRank/marginalia. Jeder Textabschnitt des Datensatzes, zu dem Marginalien erzeugt werden sollen, ist hier als einzelne Textdatei aufgeführt.

Analog zur Grammatik für PositionRank werden für SIFRank - statt ausschließlich Nominalphrasen zu betrachten - die in `posList.txt` spezifizierten POS-Kombinationen für die Ermittlung der Keyphrase-Kandidaten verwendet.

Nun kann SIFRank+ ausgeführt werden. Da SIFRank gegenüber SIFRank+ keine zusätzlichen Anpassungen benötigt, wird SIFRank für die Evaluierung ebenfalls miteinbezogen. Dadurch können ebenfalls 90,1% der Present-Marginalien abgedeckt werden.

5.3 WR-SetTrans

Die Autoren von WR-SetTrans stellen ihren Code auf Github[10] zur Verfügung. WR-SetTrans benötigt für das Training die Aufbereitung des Marginalien-Datensatzes im korrekten Format. Dafür wird der Marginalien-Datensatz aufgeteilt in Target-, Validation- und Test-Splits im Verhältnis von 80% zu 10% zu 10%. Während in der Dokumentation von WR-SetTrans Details zum korrekten Format fehlen, finden sich diese in der Dokumentation von ONE2SET, das ebenfalls auf Github[11] zugreifbar ist. Die Trainings-Splits werden jeweils aufgeteilt in Targets und Source.

Die Targets (`*_trg.txt`) enthalten pro Zeile die Target-Marginalie (vom Autor des Buches gewählte Marginalie) für ein Trainingsbeispiel. `<peos>` dient als

[9] https://github.com/IlyaSemenov/wikipedia-word-frequency
[10] https://github.com/DeepLearnXMU/WR-One2Set
[11] https://github.com/jiacheng-ye/kg_one2set

Trenner zwischen Present- und Absent-Marginalien. Da es je Trainingsbeispiel nur eine Target-Marginalie gibt, enden Present-Marginalien mit `<peos>` und Absent-Marginalien beginnen mit `<peos>`.

Jede Zeile der Source-Dateien (`*_src.txt`) repräsentiert ein Trainingsbeispiel und besteht aus dem Titel des jeweiligen Unterkapitels und dem Textabschnitt, zu dem die Marginalie erzeugt werden soll. `<eos>` fungiert als Trenner zwischen beiden. Zusätzlich werden Zahlen durch `<digit>` ersetzt.

Um die spätere Vergleichbarkeit der Lösungsansätze zu verbessern, muss die Aufteilung in Train- und Validation-Splits denen von mT5 entsprechen. Das Skript für die Aufteilung in Splits für WR-SetTrans und mT5, sowie die Aufbereitung des Marginalien-Datensatzes für WR-SetTrans findet sich in `marginalia_dataset_scripts/prepare_data.ipynb`.

Auch für WR-SetTrans wurde die Stammformbildung mittels Porter-Stemmer durch Lemmatisierung mittels spaCy ersetzt.

Das Training auf dem Marginalien-Datensatz wurde in einem Docker-Container durchgeführt und ohne Fehlermeldungen abgeschlossen. Statt die erzeugten Marginalien zu enthalten, ist allerdings sowohl `predictions.txt` als auch `predictions_filtered.txt` leer. Die Ursache dafür konnte nicht identifiziert werden. Zu den möglichen Ursachen zählen etwa ein eigener Implementierungsfehler (z.B. fehlerhafte Aufbereitung des Datensatzes) oder Versionsinkompatibilitäten.

Im Zuge der Fehlersuche wurde schließlich getestet, ob der Vorgänger SetTrans (Implementierung von ONE2SET) Marginalien erzeugt oder ebenfalls leere Ergebnisse zurückgibt. Tatsächlich erzeugt SetTrans Marginalien, jedoch nicht für alle Textabschnitte. Für diese Fälle ist nicht klar, ob ausschließlich ∅-Tokens erzeugt wurden oder ein Fehler vorliegt.

Da WR-SetTrans aufgrund unbekannter Ursachen keine Marginalien erzeugt, wird stattdessen SetTrans hilfsweise verwendet.

5.4 mT5 mit Fine-Tuning

Das Fine-Tuning von mT5 erfolgt in einem Docker-Container und benötigt, anders als die vorherigen Ansätze, keine Aufbereitung des Marginalien-Datensatzes in ein bestimmtes Format.

Für das Fine-Tuning wird jedoch ein auf Text-Summarization vortrainiertes mT5-Modell benötigt. Für die deutsche Sprache existieren auf HuggingFace[12] verschiedene Modelle. Da nicht im Voraus ersichtlich ist, welches der Modelle am besten geeignet sein wird, wird das aktuellste Modell mit den meisten Downloads, sowie ein weiteres, auf einem anderen Datensatz vortrainiertes mT5-Modell ausgewählt. Das erstere Modell `mT5-small-sum-de-en-v2`[13] (fortan kurz: mt5-sum) wurde für die deutsche Sprache auf den MLSUM-German-Datensatz trainiert, der Nachrichtenartikel der Süddeutschen Zeitung samt Zu-

[12] https://huggingface.co/
[13] https://huggingface.co/T-Systems-onsite/mt5-small-sum-de-en-v2

sammenfassung[14] enthält [46], sowie auf einen weiteren, inzwischen nicht mehr zugreifbaren Datensatz. Das letztere Modell `mt5-small-finetuned-amazon-en-de`[15] (fortan kurz: mT5-amazon) wurde auf dem Multilingual Amazon Reviews Corpus[16] trainiert, welcher Produktrezensionen auf Amazon umfasst. Die Rezensionen bestehen unter anderem aus einem Titel der Rezension und dem Textkörper.

Das Fine-Tuning beider Verfahren erfolgte jeweils mit dem Skript `mT5.py` für 5, 10 und 15 Epochen, wobei Stichproben zufolge das Fine-Tuning für 5 Epochen die jeweils besten Ergebnisse erzielte. Ebenfalls anhand von Stichproben wurde mT5-amazon dabei als geeigneter eingeschätzt. Eine mögliche Ursache dafür, warum mT5-amazon den Stichproben zufolge bessere Ergebnisse erzielt, könnte der für das Fine-Tuning für Text-Summarization verwendete Multilingual Amazon Reviews Corpus sein. mT5-amazon fasst den Titel der Rezension als Zusammenfassung auf, die aus dem Textkörper vorhergesagt werden soll. Diese Titel von Rezensionen sind oft keine ganzen Sätze und könnten somit besser für die Erzeugung von Marginalien geeignet sein. Ein möglicher Nachteil könnte jedoch die eher legere Art der Sprache oder Umgangssprache, wie sie in Rezensionen vorkommen kann, während die Marginalien aus Lehrbüchern mit eher akademischer Sprache stammen.

Die Beurteilung der Qualität der Ergebnisse von mt5-sum und mT5-amazon beruht lediglich auf Stichproben und kann keine umfassende Evaluierung und Vergleich der Modelle ersetzen. Da das Ziel dieser Arbeit der Vergleich verschiedener Ansätze ist, muss die Auswahl eines geeigneten mT5-Modells auf Basis von Stichproben getroffen werden. Zukünftig wäre eine nähere Untersuchung wünschenswert, die verschiedene mT5-Modelle auf ihre Tauglichkeit für die Erzeugung von Marginalien untersucht.

5.5 GPT-4o mit RAG

OpenAI stellt eine API für programmatischen Zugang zu GPT-4o bereit [47]. Der Einsatz von GPT-4o mit RAG für die Erzeugung von Marginalien beschränkt sich auf die Erzeugung von Prompts, die anschließend über die API an die Server von OpenAI geschickt und verarbeitet werden. Der Code zu diesem Kapitel befindet sich in `gpt.ipynb`.

Im Rahmen dieser Arbeit wurde kein Prompt Engineering durchgeführt, um die bestmögliche Prompt zu finden, allerdings wurden verschiedene Alternativen stichprobenartig getestet und verglichen. Die besten Ergebnisse konnten mit folgender Prompt erzielt werden:

> *Die folgenden Absätze stammen aus dem Unterkapitel „Unterkapitel“ im Kapitel „Kapitel“ im Lehrbuch „Buchtitel“. Sofern vorhanden, wird zu*

[14] Genauer: Beschreibungen oder Highlights („Teaser“) der Nachrichtenartikel werden als Zusammenfassung behandelt [46].
[15] https://huggingface.co/anibahug/mt5-small-finetuned-amazon-en-de
[16] https://huggingface.co/datasets/defunct-datasets/amazon_reviews_multi

jedem Absatz die zugehörige Marginalie angegeben.:

Absatz N-1 lautet: „Text des Absatzes“

Absatz N lautet: „Text des Absatzes“

Absatz N+1 lautet: „Text des Absatzes“

Aufgabe:
Erstelle eine Marginalie für Absatz N. Sofern vorhanden, kann der vorherige Absatz N-1, sowie der nachfolgende Absatz N+1 möglicherweise als Kontext für die Erzeugung in sich konsistenter Marginalien (z.B. einheitliche Art der Formulierung der Marginalien) hilfreich sein. Wichtig: Die Marginalie soll sich ausschließlich auf den Inhalt von Absatz N beziehen. Die Marginalie muss nicht unbedingt den Inhalt des Absatzes zusammenfassen, sondern kann auch lediglich einen Hinweis darauf geben, wovon der Absatz handelt.

Hinweise zur Länge der gesuchten Marginalie:
Die meisten Marginalien sind zwischen 1 bis 4 Wörter lang, nur sehr wenige sind über 6 Wörter lang. Halte die Marginalien so kurz wie möglich.

Hinweis zum Ausgabeformat:
Gib ausschließlich die erzeugte Marginalie ohne Präfix wie „Marginalie:“ oder Anführungszeichen aus.

GPT-4o wird also explizit um die Erzeugung von Marginalien gebeten. Zuvor wurde erfolgreich getestet, ob GPT-4o das Konzept der Marginalien „versteht“. Auf die Frage nach der Definition von Marginalien erklärte GPT-4o das Konzept von Marginalien übereinstimmend mit der in dieser Arbeit getroffenen Definition typischer Marginalien.

Tatsächlich können zu einer Marginalie mehrere Absätze gehören, weshalb der Begriff „Textabschnitt“ in der Prompt zutreffender wäre. Zur Vereinfachung der Prompt wird der Begriff „Absatz“ synonym verwendet, da die potenziell mehreren Absätze zu einem einzigen Absatz konkateniert werden. Dies liegt daran, dass die Unterteilung des Textabschnitts in Absätze nicht immer korrekt im Marginalien-Datensatz hinterlegt ist. Dort besteht der Textabschnitt einer Marginalie aus einem Array aus Strings. Jeder dieser Strings ist der Text eines HTML-Elements, diese stellen zwar teils, aber nicht immer einen Absatz dar.

Um Textabschnitte N-1 und N+1 für den gegebenen Textabschnitt N zu finden, muss der Text des zugehörigen Unterkapitels nach dem Text von Absatz N durchsucht werden, um vorhergehende und nachfolgende Textabschnitte ermitteln zu können. Dies ist nötig, da der Marginalien-Datensatz nur Marginalien mit ihrem Textabschnitt (und Unterkapitel) enthält, nicht aber Textabschnitte, denen keine Marginalie zugeordnet ist.

Für den Test-Split schlägt diese Suche in 12 Fällen und für den gesamten Datensatz in 118 Fällen fehl. Die Ursache konnte nicht identifiziert werden. Es ist jedoch nicht anzunehmen, dass diese 12 bzw. 118 fehlenden Ergebnisse dieinin Evaluierung signifikant beeinflussen.

Wird ein Absatz N-1 im Marginalien-Datensatz gefunden, kann die zugehörige Marginalie in der Prompt aufgeführt werden. Andernfalls werden bis zu fünf vorherige Strings aus dem Array des zugehörigen Unterkapitels verwendet. Die Festlegung auf fünf Strings soll sicherstellen, dass der auf diese Weise erhaltene Text ausreichend lang ist, um als Kontext dienen zu können.

