Zwischen Interpretation und Berechnung

Didaktische Exploration computerlinguistischer Verfahren mit augmentierten Transkripten von Verkaufsgesprächen

Paul Koop

Lehr- und Lernmaterial 2026

Zusammenfassung

Dieses Lehr- und Lernmaterial dient der didaktischen Exploration computerlinguistischer Verfahren auf der Grundlage der acht Transkripte von Verkaufsgesprächen. Im Unterschied zu den vorangegangenen ARS-Versionen 2.0 und 3.0, die auf interpretativ gebildeten Terminalzeichen basierten, wird hier der Schritt in Richtung automatischer Sprachverarbeitung vollzogen. Die Verfahren werden zu Demonstrationszwecken auf augmentierten Daten trainiert, um ihre Funktionsweise transparent zu machen. Der Fokus liegt auf dem didaktischen Erkenntnisgewinn, nicht auf empirischer Validität. Die Szenarien C (Computerlinguistische Integration) und D (Hybride Modellierung) werden schrittweise entwickelt und miteinander verglichen.

Inhaltsverzeichnis

1 Einleitung: Didaktische Ziele und methodologische Reflexion

Die vorangegangenen Versionen der Algorithmisch Rekursiven Sequenzanalyse (ARS 2.0 und 3.0) haben gezeigt, wie aus interpretativ gewonnenen Terminalzeichenketten formale Grammatiken induziert werden können. Diese Verfahren bleiben methodologisch kontrolliert: Die Kategorienbildung erfolgt durch qualitative Interpretation, die formalen Modelle explizieren lediglich die beobachtbaren Regularitäten.

Die nun folgenden Szenarien C und D wagen einen Schritt über diese methodologische Grenze hinaus. Sie explorieren, wie computerlinguistische Verfahren – insbesondere neuronale Netze, Word Embeddings und Topic Models – auf die acht Transkripte angewendet werden könnten, wenn man sie zu Demonstrationszwecken augmentiert.

Dieses Dokument ist als Lehr- und Lernmaterial konzipiert. Es verfolgt folgende didaktische Ziele:

1. **Verständnis neuronaler Architekturen**: Wie funktionieren Transformer, LSTM-Netze und Attention-Mechanismen auf Sequenzdaten?
2. **Data Augmentation als Technik**: Wie kann man mit kleinen Datensätzen umgehen, um die Funktionsweise von Verfahren zu demonstrieren?
3. **Vergleich verschiedener Modellierungsebenen**: Welche Unterschiede bestehen zwischen rein computerlinguistischen (C) und hybriden (D) Ansätzen?
4. **Methodologische Reflexion**: Wo liegen die Grenzen automatischer Verfahren im Vergleich zur interpretativen Kategorienbildung?

Alle hier vorgestellten Implementierungen arbeiten mit augmentierten Daten – die acht Originaltranskripte wurden künstlich vervielfacht, um das Training neuronaler Netze zu ermöglichen. Die Ergebnisse sind daher nicht empirisch valide, sondern dienen ausschließlich der didaktischen Veranschaulichung.

2 Die acht Transkripte: Rohdaten und Terminalzeichen

2.1 Die Rohdaten

Die folgenden acht Transkripte dokumentieren Verkaufsgespräche auf dem Aachener Marktplatz im Juni/Juli 1994. Sie bilden die empirische Grundlage aller folgenden Analysen.

2.1.1 Transkript 1 - Metzgerei

Datum: 28. Juni 1994, **Ort:** Metzgerei, Aachen, 11:00 Uhr

```
Kunde: Guten Tag!
Verk uferin: Guten Tag!
Kunde: Einmal von der groben Leberwurst, bitte.
Verk uferin: Wie viel darf's denn sein?
Kunde: Zwei hundert Gramm.
Verk uferin: Sonst noch etwas?
Kunde: Ja, dann noch ein St ck