Für den Textabschnitt N+1 wird analog nach einem entsprechenden Eintrag im Marginalien-Datensatz gesucht. Für diesen Textabschnitt wird keine Marginalie in der Prompt aufgeführt, da in einem realen Anwendungsfall ohne bereits bekannte Marginalien bei sequenzieller Erzeugung der Marginalien für nachfolgende Textabschnitte noch keine Marginalie erzeugt worden wäre. Bei erfolgloser Suche wird lediglich der nächste String des Arrays des zugehörigen Unterkapitels herangezogen. Die Annahme ist dabei, dass Textabschnitt N+1 für die Erzeugung der Marginalie weniger wichtig ist, da bei einem menschlichen Autor bei sequenzieller Erzeugung die vorherigen Textabschnitte notwendigerweise bereits gelesen sein müssen, während vermutlich weniger oft mehrere nachfolgende Textabschnitte „vorab“ gelesen werden. Zusätzlich führt diese Annahme zu kürzeren Prompts und somit geringeren Kosten.

OpenAI monetarisiert die Benutzung der API auf Basis von Tokens [48]. Für GPT-4o belaufen sich die Kosten auf 2,50$ pro eine Million Input-Tokens, sowie 10$ pro eine Million Output-Tokens [48]. Da die erstellten Prompts Marginalien mit nur wenigen Wörtern erzeugen sollen, können die Kosten für die Output-Tokens vernachlässigt werden. Da die Prompts aber aus mehreren hundert Wörtern bestehen können, ist die Nutzung der Batch API von Vorteil, da für diese nur Kosten in halber Höhe anfallen [48].

Die Batch API ermöglicht die asynchrone Verarbeitung mehrerer Anfragen in Batches [49]. Die Ergebnisse werden innerhalb von 24 Stunden zur Verfügung gestellt [49], je nach Auslastung können die Ergebnisse aber auch schon innerhalb weniger Minuten bereitstehen. Für die Batch API gelten allerdings zwei Einschränkungen: Zum einen dürfen die einzelnen Batches maximal je 90.000 Tokens umfassen, zum anderen können maximal 90.000 Tokens gleichzeitig in Bearbeitung sein [50]. Es können also nicht alle Batches vorab an den Server geschickt und anschließend der Reihe nach verarbeitet werden.

Um mit diesen Einschränkungen umzugehen, werden die Prompts zuerst für alle Elemente des Marginalien-Datensatzes erstellt und in einer großen JSONL-Datei gespeichert. Anschließend wird diese Datei auf Grundlage der Token-Anzahl in kleinere Batches unterteilt. Durch diesen Zwischenschritt kann bei Bedarf auf die Batch API verzichtet werden und Anfragen direkt synchron über die API gestellt werden. Anschließend werden die Batches in einer Schleife nacheinander an den Server zur Verarbeitung geschickt. Nach Abschicken eines Batches wird dessen Bearbeitungsstatus minutenweise abgefragt. Ist ein Batch erfolgreich bearbeitet, können die Ergebnisse abgefragt, gespeichert und der nächste

Batch abgeschickt werden. Dabei kommt es oft vor, dass ein Batch zwar schon fertig verarbeitet ist, die Batch API aber dennoch meldet, dass nur 90.000 Tokens gleichzeitig in Bearbeitung sein dürfen. Dadurch kommt es teils zu längeren Wartezeiten zwischen der Verarbeitung von Batches.

Insgesamt umfassen die Prompts aller Elemente des Marginalien-Datensatzes 10.075.771 Tokens, was Kosten in Höhe von 13,23$ für die Input-Tokens und 0,38$ für die Output-Tokens verursacht hat. In diesen Kosten sind anfängliche Tests mit inbegriffen.

6 Evaluierung

Im Folgenden werden die vorgestellten Lösungsansätze quantitativ und qualitativ evaluiert. Der Einbezug einer qualitativen Evaluierung ist deshalb wichtig, da die quantitativen Messgrößen nur eingeschränkt aussagekräftig sind.

6.1 Quantitative Evaluierung

Wu et al. schlagen vier Qualitätskriterien für Keyphrase Prediction vor: Reference Agreement, Faithfulness, Diversity und Utility [51]. Reference Agreement misst, wie sehr die erzeugten Keyphrases mit den Referenz-Keyphrases übereinstimmen [51]. Faithfulness beschreibt, inwieweit die erzeugten Keyphrases faktisch mit dem zugehörigen Text übereinstimmen [51]. Denkbar wären hier etwa LLM-basierte Verfahren, die Informationen „halluzinieren“. Diversity soll zeigen, wie semantisch unterschiedlich die erzeugten Keyphrases sind [51]. Utility bemisst, wie nützlich die erzeugten Keyphrases für Ad-Hoc-Retrieval sind, d.h. inwieweit die Retrieval-Leistung verbessert wird [51].

Für die Erzeugung von Marginalien ist Reference Agreement ein sinnvolles Qualitätskriterium. Ebenso ist Faithfulness für alle Lösungsansätze relevant, da selbst extraktive Verfahren Marginalien erzeugen können, die den zugehörigen Textabschnitt falsch darstellen. Für den fiktiven Textabschnitt „Bevor wir uns mit Absent Keyphrases befassen, wollen wir uns Verfahren für die Extraktion von Present Keyphrases anschauen: [...]“ wäre die Keyphrase „Absent Keyphrases“ irreführend, da dieser Textabschnitt gar nicht davon handelt. Utility kann im Falle der Marginalien auch ein relevantes Kriterium darstellen, da die Marginalien auch dazu dienen sollen, relevante Textabschnitte schnell zu identifizieren. Diversity ist hier hingegen weniger relevant, da stets nur eine Marginalie pro Textabschnitt erzeugt werden soll. Für den Rahmen dieser Arbeit soll die quantitative Evaluierung auf Reference Agreement als Qualitätskriterium beschränkt werden. Die qualitative Evaluierung kann implizit allerdings auch Hinweise auf Faithfulness geben.

Die quantitative Evaluierung erfolgt auf dem Test-Split des Marginalien-Datensatzes, der 1182 Marginalien umfasst. Da SetTrans und mT5-amazon auf dem Train-Split des Marginalien-Datensatzes trainiert worden sind, hat die Evaluierung für diese Verfahren auf dem Test-Split zu erfolgen, damit keine Marginalien für bereits „gesehene“ Textabschnitte erzeugt werden und die Ergebnisse verfälscht werden. Für bessere Vergleichbarkeit werden auch die restlichen

unüberwachten Verfahren auf dem Test-Split evaluiert. Die zur Ermittlung der Messgrößen verwendeten Python-Skripte finden sich in `marginalia_dataset_scripts`.

Fortan bezeichnet mT5-amazon das durch das Fine-Tuning auf dem Marginalien-Datensatz erhaltene Modell.

F_1-Maß Für die Evaluierung von Keyphrase-Verfahren wird oft das F_1-Maß herangezogen [6]:

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6}$$

Stimmen die von den Verfahren erzeugten Marginalien (fortan: System-Marginalien) mit den „wahren“ (von den Autoren erdachten) Marginalien (fortan: Referenz-Marginalien) überein, werden diese als Treffer bezeichnet. Somit bezeichnet die Precision hier das Verhältnis der Anzahl an Treffern zur Anzahl insgesamt erzeugter Marginalien und der Recall das Verhältnis der Anzahl an Treffern zur Anzahl an möglichen Treffern.

Wird das F_1-Maß in einer Publikation verwendet, wird üblicherweise F_1@k mit $k \in \{5, 10, 15\}$ aufgeführt, welches das F_1-Maß für die k besten Keyphrases bezeichnet. F_1@M repräsentiert das F_1-Maß für alle M erzeugten Keyphrases.

Eine Keyphrase wird in den meisten Publikationen genau dann als Treffer angesehen, wenn diese (meist nach der Stammformbildung) komplett mit der Referenz-Keyphrase übereinstimmt, die sogenannte Exact Match Evaluation [6]. Dies kann jedoch dazu führen, dass synonyme oder sehr bedeutungsähnliche Keyphrases ebenso als „falsch“ gelten, wie völlig abwegige Keyphrases.

Abb. 8 zeigt die Anzahl der Exact Matches für die verschiedenen Verfahren. Dabei führen mT5-amazon und GPT-4o mit großem Abstand. Aus der Abbildung lässt sich außerdem ableiten, dass die Gewichtung nach der Position der Kandidaten im Text bei SIFRank+ einen deutlichen Vorteil für die Anzahl der Exact Matches gegenüber SIFRank darstellt. Bemerkenswerterweise ist TF-IDF hinsichtlich der Anzahl an Exact Matches eine starke Baseline und übertrifft als nicht auf Keyphrase-Extraction spezialisiertes Verfahren die restlichen Keyphrase-Extraction-Verfahren deutlich.

Dem gegenüber ist Approximate Matching [52] eine einfache Möglichkeit, auch dann System-Marginalien als Treffer zu werten, wenn diese nicht komplett mit der Referenz-Marginalie übereinstimmen. Dabei gilt eine System-Marginalie auch dann als Treffer, wenn sie nach der Stammformbildung die Referenz-Keyphrase als Substring enthält oder wenn sie in der Referenz-Keyphrase als Substring enthalten ist [52]. Es existieren verschiedene andere Alternativen zum Exact Matching. Approximate Matching stellt hier aber eine einfach implementierbare Möglichkeit dar, um einen Vergleich zum „strengen“ Exact Matching zeigen zu können.

Das F_1-Maß kann nur lexikalische Ähnlichkeit der erzeugten Marginalien gegenüber den wahren Marginalien messen. Sowohl semantische Ähnlichkeit als auch grammatikalische und syntaktische Korrektheit können auf diese Art nicht

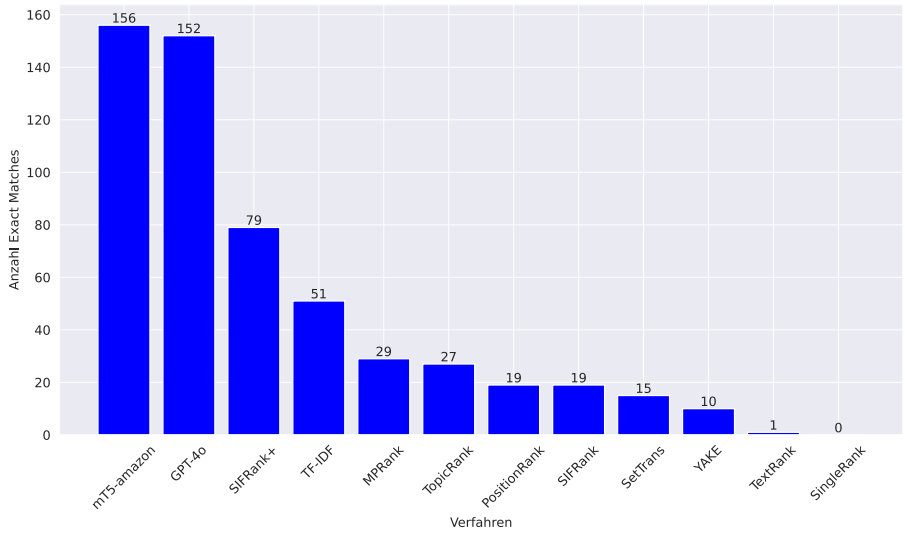

Abb. 8: Anzahl der Exact Matches auf dem Test-Split für die verschiedenen Verfahren. System- und Referenz-Marginalien wurden auf Stammformen zurückgeführt.

untersucht werden. Zudem existiert im Fall der Marginalien nur eine Referenz-Marginalie pro Textabschnitt. Der Recall betrachtet hier also, ob die einzige Referenz-Marginalie gefunden wurde. Ist die einzige Referenz-Marginalie in den M von einem Verfahren erzeugten System-Marginalien enthalten, sinkt die Precision umso weiter, je größer das für die Berechnung der Precision betrachtete M ist.

Anstatt das F_1-Maß dokumentenweise (für jeden Vorhersage einzeln) zu ermitteln und danach den Mittelwert über alle Dokumente zu bestimmen (auch Makro-F_1 genannt [53]), berichten manche Publikationen den Mikro-F_1. Der Mikro-F_1 wird berechnet, indem zuerst die Gesamtanzahl der True Positives, False Positives und False Negatives über alle Vorhersagen hinweg summiert werden und daraus anschließend das F_1-Maß bestimmt wird [53]. Thomas et al. empfehlen in ihrer 2024 veröffentlichten Studie über die Reproduzierbarkeit der Evaluierungsergebnisse von Keyphrase-Generation-Verfahren [53] den Makro-F_1 anzugeben, da dies für Keyphrase-Generation-Verfahren üblich ist. Den Makro-F_1 gegenüber dem Mikro-F_1 zu verwenden, beruht somit auf Konvention und damit einhergehender besserer Vergleichbarkeit mit anderen Verfahren, nicht aufgrund einer womöglichen Überlegenheit des Makro-F_1 [53]. Alternativ können beide Maße angegeben werden [53]. Für die Evaluierung der Marginalien werden beide Maße berechnet, für bessere Übersichtlichkeit aber nur Makro-F_1 in den Tabellen aufgeführt. Die F_1-Maße und alle im Folgenden betrachteten Maße sind in `quantitative_evaluation_results` zu finden.

Tabelle 4 zeigt die Makro-F_1-Werte an Rangposition 1 für verschiedene Baseline-Verfahren, sowie die vorgestellten Lösungsansätze. Alle Verfahren außer mT5-amazon und GPT-4o generieren mehrere Marginalien, sodass für eine bessere Vergleichbarkeit die Bestimmung der F_1-Werte an Rangposition 1 für alle Verfahren sinnvoll ist. Zudem gibt dies einen besseren Hinweis auf die tatsächliche Ergebnisqualität in der Praxis, da für jeden Textabschnitt nur eine Marginalie gesucht wird.

	F_1@1 Exact Matching			F_1@1 Approximate Matching		
	present	absent	gesamt	present	absent	gesamt
TF-IDF	0.0431	0	0.0431	**0.2648**	0	0.2648
YAKE!	0.0085	0	0.0085	0.0812	0	0.0812
SingleRank	0	0	0	0.0685	0	0.0685
TextRank	0.0008	0	0.0008	0.0474	0	0.0474
PositionRank	0.0161	0	0.0161	0.1168	0	0.1168
TopicRank	0.0228	0	0.0228	0.1954	0	0.1954
MPRank	0.0245	0	0.0245	0.1954	0	0.1954
SIFRank	0.0161	0	0.0161	0.0685	0.0042*	0.0728
SIFRank+	0.0668	0	0.0668	0.1946	0.0042*	0.1988
SetTrans	0.0196	0	0.0196	0.1265	0.013	0.1395
mT5-amazon	**0.1311**	0.0008	**0.132**	0.264	0.011	0.275
GPT-4o	0.1026	**0.0274**	0.1299	0.147	**0.1521**	**0.2991**

Tabelle 4: Makro-F_1-Werte an Rangposition 1 für die vorgestellten Verfahren sowie Baseline-Verfahren. Die mit Stern (*) markierten Werte sind fälschlicherweise ungleich 0. Ein F_1-Wert von 1 ist der bestmögliche Wert, 0 der schlechteste.

Die F_1-Werte sind unterteilt in F_1-Werte für die von den Verfahren erzeugten Present-Marginalien, Absent-Marginalien und den Gesamtwert für beide Marginalien-Typen. Der absent F_1-Wert von GPT-4o bedeutet also, dass von GPT-4o erzeugte Marginalien, die nicht im zugehörigen Textabschnitt vorkommen, einen F_1-Wert von 0.0274 aufweisen und daher seltener mit der Referenz-Marginalie exakt übereinstimmen als die Present-Marginalien von GPT-4o.

Alle Verfahren außer GPT-4o, mT5-amazon und SetTrans sind nicht in der Lage, Absent-Marginalien zu erzeugen. Dennoch treten beim Approximate Matching für SIFRank und SIFRank+ in Tabelle 4 Werte auf, die zwar sehr nahe an der Null, aber ungleich Null sind. Die Ursache dafür liegt vermutlich an unterschiedlicher Tokenisierung oder Stammformbildung zwischen Marginalien und Textabschnitten, sodass Present-Marginalien fälschlicherweise als Absent-Marginalien eingestuft werden. Dieser Fehler konnte reduziert, aber nicht elimi-

niert werden, weshalb die betroffenen Werte in den Tabellen mit einem Stern (*) markiert sind.

Anhand von Tabelle 4 ist ersichtlich, dass die LLM-basierten Verfahren mT5-amazon und GPT-4o die besten Werte erzielen. Besonders für das Exact Matching ist der Abstand zu den anderen Verfahren groß: Während GPT-4o mit 0.1299 den zweitbesten Gesamtwert erreicht, ist SIFRank+ mit 0.0668 als drittbester Gesamtwert um fast 50% niedriger. Beim Approximate Matching hingegen fällt dieser Abstand nicht derart groß aus: mT5-amazon hat den zweitbesten Wert mit 0.275 und TF-IDF bemerkenswerterweise den drittbesten mit 0.2648. Die F_1-Werte sämtlicher Verfahren profitieren immens vom weniger restriktiven Approximate Matching und sind um ein Vielfaches höher.

Beim Exact Matching lässt sich zudem beobachten, dass mT5-amazon den mit 0,1311 besten F_1@1-Wert für Present-Marginalien aufweist, jedoch einen mit 0,0008 sehr niedrigen Wert für Absent-Marginalien. Bei GPT-4o ist zwar auch ein deutlicher Unterschied zwischen beiden Werten erkennbar, dieser fällt aber mit 0,1026 zu 0,0274 um ein Vielfaches geringer aus. Die Ursache hierfür ist unklar, könnte aber durch besseres Textverständnis von GPT-4o bedingt sein. SetTrans kann als Keyphrase-Generation-Verfahren zwar auch Absent-Marginalien generieren, erzielt aber beim Exact Matching einen F_1@1-Wert von 0 für Absent-Marginalien. Lediglich beim Approximate Matching erreicht SetTrans den Wert von 0,011 und liegt damit knapp vor mT5-amazon. GPT-4o liegt um mehr als das Zehnfache höher bei 0,1521.

Tabelle 5 zeigt zum Vergleich die F_1-Werte für alle M jeweils von den Verfahren erzeugten Marginalien. GPT-4o und mT5-amazon sind hierbei nicht erneut aufgeführt, da sie stets nur eine Marginalie erzeugen, während die restlichen Verfahren bis zu 10 Marginalien erzeugen können. Die Werte beider Verfahren sind daher identisch zu denen in Tabelle 4.

Durch die Betrachtung von jeweils zehn System-Marginalien erhalten die Verfahren mehr Chancen, die Referenz-Marginalie zu treffen. Hier fällt auf, dass sich für Exact Matching die Werte gegenüber F_1@1 bis auf TF-IDF und SIFRank+ etwas verbessern. Die Verschlechterung von TF-IDF und SIFRank+ kann bedeuten, dass, wenn jene eine zur Referenz-Marginalie ähnliche Marginalie erzeugen, diese eher an Rangposition 1 vorkommt als an Position 10. Umgekehrt kann die Verbesserung der übrigen Verfahren darauf hinweisen, dass diese Verfahren zur Referenz-Marginalie ähnliche Marginalien erzeugen, aber erst später in der Liste und somit weniger oft auf Platz 1.

Mean Reciprocal Rank Um dies näher zu untersuchen, kann ein weiteres für Keyphrase-Prediction-Verfahren eingesetztes Maß herangezogen werden: der Mean Reciprocal Rank (MRR) [54]. Angepasst auf die Evaluierung der Marginalien ist dieser nach [54] definiert als

$$MRR = \frac{1}{|T|} \sum_{t \in T} \frac{1}{Rang_t} \tag{7}$$

	F_1@M Exact Matching			F_1@M Approximate Matching		
	present	absent	gesamt	present	absent	gesamt
TF-IDF	0.0313	0	0.0313	0.1894	0	0.1894
YAKE!	0.0216	0	0.0216	0.1335	0.0008*	0.1336
SingleRank	0.0079	0	0.0079	0.0981	0	0.0981
TextRank	0.0099	0	0.0099	0.1159	0	0.1159
PositionRank	0.0329	0	0.0329	0.1557	0	0.1557
TopicRank	0.0344	0	0.0344	0.2311	0	0.2311
MPRank	0.034	0	0.034	**0.2329**	0	**0.2329**
SIFRank	0.0284	0	0.0261	0.1448	0.026*	0.1393
SIFRank+	**0.0422**	0	**0.0389**	0.1994	0.0213*	0.1898
SetTrans	0.0209	0	0.0198	0.1352	**0.0156**	0.1391

Tabelle 5: Makro-F_1-Werte an Rangposition M für die vorgestellten Verfahren sowie Baseline-Verfahren. M ist die Anzahl der durch das jeweilige Verfahren erzeugten Marginalien. Die mit Stern (*) markierten Werte sind fälschlicherweise ungleich 0. Ein F_1-Wert von 1 ist der bestmögliche Wert, 0 der schlechteste.

mit $Rang_t$ als Rangposition des ersten (und einzigen) Treffers in der Liste der erzeugten Marginalien, T als Menge aller Textabschnitte und t als bestimmten Textabschnitt aus T. Zwar ermisst sich der Erfolg der Lösungsansätze im praktischen Anwendungsfall ausschließlich daran, wie gut die Marginalie auf Rang 1 ist, dennoch kann der MRR für den Vergleich der Lösungsansätze sinnvoll sein. Aus Gründen der Vergleichbarkeit sind GPT-4o und mT5-amazon nicht aufgeführt, da diese jeweils nur eine Marginalie erzeugen. Die Werte beider Verfahren sind hier erneut identisch zu denen in Tabelle 4.

Tabelle 6 zeigt den MRR für Exact sowie Approximate Matching. Tatsächlich erreichen SIFRank+ und TF-IDF hier die höchsten Werte, können also Treffer durchschnittlich weiter vorne in der Rangfolge der Ergebnisse platzieren. Der Abstand des MRR zwischen SIFRank+ und TF-IDF und den übrigen Verfahren ist für Approximate Matching besonders groß. Der $F_1@M$ von TextRank und SIFRank weist verglichen zum $F_1@1$ den größten Anstieg auf. In Übereinstimmung damit erzielt TextRank den niedrigsten und SIFRank den fünftniedrigsten MRR für Approximate Matching aus zehn Verfahren. Dagegen verzeichnet TF-IDF als Verfahren mit dem höchsten MRR hier den größten und einzigen Verlust zwischen $F_1@1$ und $F_1@M$. SIFRank+ mit dem zweithöchsten MRR weist zwar keinen Verlust auf, jedoch nur einen sehr geringen Anstieg zwischen $F_1@1$ und $F_1@M$.

Neben der Beschränkung auf lexikalische Ähnlichkeit durch Maße wie F_1 oder MRR betrifft eine weitere Einschränkung die Exact- und Approximate-Match-Evaluierung. Für diese wird üblicherweise Stammformbildung durchgeführt, sodass beispielsweise auch System-Marginalien, die anders flektiert sind als die

	MRR Exact Matching	MRR Approximate Matching
TF-IDF	0.075	**0,3766**
YAKE!	0.0324	0.1653
SingleRank	0.006	0.1324
TextRank	0.0095	0.1243
PositionRank	0.0517	0.2298
TopicRank	0.0376	0.247
MPRank	0.0386	0.2484
SIFRank	0.0405	0.1723
SIFRank+	**0.105**	0.3333
SetTrans	0.0202	0.1449

Tabelle 6: Mean Reciprocal Rank für Exact und Approximate Matching. Ein MRR von 1 ist der bestmögliche Wert. Je niedriger der MRR, desto später treten Treffer durchschnittlich in einer Rangfolge auf.

Referenz-Marginalie, als übereinstimmend gewertet werden. Während dies für den inhaltlichen Vergleich von System- und Referenz-Marginalie sinnvoll ist, wird ein grammatikalischer Vergleich durch die Stammformbildung verhindert. Die erzeugten Marginalien sind nicht Teil eines Satzes, sondern sollen für sich alleine stehen können. Dadurch kann insbesondere bei extraktiven Verfahren eine Marginalie zwar inhaltlich übereinstimmend mit der Referenz sein, aber unpassend flektiert oder dekliniert sein: Die im Textabschnitt enthaltene System-Marginalie von TF-IDF „Beispielen“ müsste als alleinstehende Marginalie im Nominativ Plural stehen („Beispiele“). Für den Rahmen dieser Arbeit wird der Fokus auf den Inhalt statt auf die grammatikalische Korrektheit der Marginalien gesetzt. Zukünftig bedarf es weiterer Untersuchungen, die auch diesen Aspekt miteinbeziehen.

LLM-basierte Maße Im Gegensatz zu den bisher betrachteten Maßen können LLM-basierte Maße semantische Übereinstimmung messen. Ein Beispiel dafür ist MoverScore, das die semantische Ähnlichkeit zwischen einem generierten Text und einem Referenz-Text bestimmt [55]. Dafür verwendet MoverScore kontextualisierte Einbettungen wie BERT, um die Bedeutung von Wörtern im Kontext zu erfassen [55]. MoverScore nutzt dazu die Word Mover's Distance, welche die minimale „Reisedistanz“ ermittelt, die nötig ist, um vom Embedding der Wörter oder n-Gramme des Referenz-Texts zu denen des generierten Textes zu gelangen [55].

Ein möglicher Nachteil ist hierbei, dass sowohl die erzeugten als auch Referenz-Marginalien nur aus wenigen Wörtern, teils sogar nur aus einem einzelnen Wort

bestehen. Die kontextualisierten Einbettungen können in diesen Fällen also kaum auf jeglichen Kontext zurückgreifen. Möglicherweise ist MoverScore für längere Texte, wie sie bei Summarization vorkommen, besser geeignet. Darüber hinaus könnte der Vergleich der semantischen Ähnlichkeit zwischen den erzeugten Marginalien und den zugehörigen Textabschnitten über die verschiedenen Lösungsansätze sinnvoll sein.

Erste Tests unter Verwendung eines deutschen BERT-Modells[17] zeigen aber auch Einschränkungen von MoverScore[18]. Für die fiktiven Beispielsätze

S1: Der Hund biss den Mann
S2: Der Hund verletzte den Mann
S3: Der Mann biss den Hund

wird für die semantisch sehr ähnlichen Sätze S1 und S2 ein MoverScore von gerundet 0,76 zurückgegeben. S1 und S3, in denen die Tatrelation invertiert ist, ergeben einen MoverScore von gerundet 0,84. Somit ist $MoverScore(S1, S2) < MoverScore(S1, S3)$. Da S2 zu S1 semantisch ähnlicher ist, als S3 zu S1, müsste stattdessen aber $MoverScore(S1, S2) > MoverScore(S1, S3)$ gelten.

Dass die Lösungsverfahren Marginalien ähnlich den fiktiven Beispielsätzen erzeugen, ist allerdings unwahrscheinlich. MoverScore wird deshalb dennoch zur Evaluierung eingesetzt, bei der Interpretation der Ergebnisse müssen die Einschränkungen aber beachtet werden.

Tabelle 7 zeigt die ermittelten MoverScores an Rangposition 1. Die linke Hälfte der Tabelle stellt den ursprünglichen MoverScore dar, der die System-Marginalien an Rangposition 1 und die Referenz-Marginalien vergleicht. In der rechten Hälfte der Tabelle werden dagegen System-Marginalien an Rangposition 1 und dazugehörige Textabschnitte miteinander verglichen.

Hinsichtlich des ursprünglichen MoverScores gehören, wie schon beim F_1-Maß, GPT-4o und mT5-amazon mit Abstand zu den besten Verfahren. Die übrigen Werte der übrigen Verfahren liegen näher zusammen. Anders als für F_1 und MRR erzielt SetTrans hier den dritthöchsten Wert und selbst TF-IDF und YAKE! übertreffen SIFRank+. Im Gegensatz zu TF-IDF erreicht YAKE! wesentlich niedrigere F_1@1-Werte als SIFRank+, erreicht aber einen moderat höheren MoverScore.

Betrachtet man den MoverScore zwischen System-Marginalien an Rangposition 1 und den dazugehörigen Textabschnitten, ergibt sich eine gänzlich andere Rangfolge der Verfahren, zudem liegen die Werte der Verfahren näher beieinander. Besonders unerwartet ist, dass SingleRank den höchsten MoverScore erzielt, obwohl es hinsichtlich F_1 und MRR in den meisten Fällen die schlechtesten Ergebnisse aller Verfahren aufweist. Werden die von SingleRank erzeugten Marginalien untersucht, fällt auf, dass diese überwiegend signifikant länger sind als die der anderen Verfahren. Tatsächlich bestehen die Marginalien von

[17] https://huggingface.co/dbmdz/bert-base-german-cased
[18] Die verwendete Implementierung ist unter https://github.com/AIPHES/emnlp19-moverscore zu finden.

SingleRank aus durchschnittlich 5,03 Wörtern, während die restlichen Verfahren durchschnittliche Längen von 1,23 bis 2,74 aufweisen. Als extraktives Verfahren übernimmt SingleRank Marginalien wortwörtlich aus dem Textabschnitt, sodass längere Marginalien längere Auszüge aus dem Textabschnitt darstellen und diesem folglich semantisch umso ähnlicher sind.

Im Zuge dieser Untersuchung wird ersichtlich, dass wenn die Verfahren nach der absteigenden durchschnittlichen Länge ihrer Marginalien sortiert werden, die Reihenfolge fast identisch mit der Reihenfolge der MoverScores zwischen System-Marginalien und den dazugehörigen Textabschnitten in Abb. 7 ist. Lediglich TF-IDF und SetTrans sind in der Reihenfolge vertauscht, ansonsten stimmt die Reihenfolge überein. Demzufolge scheint der MoverScore zwischen System-Marginalien und den dazugehörigen Textabschnitten kein adäquates Maß für die Evaluierung der Verfahren zu sein, da dieser letztlich nur die Länge der erzeugten Marginalien widerspiegelt.

	MoverScore	
	Marginalie ↔ Marginalie	Marginalie ↔ Text
TF-IDF	0.5626	0.4803
YAKE!	0.5732	<u>0.4922</u>
SingleRank	0.5193	**0.4944**
TextRank	0.535	0.487
PositionRank	0.5463	0.4865
TopicRank	0.5569	0.4788
MPRank	0.5573	0.4789
SIFRank	0.5492	0.4854
SIFRank+	0.558	0.4835
SetTrans	0.5874	0.4795
mT5-amazon	<u>0.6376</u>	0.4842
GPT-4o	**0.6543**	0.4903

Tabelle 7: MoverScore-Werte für die verschiedenen Verfahren. Die linke Hälfte ermittelt den MoverScore für System- und Referenz-Marginalie, die rechte Hälfte für System-Marginalie und den zugehörigen Textabschnitt.

6.2 Qualitative Evaluierung

Aufgrund der in Kapitel 6.1 betrachteten Einschränkungen der quantitativen Messgrößen ist eine qualitative Evaluierung der erzeugten Marginalien angezeigt. Dafür werden im Folgenden die System-Marginalien der verschiedenen

Lösungsansätze beispielhaft untersucht und mit den Referenz-Marginalien verglichen. Der Übersichtlichkeit halber werden nicht die erzeugten Marginalien aller Verfahren aufgeführt, sondern nur die vorgestellten Lösungsansätze SIFRank+, SetTrans, mT5-amazon und GPT-4o, sowie zwei statistische Verfahren YAKE! und PositionRank als Baseline. Außer mT5-amazon und GPT-4o erzeugen alle Verfahren mehrere Marginalien. Für diese Verfahren wird die Marginalie an Rangposition 1 gewählt. Anhand einer Umfrage werden anschließend die Präferenzen zwischen den Verfahren analysiert.

Abb. 9 zeigt einen Textabschnitt, dessen Referenz-Marginalie sowohl im Textabschnitt selbst enthalten ist, als auch zu Beginn des Textabschnitts steht. Dies ermöglicht es zum einen, den extraktiven Verfahren mit der Referenz-Marginalie übereinstimmende System-Marginalien zu generieren und ist zum anderen vorteilhaft für Verfahren wie SIFRank+, die die Position der Kandidaten miteinbeziehen. Zusätzlich tritt der Begriff JSON viermal innerhalb des Textabschnitts auf, was für die statistischen Verfahren günstig sein kann.

Tatsächlich beziehen sich alle betrachteten System-Marginalien auf JSON und sind somit nahe an der Referenz-Marginalie. mT5-amazon, GPT-4o und auch SIFRank+ als unüberwachtes Verfahren stimmen inhaltlich exakt mit der Referenz überein, wobei SIFRank+ die Marginalie in Kleinschreibung erzeugt. Auch YAKE! als statistisches Verfahren generiert eine gute Marginalie, die die ausgeschriebene Form des Akronyms JSON darstellt. Dabei ist es diskutabel, ob das Akronym aufgrund der Kürze als Marginalie etwas besser geeignet ist. Je nach Abwägung können beide Varianten aber als gleichwertig betrachtet werden.

PositionRank wählt dagegen eine gegen Ende des Textabschnitts auftretende Wortfolge als Marginalie aus, die zwar das Wort JSON enthält, aber irreführend sein kann. „json datenformaten wie xml“ kann so interpretiert werden, dass JSON ein zu XML ähnliches Datenformat ist, was im Textabschnitt so aber nicht beschrieben wird. SetTrans generiert keine Marginalie.

Zusätzlich wird am Beispiel von PositionRank deutlich, dass die extraktiven Verfahren allgemein die Keyphrases bzw. Marginalien so aus dem Text extrahieren, wie sie dort vorkommen. Dadurch können die extrahierten Wortformen als eigenständige Marginalie unpassend sein. Diese Problematik tritt bei der deutschen Sprache aufgrund der Vielzahl an Flexions- und Deklinationsformen vermutlich häufiger auf als bei englischen Texten.

Abb. 10 zeigt ein Beispiel für eine teilweise Absent-Marginalie, da die Referenz-Marginalie „HSQL-Datenbank“ nicht als solche im Textabschnitt auftritt, Teile von ihr aber schon. Darüber hinaus handelt es sich um eine Marginalie mit Zusatzinformation, da auf Basis des Textes zwar vermutet werden kann, aber nicht explizit beschrieben wird, dass das „DB“ in „HSQLDB“ für „Datenbank“ steht.

Bis auf PositionRank enthalten alle System-Marginalien zumindest Teile der Referenz-Marginalie. SIFRank+ und SetTrans erzeugen die System-Marginalie „Datenbank“. Dies kann als zutreffende Marginalie betrachtet werden. Eine Spezifizierung, um welche Datenbank es sich genau handelt, könnte aber in diesem Fall zu bevorzugen sein. mT5-amazon und GPT-4o generieren zusätzlich den konkreten Namen der Datenbank. GPT-4o ordnet HSQLDB zudem in eine be-

Netzsicherheit, Kap. 4: Asymmetrische kryptografische Verfahren [29]

JSON (JavaScript Object Notation) ist eine Darstellung von Daten, die vor allem für JavaScript optimiert ist. Wie JavaScript auch, sind die Daten dynamisch typisiert. Mittlerweile gibt es aber eigentlich für alle Programmiersprachen passende JSON-Bibliotheken. Es gibt außerdem Typsysteme wie JSON Schema [16], die für JSON eine entsprechende Validierung ergänzen. Damit steht JSON Datenformaten wie XML in nichts mehr nach.

Referenz: JSON

SIFRank+:	json
(**SetTrans:**	*fehlt*)
mT5-amazon:	JSON
GPT-4o:	JSON
YAKE!:	JavaScript Object Notation
PositionRank:	json datenformaten wie xml

Abb. 9: Beispiel für einen Textabschnitt, für den die Mehrheit der Verfahren die Referenz-Marginalie trifft oder eine ähnlich gute Marginalie erzeugt.

stimmte Art von Datenbank ein und gibt somit mehr inhaltliche Information. YAKE! erzeugt ebenfalls eine zutreffende Marginalie, die sich aufgrund der Verwendung von Verben von den anderen System-Marginalien unterscheidet.

Ähnlich wie zuvor wird auch hier eine subjektive Komponente beim Vergleich verschiedener Marginalien-Varianten deutlich: Zwar lässt sich teils entscheiden, welche Marginalien besser zu dem jeweiligen Textabschnitt passen als andere, allerdings kann diese Einschätzung auch subjektiv sein.

Dies wird auch in Abb. 11 deutlich, dessen System-Marginalien deutlich von der Referenz-Marginalie abweichen. Während die System-Marginalie als Absent-Marginalie allgemein von der „Vorgehensweise bei der Verlegung“ spricht, setzen die meisten System-Marginalien den Fokus auf einen konkreten Aspekt, der bei der Verlegung beachtet werden muss: mechanische Belastung des Kabels.

Es kann auch argumentiert werden, dass aufgrund der Aufzählung von Vorsichtsmaßnahmen bei der Verlegung und der Betonung deren Wichtigkeit mittels Adverbien („[...] mechanische Belastung **unbedingt** [...] vermeiden“, „Kabel **keinesfalls** geknickt verlegen“, „[...] Biegegrade [...], die **keinesfalls** unterschritten werden sollten“) die Marginalie von GPT-4o am besten geeignet ist, da diese im Gegensatz zur Referenz-Marginalie den wichtigsten Kernpunkt bzw. die Lektion des Textabschnitts zusammenfasst. Insgesamt wird auch an diesem Beispiel ersichtlich, dass Marginalien-Varianten unterschiedliche Aspekte des gleichen Textabschnitts betrachten können und fehlende Übereinstimmung mit der Referenz-Marginalie nicht gleichbedeutend mit schlechten System-Marginalien sein muss.

Netzsicherheit, Kap. 11: Sicherheitsprotokolle der Datensicherungsschicht [29]

Die Datenbank HSQLDB verwaltet und speichert die Daten. Es ist eine In-Memory-Datenbank, die in Java geschrieben ist. Sie legt die Daten im RAM ab, sodass beim erneuten Start der Anwendung die Daten verloren sind. Für einen produktiven Einsatz ist diese Datenbank nicht sinnvoll, auch wenn sie die Daten auf eine Festplatte schreiben kann. Dafür muss so kein zusätzlicher Datenbank-Server installiert werden, was die Beispielanwendung sehr einfach hält. Die Datenbank läuft in der jeweiligen Java-Anwendung.

Referenz:
HSQL-Datenbank

SIFRank+:	datenbank
SetTrans:	Datenbank
mT5-amazon:	Datenbank HSQLDB
GPT-4o:	HSQLDB als In-Memory-Datenbank
YAKE!:	HSQLDB verwaltet und speichert
PositionRank:	daten im ram

Abb. 10: Beispiel für einen Textabschnitt, für den nahezu alle Verfahren gute Marginalien erzeugen, aber keine die Referenz-Marginalie genau trifft. Dafür wäre Hintergrundwissen oder Erschließen aus dem Kontext notwendig, da im Textabschnitt nur „HSQLDB“, nicht aber „HSQL-Datenbank“ erwähnt wird.

Außerdem zeigt Abb. 11, dass extraktive Verfahren nicht notwendigerweise nur bei Present-Marginalien erfolgreich sein können, sondern auch bei Absent-Marginalien gute Ergebnisse erzeugen können, die aber möglicherweise einen anderen Fokus setzen. Abgesehen von der grammatikalischen Korrektheit ist die Marginalie von SIFRank+ zutreffend, da der Textabschnitt in der Tat von der Belastung des Kabels handelt. Auch die System-Marginalie von YAKE! bezieht sich auf mechanische Belastung. PositionRank generiert auch hier das wohl am wenigsten repräsentative Ergebnis, da der Textabschnitt nicht nur oder nicht überwiegend vom Abrollen des Kabels handelt, sondern das Abrollen nur einen kleinen Teil des Inhalts darstellt. SetTrans erzeugt erneut keine Marginalie.

Auf Basis der betrachteten Beispiele und einer weiteren Stichprobe zur Gewinnung eines ersten Eindrucks erzeugen GPT-4o und mT5-amazon die besten Marginalien. Doch auch SIFRank+ und sogar YAKE! können mitunter gute Marginalien erzeugen, insbesondere, wenn es sich um Present-Marginalien handelt, die zu Beginn des Textabschnitts vorkommen. PositionRank schneidet hingegen der Stichprobe zufolge schlechter ab und SetTrans erzeugt oft keine Marginalien.

Während diese stichprobenartige Betrachtung einzelner Beispiele interessante Einblicke gewähren kann, können auf diese Weise nicht alle 1182 Elemente des Test-Splits untersucht werden. Darüber hinaus ist für den Vergleich der (teils subjektiven) Tauglichkeit der verschiedenen System-Marginalien das Einholen verschiedener Meinungen sinnvoll. Daher wurde eine Online-Umfrage durchgeführt.

Für die Umfrage wurden 100 Marginalien samt Textabschnitt zufällig aus dem Test-Split ausgewählt und in einen separaten Datensatz kopiert. Für jede dieser Marginalien wurden die System-Marginalien von SIFRank+, mT5-amazon, GPT-4o, YAKE! und PositionRank erzeugt und hinzugefügt. SetTrans ist nicht Teil der Umfrage, da zum Zeitpunkt des Beginns der Umfrage noch die Fehlersuche bei WR-SetTrans andauerte. Die Alternative SetTrans erzeugt zudem in vielen Fällen kein Ergebnis.

Die Umfrage erfolgt über ein am Lehrstuhl für Medieninformatik entwickeltes Umfrage-Tool, das während des Umfragezeitraums online zugänglich war. Teilnehmer erhalten initial eine kurze Einführung, wovon die Umfrage handelt und was getan werden soll. Anschließend werden pro Umfragedurchgang zehn Fragen gestellt. Jede Frage besteht aus einem Textabschnitt und der Auswahl zwischen zwei zufällig gewählten System-Marginalien (Abb. 12). Der Teilnehmer soll jeweils auswählen, welche der beiden Marginalien besser zum Textabschnitt passt. Dieser A/B-Vergleich wurde gewählt, da es tendenziell einfacher ist, eine Alternative aus zwei Möglichkeiten zu wählen, als etwa eine Rangfolge aus allen fünf System-Marginalien zu bestimmen oder eine einzelne Marginalie auf einer Likert-Skala zu bewerten. Dies gilt insbesondere vor dem Hintergrund, dass die Marginalien unterschiedliche Aspekte betrachten können und es nicht unbedingt eine objektiv bessere Alternative gibt. Sind zwei System-Marginalien identisch oder werden als gleich gut eingeschätzt, kann auch die Auswahl „Beide gleich

Netzsicherheit, Kap. 11: Sicherheitsprotokolle der Datensicherungsschicht [29]

Bei der Verlegung muss man unbedingt eine zu starke mechanische Belastung des Kabels vermeiden. Das fängt beim Abrollen des Kabels bereits an, das auf jeden Fall gleichmäßig in einer Richtung erfolgen sollte. Das Einziehen in einen Kabelkanal darf das Kabel nicht zu stark durch Zug beanspruchen. Wird zum Beispiel ein TP-Kabel bei der Installation durch Kabelbinder oder Halteschellen zu stark gequetscht, können zum einen die Adern beschädigt und zum anderen die Lage der Adernpaare zueinander verändert werden, wodurch sich die Charakteristik des TP-Kabels nachhaltig verschlechtern kann. Außerdem sollte der Installateur bei den Kabeln gewisse Biegeradien einhalten und die Kabel keinesfalls geknickt verlegen. Dies gilt unabhängig davon, ob es sich um ein LWL-oder TP-Kabel handelt. Die Kabelhersteller geben in der Regel in ihren Datenblätter entsprechende minimal zulässige Biegeradien an, die keinesfalls unterschritten werden sollten.

Referenz: Vorgehensweise bei der Verlegung	**SIFRank+:**	belastung der kabels
	(**SetTrans:**	*fehlt*)
	mT5-amazon:	starke mechanische Belastung
	GPT-4o:	Mechanische Beanspruchung vermeiden
	YAKE!:	starke mechanische Belastung
	PositionRank:	abrollen des kabels

Abb. 11: Beispiel für einen Textabschnitt, dessen System-Marginalien überwiegend zwar grundsätzlich zutreffend sind, aber einen anderen Aspekt des Textes als die Referenz-Marginalie betrachten.

gut“ getroffen werden. Nach Beantworten aller zehn Fragen kann der Teilnehmer einen neuen Umfragedurchlauf starten.

Abb. 12: Screenshot des Umfrage-Tools. Die Seitenleiste links zeigt den Fortschritt innerhalb der aktuellen Umfrage.

Die Beschränkung auf 100 Marginalien mit je 5 System-Marginalien wurde gewählt, damit die Wahrscheinlichkeit höher ist, dass auch bei geringeren Teilnehmerzahlen mehrere Teilnehmer die gleichen Fragen mit unterschiedlichen System-Marginalien zur Auswahl beantworten werden. Auf diese Weise soll untersucht werden, welche Verfahren System-Marginalien erzeugen, die von den Umfrageteilnehmern gegenüber den Ergebnissen anderer Verfahren bevorzugt werden. Zur besseren Lesbarkeit wird im Folgenden von der Bewertung von Verfahren gesprochen, auch wenn die Teilnehmer der Umfrage nicht direkt die Verfahren selbst bewerten, sondern die durch jene erzeugten Marginalien. Außerdem wird eine Frage bestehend aus einem Textabschnitt und zwei Marginalien-Alternativen fortan kurz als Frage bezeichnet. Wird ein Verfahren A in einer Frage gegenüber einem Verfahren B als besser eingestuft, wird dies fortan als „Verfahren A wird ausgewählt“ bezeichnet.

Im Rahmen der Umfrage wurden 412 Fragen von 38 Teilnehmenden beantwortet. Die Antworten sind in einer MongoDB-Collection persistiert. Jede Antwort umfasst eine Referenz auf den gezeigten Textausschnitt, die Namen der zwei zur Auswahl gestellten Verfahren sowie das vom Teilnehmer per „Diese ist besser“-Button ausgewählte Verfahren.

Jede Frage wurde durchschnittlich 4,12 mal beantwortet, bei einem Median von 4, einem Minimum von 1 und einem Maximum von 8. Die für eine Frage zufällig zur Anzeige ausgewählten Zweier-Kombinationen der Verfahren wurden

durchschnittlich 41,2 mal angezeigt, bei einem Median von 42,5, einem Minimum von 34 und einem Maximum von 48. Da zehn unterschiedliche, ungeordnete Kombinationen möglich sind, läge eine ideale Gleichverteilung bei 41,3 Fragen pro Verfahren-Kombination. Im Folgenden soll anhand dieser Umfrageergebnisse untersucht werden, welche Verfahren von den Teilnehmenden im Vergleich zu anderen Verfahren bevorzugt werden.

Auswahlhäufigkeit der Verfahren In Abb. 13 wird visualisiert, wie oft die jeweiligen Verfahren prozentual von den Teilnehmern ausgewählt wurden. Wie schon in der quantitativen Evaluierung zeichnen sich hier GPT-4o und mT5-amazon als führende Verfahren ab. SIFRank+ und PositionRank wurden ähnlich oft ausgewählt, während YAKE! in nur 7,28% der Fälle ausgewählt wurde. Dass SIFRank+ öfter ausgewählt wurde als YAKE! und damit gegenüber einem anderen Verfahren häufiger als besser eingestuft wurde, spiegelt sich in den F_1@1-Werten bei Exact Matching (Abb. 4) wider. Dort erreicht SIFRank+ einen Wert von 0,0668 gegenüber 0,0085 von YAKE!. PositionRank weist zwar ebenfalls einen höheren F_1@1-Wert als YAKE! auf, dieser liegt mit 0,0161 jedoch deutlich näher an YAKE!. Dieselbe Rangfolge von SIFRank+, PositionRank und YAKE! ist auch für Approximate Matching bei F_1@1 zu beobachten.

Dem gegenüber liegt hinsichtlich der MoverScore-Werte (Abb. 7) YAKE! vor SIFRank+ und PositionRank, was nicht mit den Ergebnissen der Umfrage in Abb. 13 übereinstimmt. GPT-4o und mT5-amazon hingegen weisen passend zu Abb. 13 die erst- und zweitbesten MoverScore-Werte auf.

In 16,75% der Fälle - und damit an dritter Stelle noch vor den restlichen drei Verfahren - wurden die den Teilnehmern angezeigten Verfahren als gleich gut eingestuft. Die Option „Beide gleich gut“ erlaubt dabei aber keine eindeutigen Schlüsse über den Grund für diese Einstufung. Die Marginalien beider Verfahren könnten inhaltsgleich sein, aber gegebenenfalls anders formuliert, könnten gut zum Textabschnitt passen, aber verschiedene Aspekte des Textabschnitts beleuchten oder könnten beide gänzlich unpassend sein. Zukünftig wären nähere Untersuchungen der Gründe für die Einstufung als gleich gut wünschenswert.

Auswahlhäufigkeit im paarweisen Vergleich Während Abb. 13 die Auswahlhäufigkeiten der Verfahren insgesamt darstellt, lässt sich anhand von Abb. 14 die Auswahlhäufigkeit im paarweisen Vergleich ablesen. Abb. 14 zeigt also, welche der jeweils in einer Frage gezeigten Verfahren im Vergleich zueinander häufiger ausgewählt wurden. Da manche Verfahrens-Kombinationen öfter vorkommen als andere, wird jeder Wert anhand der Auftretenszahl der jeweiligen Kombination (d.h. wie oft diese Kombination den Teilnehmern in einer Frage angezeigt wurde) normiert. Die in einer Verfahrens-Kombination von den Teilnehmern bevorzugten Verfahren sind in den Zeilen der Matrix angetragen, die Spalten zeigen die im Vergleich als schlechter eingestuften Verfahren.

Ausgehend von den Zeilen kann nun beispielsweise abgelesen werden, mit welcher (normierten) Wahrscheinlichkeit GPT-4o gegenüber YAKE! bevorzugt wird, wenn ein Teilnehmer aus beiden die bessere Marginalie wählen muss. In

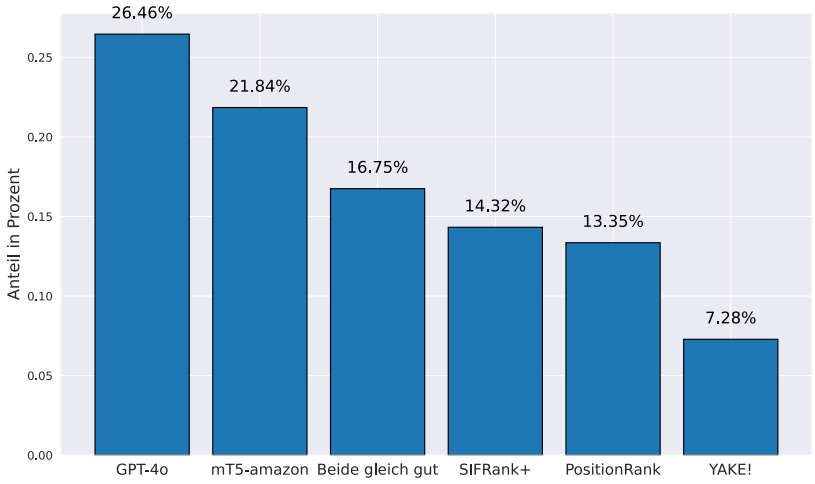

Abb. 13: Relative Auswahlhäufigkeit der verschiedenen Verfahren in der Umfrage.

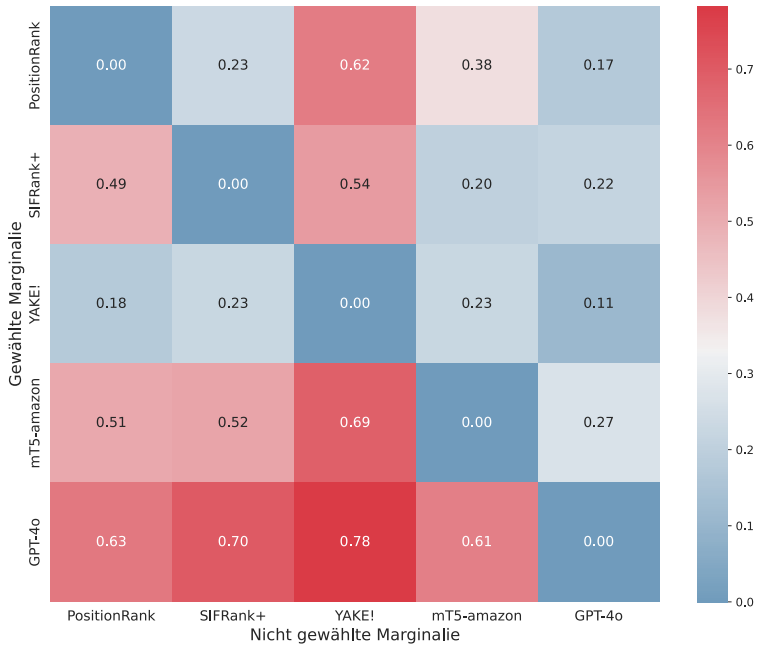

Abb. 14: Heatmap für die normierten Wahrscheinlichkeiten, dass Marginalie A (y-Achse) gegenüber Marginalie B (x-Achse) bevorzugt wird.

diesem Beispiel wird GPT-4o mit einer Wahrscheinlichkeit von 0,78 gegenüber YAKE! bevorzugt. Umgekehrt wird YAKE! gegenüber GPT-4o mit einer Wahrscheinlichkeit von nur 0,11 bevorzugt. Da es für jede Frage die Auswahl zwischen den drei Antwortmöglichkeiten Verfahren A, Verfahren B oder „Beide gleich gut“ gibt, hätten diese Antwortmöglichkeiten bei einer Gleichverteilung eine Wahrscheinlichkeit von je $\frac{1}{3}$. Ein Wert über rund 0,33 bedeutet also, dass dieses Verfahren häufiger bevorzugt wird, als das andere Verfahren sowie die Option „Beide gleich gut“. Da die Auswahl von „Beide gleich gut“ in Abb. 10 nicht dargestellt wird, sondern separat in Abb. 15, addieren sich die wechselseitigen Wahrscheinlichkeiten einer Verfahrens-Kombination in Abb. 14 nicht zu 1.

Auf den ersten Blick ist anhand der Rotfärbung der GPT-4o-Zeile in Abb. 14 zu sehen, dass GPT-4o gegenüber den anderen Verfahren überwiegend bevorzugt wird. Auch mT5-amazon wird öfter als die anderen Verfahren bevorzugt, wird aber gegenüber GPT-4o seltener gewählt. Anhand der Blaufärbung der YAKE!-Zeile ist ersichtlich, dass YAKE! oft der „Verlierer“ ist und im Vergleich mit den anderen Verfahren als schlechter bewertet wird. Die nur schwach eingefärbten Felder in den Zeilen von PositionRank und mT5-amazon visualisieren, dass dort keine starke Tendenz der Bevorzugung oder Ablehnung zu erkennen ist. Hat ein Teilnehmer die Auswahl zwischen SIFRank+ und PositionRank, wird mit einer Wahrscheinlichkeit von 0,49 SIFRank+ gewählt, mit einer Wahrscheinlichkeit von 0,23 PositionRank und in den restlichen Fällen werden beide als gleich gut eingestuft.

Auch hier stellen GPT-4o und mT5-amazon übereinstimmend mit den F_1@1- und MoverScore-Werten der quantitativen Evaluierung die beiden besten Verfahren dar, während YAKE! das Schlusslicht bildet. Die Abstufung zwischen SIFRank+ und PositionRank ist anhand der Farben etwas schwieriger. Werden die Wahrscheinlichkeiten der Zeilen für SIFRank+ und PositionRank jeweils addiert, führt SIFRank+ mit 1,45 nur knapp vor PositionRank mit 1,4. Somit sind beide auf Basis der Umfrageergebnisse annähernd gleich gut. Auch hier ist zu beobachten, dass der quantitative Unterschied der F_1@1-Werte beider deutlich größer ist, als es die qualitativen Ergebnisse zeigen.

Dies unterstreicht auch Abb. 15, die normiert nach der Auftretenszahl der jeweiligen Kombination betrachtet, welche Verfahren besonders oft als gleich gut bewertet wurden. Besonders oft werden demzufolge SIFRank+ und PositionRank als gleich gut eingestuft. Anhand der quantitativen Werte weniger erwartbar wurden zudem SIFRank+ und mT5-amazon öfter als andere Kombinationen als gleich gut bewertet. Dagegen werden GPT-4o und SIFRank+ sowie mT5-amazon und YAKE! am seltensten als gleich gut eingeschätzt.

Rangordnung der Verfahren Eine Möglichkeit, aus den paarweisen Vergleichen eine Rangordnung der in der Umfrage betrachteten Verfahren zu erstellen, ist das Elo-Rating. Ursprünglich für das Bewerten von Schachspielern erfunden, weist das Elo-Rating jedem Spieler ein Rating zu, das bei einem Sieg steigt und bei einer Niederlage sinkt [56]. Wird die Auswahl zwischen zwei Verfahren in der

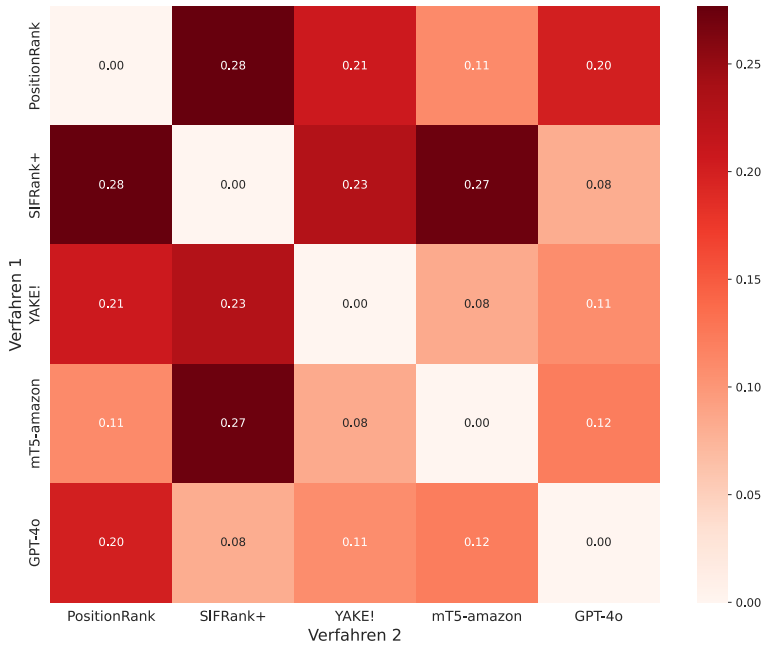

Abb. 15: Heatmap für die normierten Wahrscheinlichkeiten, dass zwei Verfahren mit „Beide gleich gut“ bewertet werden.

Umfrage als Partie betrachtet, steigt das Rating des ausgewählten Verfahrens als „Sieger“, während das Rating des „Verlierers“ sinkt [56].

Je nach Wahl des nach einer Partie zur Anpassung des Elo-Ratings verwendeten Parameters K kann sich die Rangfolge von SIFRank+, mT5-amazon und PositionRank jedoch ändern. Je höher K desto mehr erhöht sich das Elo-Rating des Siegers und desto mehr sinkt das Elo-Rating des Verlierers. Einzelne Partien erhalten somit mehr Gewicht und können sich stärker auf die Rangfolge auswirken.

Hier scheint $K = 5$ eine geeignete Wahl zu sein, da die Entwicklung der Elo-Ratings ab einem bestimmten Punkt stabil bleibt, d.h. es sind keine drastischen Veränderungen des Elo-Ratings zu erwarten. Abb. 16 zeigt diese Entwicklung im Verlauf der 412 Partien (Fragen). Die Graphen von GPT-4o, YAKE! und mT5-amazon haben hinsichtlich der Entwicklung ihres Elo-Ratings eine klare Tendenz, während PositionRank und SIFRank+ dicht beieinander liegen und ihre Rangfolge mehrfach wechseln.

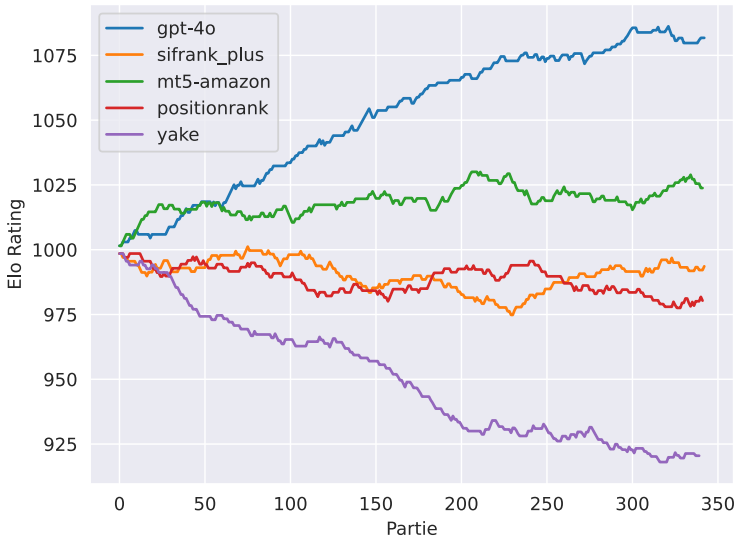

Abb. 16: Entwicklung der Elo-Ratings im Verlauf der Partien (Fragen) für $K = 5$. Partien mit der Auswahl „Beide gleich gut“ werden für das Elo-Rating ignoriert.

Für die Rangfolge der Marginalien-Verfahren wurde $K \in \{1, 3, 5, 10, 15, 20, 25, 32, 50\}$ getestet. Dabei ist GPT-4o stets auf dem ersten und YAKE! stets auf dem letzten Platz, die Reihenfolge der übrigen Verfahren variiert ab $K = 25$.

Da die Umfrage aus nur 100 verschiedenen, zufällig ausgewählten Fragen besteht und mit 412 beantworteten Fragen jeder Textabschnitt durchschnittlich

nur rund viermal bewertet wurde, kann die Umfrage zwar als Eindruck dienen, für noch aussagekräftigere Ergebnisse wäre eine größere Datenbasis mit mehr Teilnehmern wünschenswert. Zum einen ist es durch die Auswahl von 100 aus 11.808 Textabschnitten mit Marginalien möglich, dass für manche Verfahren überproportional viele schlechte Marginalien Teil der Umfrage sind, obwohl das Verfahren überwiegend gute Marginalien erzeugt. Gleiches gilt für eine potenzielle Überschätzung eines Verfahrens aufgrund überproportional vieler guter Marginalien in der Umfrage. Zum anderen

Trotz der Einschränkungen ist die mittels Elo-Rating ermittelte Rangordnung plausibel, da sie sowohl mit dem persönlichen Eindruck auf Basis von Stichproben als auch mit den Ergebnissen der quantitativen Evaluierung übereinstimmt. Dass GPT-4o aus den in der Umfrage betrachteten Verfahren die besten Ergebnisse und YAKE! die schlechtesten Ergebnisse erzielt, geht aus der Umfrage recht klar hervor. Dass mT5-amazon bessere Marginalien als SIFRank+ und PositionRank generiert, spiegelt sich auch in der quantitativen Evaluierung wider und geht aus den Umfragergebnissen tendenziell auch hervor, wenn auch weniger eindeutig als bei GPT-4o. SIFRank+ und PositionRank müssen anhand der Umfrageergebnisse als etwa gleich gut bewertet werden, da diese sowohl hinsichtlich der Auswahlhäufigkeit im paarweisen Vergleich als auch hinsichtlich Elo-Rating nur geringe Unterschiede aufweisen. Somit ist zwischen beiden keine klare Präferenz aus den Umfragedaten abzuleiten. In der quantitativen Evaluierung hingegen sind die Unterschiede zwischen SIFRank+ und PositionRank stärker ausgeprägt.

Abschließend kann trotz der Einschränkungen der quantitativen und qualitativen Evaluierung auf Basis übereinstimmender Tendenzen von F_1, MRR, MoverScore und den Umfrageergebnissen gefolgert werden, dass die LLM-basierten Verfahren GPT-4o und mT5-amazon unter den betrachteten Verfahren die besten Marginalien generieren. SIFRank+ ist hinsichtlich F_1 für Exact Matching sowie MRR das drittbeste Verfahren, befindet sich aber der Umfrage zufolge auf einem ähnlichen Niveau wie PositionRank. Dabei bestehen Diskrepanzen zwischen SIFRank+ und PositionRank hinsichtlich des deutlichen Unterschieds in der quantitativen und dem „Kopf-an-Kopf-Rennen“ in der qualitativen Evaluierung. SetTrans hat zwar den drittbesten MoverScore, gehört allerdings bezüglich der anderen Maße zu den schlechteren Verfahren und erzeugt in vielen Fällen keine Marginalie.

Unter den Baselines führt TF-IDF, obwohl es sich dabei um kein Keyphrase-Extraction-Verfahren handelt. Anders als die restlichen Baselines, die ausschließlich auf dem jeweiligen Dokument arbeiten, hat TF-IDF die korpusweite Dokumentenhäufigkeit zur Verfügung, was ein Grund für die besseren Ergebnisse sein könnte. Es scheint daher sinnvoll, TF-IDF auch bei zukünftigen Untersuchungen als Baseline heranzuziehen.

7 Diskussion und Ausblick

Diese Arbeit vermittelt einen ersten Eindruck, inwiefern sich Keyphrase-Prediction-Verfahren und LLMs für das in der Forschung bisher noch nicht betrachtete Problem der Erzeugung von Marginalien eignen. Im Folgenden werden Einschränkungen dieser Arbeit und Richtungen für zukünftige Weiterentwicklung und Forschung aufgezeigt.

7.1 Auswahl der Verfahren und deren Evaluierung

Die ausgewählten Verfahren bilden nur einen kleinen Teil einer Vielzahl an Keyphrase-Prediction-Verfahren ab. Es ist also möglich, dass andere Verfahren existieren, die sich besser für die Erzeugung von Marginalien eignen würden. Die Vorab-Einschätzung, welche Verfahren am besten geeignet sein könnten, musste für diese Arbeit aufgrund von Maßen wie dem F_1-Maß getroffen werden, welches auf lexikalische Übereinstimmung mit Referenz-Keyphrases beschränkt ist und somit sowohl für die Erzeugung von Keyphrases als auch Marginalien nur bedingt aussagekräftig ist. Die qualitative Evaluierung zeigt zwar teils Übereinstimmungen mit den Maßen der quantitativen Evaluierung, es bestehen aber auch Unterschiede.

Daher bedarf es für weitere Untersuchungen zur Erzeugung von Marginalien und besonders in der Forschung zur Keyphrase-Prediction allgemein zukünftig der Entwicklung und des vermehrten Einsatzes von Maßen, die semantische Ähnlichkeit quantifizieren. Es existieren bereits semantische Maße wie etwa MoverScore, diese werden jedoch in vielen Papern nicht zur Evaluierung herangezogen.

Außerdem nimmt die Evaluierung als Vergleich der Übereinstimmung zwischen Referenz und System-Output eine Objektivität und Vollständigkeit der Referenzen an, die so nicht notwendigerweise existieren muss, da die Qualität von Keyphrases intersubjektiv unterschiedlich beurteilt werden kann. Folglich wären zukünftige Untersuchungen zu Maßen, die den System-Output statt mit einer Referenz mit dem zugehörigen Dokument vergleichen, wünschenswert. In dieser Arbeit wurde versucht, den MoverScore dafür einzusetzen, dies stellte sich allerdings als nicht geeignet heraus. LLMs wie etwa GPT-4o könnten hierfür besser geeignet sein.

Bezüglich der qualitativen Evaluierung wäre eine Umfrage mit einem größeren Fragenkatalog, mehr Teilnehmern und mehr beantworteten Fragen hilfreich, um die Verfahren besser bewerten zu können. Während die im Rahmen dieser Arbeit durchgeführte Umfrage auf eine signifikante Überlegenheit von GPT-4o und Unterlegenheit von YAKE! gegenüber mT5-amazon, SIFRank+ und PositionRank hindeutet, ist besonders die Bewertung letzterer beider Verfahren weniger eindeutig. Durch eine größere Menge beantworteter Fragen könnten die Ergebnisse aussagekräftiger werden. Zusätzlich wäre die Aufnahme weiterer Verfahren wie TF-IDF oder WR-SetTrans interessant, sofern WR-SetTrans zukünftig fehlerfrei ausgeführt werden kann.

7.2 Verbesserung und Weiterentwicklung der Verfahren

WR-SetTrans konnte aus bislang unbekannten Gründen für die Erzeugung von Marginalien leider nicht funktionsfähig betrieben werden und SetTrans nur eingeschränkt. Aufgrund der vielversprechenden F_1-Werte in [5] ist es erstrebenswert, die Problemursachen zu identifizieren und in weiteren Untersuchungen für die Erzeugung von Marginalien einzusetzen.

Doch selbst die betrachteten Verfahren, die fehlerfrei funktionieren, könnten noch optimiert werden und dadurch möglicherweise bessere Marginalien erzeugen. Für die Baseline-Verfahren, aber auch SIFRank und SIFRank+ können teils POS, Länge und Aufbau der Kandidaten-Keyphrases spezifiziert werden. Hier könnte untersucht werden, welche Parameter die besten Ergebnisse hervorbringen. Für mT5 könnte Hyperparameter-Tuning durchgeführt werden, andere vortrainierte mT5-Modelle für das Fine-Tuning eingesetzt oder gänzlich andere LLMs getestet werden. Auch die Ergebnisse von GPT-4o lassen sich eventuell verbessern, indem ebenfalls Fine-Tuning mit dem Marginalien-Datensatz durchgeführt wird oder mittels Prompt Engineering.

Speziell für die Erzeugung von Marginalien könnten dedizierte, für dieses Problem entwickelte Verfahren möglicherweise vorteilhaft sein, die im Gegensatz zu den betrachteten Keyphrase-Prediction-Verfahren nicht nur dokumentenweise arbeiten, sondern dokumentenübergreifend. Im Fall der Marginalien bedeutet dies für die Bearbeitung des aktuellen Textabschnitts die Berücksichtigung anderer Textabschnitte und deren Marginalien, wie es mittels GPT-4o mit RAG in dieser Arbeit getestet wurde. Es ist aber noch nicht bekannt, ob solche dokumentenübergreifenden Verfahren tatsächlich einen signifikanten Vorteil bieten. Der nächste Schritt könnte daher sein, GPT-4o ohne RAG einzusetzen und die Unterschiede zu analysieren.

Eine einfache Form der dokumentenübergreifenden Bearbeitung wäre die Berücksichtigung bereits erzeugter Marginalien. Wird für einen Textabschnitt eine Liste an Marginalien erzeugt, könnte ausgehend von der Marginalie an Rangposition 1 geprüft werden, ob diese Marginalie bereits vorherigen Textabschnitten zugewiesen wurde. Falls dies zutrifft, wird die in der Rangfolge nächste Marginalie geprüft. Dies geschieht solange, bis eine noch nicht zugewiesene Marginalie identifiziert wird oder alle Marginalien der Liste bereits geprüft wurden.

7.3 Marginalien-Datensatz

Auch hinsichtlich der verwendeten Datenbasis besteht zukünftiges Verbesserungspotential. Der Marginalien-Datensatz wird automatisch extrahiert, um eine möglichst große Datenbasis zu erlangen. Dies erschwert jedoch die Qualitätssicherung, da nicht der gesamte Datensatz manuell überprüft werden kann. Für den Marginalien-Datensatz wurde zwar versucht, die Extraktion so robust wie möglich zu gestalten, dennoch können nicht alle Randfälle abgedeckt werden. Es bedarf daher der weiteren Verbesserung der Robustheit, als auch der Erweiterung des Marginalien-Datensatzes durch zusätzliche Lehrbücher.

Außerdem erfolgt die Extraktion speziell angepasst auf die HTML-Struktur der Lehrbücher des dpunkt.verlag, sodass für Lehrbücher anderer Verlage entsprechende Anpassungen notwendig sind, sofern deren HTML-Struktur abweicht.

Des Weiteren umfasst der Marginalien-Datensatz ausschließlich Lehrbücher aus dem Bereich der Informatik. Dadurch ist anzunehmen, dass besonders überwachte Verfahren, die auf dem Marginalien-Datensatz trainiert worden sind, auf domänenfremden Texten schlechter abschneiden werden. Aber auch die Baseline-Verfahren und SIFRank(+) erzielen für domänenfremde Texte vermutlich schlechtere Ergebnisse, da diese Verfahren die Kandidatenauswahl anhand der Verteilung der POS bzw. POS-Kombinationen im Marginalien-Datensatz treffen. In welchem Ausmaß diese Annahmen zutreffen, bedarf weiterer Untersuchung.

Nicht nur für den speziellen Fall der Erzeugung von Marginalien, sondern auch für Keyphrase-Prediction stellt Deutsch als zu verarbeitende Sprache eine Schwierigkeit dar, da die meisten Verfahren auf die englische Sprache ausgerichtet sind. Die nötigen Anpassungen können unterschiedlich umfangreich sein, erfordern aber bei überwachten Verfahren zumindest ein erneutes Training. Für die englische Sprache können die Verfahren meist direkt angewandt werden.

7.4 Neue Problemstellung

Von Keyphrase-Prediction-Verfahren prinzipbedingt nicht behandelt - aber relevant für die Erzeugung von Marginalien - ist außerdem die automatische Einteilung eines zusammenhängenden Textes in Abschnitte, für die Marginalien erzeugt werden sollen. Für jeden einzelnen Absatz eines Textes Marginalien zu erzeugen, kann unerwünscht sein, wenn aufeinanderfolgende Absätze auf geeignete Weise durch eine gemeinsame Marginalie beschrieben werden können. Ein ideales Verfahren zur Erzeugung von Marginalien würde demnach einen Text, z.B. ein Kapitel eines Buches, als Eingabe erhalten, diesen Text unter Zuhilfenahme bestehender Strukturen wie Absätzen und (Zwischen-)Überschriften in Sinnabschnitte unterteilen und für diese Sinnabschnitte Marginalien generieren. Für diese Einteilung in Sinnabschnitte könnten ebenfalls LLMs aufgrund ihres „Sprachverständnisses“ zum Einsatz kommen.

Während diese Arbeit das Problem der Erzeugung von Marginalien sicher noch nicht abschließend löst, stellt sie jedoch durch die beispielhafte Untersuchung eines Marginalien-Datensatzes und die Anwendung und Evaluierung ausgewählter Keyphrase-Prediction-Verfahren und den Vergleich mit LLMs einen ersten Schritt in diese Richtung dar. Es ist zu erwarten, dass mithilfe der zuvor genannten zukünftigen Weiterentwicklungen die Qualität der erzeugten Marginalien weiter verbessert werden kann. Außerdem würde weitere Forschung zu semantischen Maßen und vor allem deren weitgehende Etablierung in wissenschaftlichen Publikationen nicht nur im Bereich der Erzeugung von Marginalien nützlich sein, sondern auch einen aussagekräftigeren Vergleich von Verfahren der Keyphrase-Prediction und auch Text-Summarization ermöglichen.

Literatur

1. H. P. Luhn, "The automatic creation of literature abstracts," *IBM Journal of research and development*, vol. 2, no. 2, pp. 159–165, 1958.
2. M. Luo, B. Xue, and B. Niu, "A comprehensive survey for automatic text summarization: Techniques, approaches and perspectives," *Neurocomputing*, vol. 603, p. 128280, Oct. 2024.
3. R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (D. Lin and D. Wu, eds.), (Barcelona, Spain), pp. 404–411, Association for Computational Linguistics, July 2004.
4. K. S. Hasan and V. Ng, "Conundrums in unsupervised keyphrase extraction: making sense of the state-of-the-art," in *Coling 2010: Posters*, pp. 365–373, 2010.
5. B. Xie, J. Song, L. Shao, S. Wu, X. Wei, B. Yang, H. Lin, J. Xie, and J. Su, "From statistical methods to deep learning, automatic keyphrase prediction: A survey," *Information Processing & Management*, vol. 60, p. 103382, July 2023.
6. E. Papagiannopoulou and G. Tsoumakas, "A review of keyphrase extraction," *WIREs Data Mining and Knowledge Discovery*, vol. 10, Sept. 2019.
7. H. P. Luhn, "A statistical approach to mechanized encoding and searching of literary information," *IBM Journal of Research and Development*, vol. 1, pp. 309–317, Oct. 1957.
8. K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.
9. R. Campos, V. Mangaravite, A. Pasquali, A. M. Jorge, C. Nunes, and A. Jatowt, *A Text Feature Based Automatic Keyword Extraction Method for Single Documents*, pp. 684–691. Springer International Publishing, 2018.
10. V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Proceedings of the Soviet physics doklady*, 1966.
11. C. Florescu and C. Caragea, "Positionrank: An unsupervised approach to keyphrase extraction from scholarly documents," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 2017.
12. S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, pp. 107–117, Apr. 1998.
13. N. Giarelis and N. Karacapilidis, "Deep learning and embeddings-based approaches for keyphrase extraction: a literature review," *Knowledge and Information Systems*, July 2024.
14. K. Patel and C. Caragea, "Exploiting position and contextual word embeddings for keyphrase extraction from scientific papers," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 1585–1591, Association for Computational Linguistics, 2021.
15. I. Beltagy, K. Lo, and A. Cohan, "SciBERT: A pretrained language model for scientific text," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), (Hong Kong, China), pp. 3615–3620, Association for Computational Linguistics, Nov. 2019.
16. E. Papagiannopoulou and G. Tsoumakas, "Local word vectors guiding keyphrase extraction," *Information Processing & Management*, vol. 54, pp. 888–902, Nov. 2018.

17. J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.
18. Z. Sun, J. Tang, P. Du, Z.-H. Deng, and J.-Y. Nie, "Divgraphpointer: A graph pointer network for extracting diverse keyphrases," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19, pp. 755–764, ACM, July 2019.
19. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.
20. R. Meng, S. Zhao, S. Han, D. He, P. Brusilovsky, and Y. Chi, "Deep keyphrase generation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (R. Barzilay and M.-Y. Kan, eds.), (Vancouver, Canada), pp. 582–592, Association for Computational Linguistics, July 2017.
21. A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp. 216–223, 2003.
22. M. Krapivin, A. Autaeu, M. Marchese, *et al.*, "Large dataset for keyphrases extraction," 2009.
23. T. D. Nguyen and M.-Y. Kan, "Keyphrase extraction in scientific publications," in *International conference on Asian digital libraries*, pp. 317–326, Springer, 2007.
24. S. N. Kim, O. Medelyan, M.-Y. Kan, T. Baldwin, and L. Pingar, "Semeval-2010 task 5: Automatic keyphrase extraction from scientific," in *Proc. 5th Int. Workshop Semantic Eval*, pp. 21–26.
25. X. Shen, Y. Wang, R. Meng, and J. Shang, "Unsupervised deep keyphrase generation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 11303–11311, June 2022.
26. J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," in *Proceedings of the 1st Workshop on Representation Learning for NLP* (P. Blunsom, K. Cho, S. Cohen, E. Grefenstette, K. M. Hermann, L. Rimell, J. Weston, and S. W.-t. Yih, eds.), (Berlin, Germany), pp. 78–86, Association for Computational Linguistics, Aug. 2016.
27. H. Wu, B. Ma, W. Liu, T. Chen, and D. Nie, "Fast and constrained absent keyphrase generation by prompt-based learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 11495–11503, June 2022.
28. Duden Online-Wörterbuch, "Marginalie, die." `https://www.duden.de/rechtschreibung/Marginalie`. Zugriff am 1. Februar 2025.
29. G. Schäfer and M. Roßberg, *Netzsicherheit: Grundlagen Protokolle-Mobile drahtlose Kommunikation-Schutz von Kommunikationsinfrastrukturen*. dpunkt. verlag, 2014.
30. A. Johannsen, A. Kramer, H. Kostal, and E. Sadowicz, *Basiswissen für Softwareprojektmanager im klassischen und agilen Umfeld: Aus-und Weiterbildung zum ASQF® Certified Professional for Project Management (CPPM)*. dpunkt. verlag, 2017.
31. A. Spillner and T. Linz, *Basiswissen Softwaretest: Aus-und Weiterbildung zum Certified Tester–Foundation Level nach ISTQB-Standard*. dpunkt. verlag, 2019.
32. A. Geschonneck, *Computer-Forensik: Computerstraftaten erkennen, ermitteln, aufklären*. dpunkt. verlag, 2014.

33. W. Chen, H. P. Chan, P. Li, L. Bing, and I. King, "An integrated approach for keyphrase generation via exploring the power of retrieval and extraction," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), (Minneapolis, Minnesota), pp. 2846–2856, Association for Computational Linguistics, June 2019.
34. Y. Sun, H. Qiu, Y. Zheng, Z. Wang, and C. Zhang, "Sifrank: A new baseline for unsupervised keyphrase extraction based on pre-trained language model," *IEEE Access*, vol. 8, pp. 10896–10906, 2020.
35. L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel, "mT5: A massively multilingual pre-trained text-to-text transformer," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, eds.), (Online), pp. 483–498, Association for Computational Linguistics, June 2021.
36. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.
37. S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *International conference on learning representations*, 2017.
38. M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Association for Computational Linguistics, 2018.
39. B. Xie, X. Wei, B. Yang, H. Lin, J. Xie, X. Wang, M. Zhang, and J. Su, "WR-One2Set: Towards well-calibrated keyphrase generation," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (Y. Goldberg, Z. Kozareva, and Y. Zhang, eds.), (Abu Dhabi, United Arab Emirates), pp. 7283–7293, Association for Computational Linguistics, Dec. 2022.
40. J. Ye, T. Gui, Y. Luo, Y. Xu, and Q. Zhang, "One2set: Generating diverse keyphrases as a set," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, 2021.
41. X. Yuan, T. Wang, R. Meng, K. Thaker, P. Brusilovsky, D. He, and A. Trischler, "One size does not fit all: Generating and evaluating variable number of keyphrases," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2020.
42. Z. Zhao and P. Chen, *To Adapt or to Fine-Tune: A Case Study on Abstractive Summarization*, pp. 133–146. Springer International Publishing, 2022.
43. OpenAI, "Hello gpt-4o." `https://openai.com/index/hello-gpt-4o/`, 2024. Zugriff am 01.02.2025.
44. OpenAI et al., "Gpt-4 technical report," 2023.
45. Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2023.

46. T. Scialom, P.-A. Dray, S. Lamprier, B. Piwowarski, and J. Staiano, "MLSUM: The multilingual summarization corpus," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (B. Webber, T. Cohn, Y. He, and Y. Liu, eds.), (Online), pp. 8051–8067, Association for Computational Linguistics, Nov. 2020.
47. OpenAI, "Api-reference: Introduction." `https://platform.openai.com/docs/api-reference/introduction`. Zugriff am 01.02.2025.
48. OpenAI, "Api pricing." `https://openai.com/api/pricing/`. Zugriff am 01.02.2025.
49. OpenAI, "Batch api." `https://platform.openai.com/docs/guides/batch`, 2024. Zugriff am 01.02.2025.
50. OpenAI, "Rate limits." `https://platform.openai.com/docs/guides/rate-limits?tier=tier-one`, 2024. Zugriff am 01.02.2025.
51. D. Wu, D. Yin, and K.-W. Chang, "KPEval: Towards fine-grained semantic-based keyphrase evaluation," in *Findings of the Association for Computational Linguistics: ACL 2024* (L.-W. Ku, A. Martins, and V. Srikumar, eds.), (Bangkok, Thailand), pp. 1959–1981, Association for Computational Linguistics, Aug. 2024.
52. T. Zesch and I. Gurevych, "Approximate matching for evaluating keyphrase extraction," in *Proceedings of the International Conference RANLP-2009*, pp. 484–489, 2009.
53. E. Thomas and S. Vajjala, "Keyphrase generation: Lessons from a reproducibility study," in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy* (N. Calzolari, M. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, eds.), pp. 9720–9731, ELRA and ICCL, 2024.
54. E. M. Voorhees *et al.*, "The trec-8 question answering track report.," in *Trec*, vol. 99, pp. 77–82, 1999.
55. W. Zhao, M. Peyrard, F. Liu, Y. Gao, C. M. Meyer, and S. Eger, "MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), (Hong Kong, China), pp. 563–578, Association for Computational Linguistics, Nov. 2019.
56. A. Elo, *The Rating of Chessplayers: Past and Present*. Ishi Press International, 2008.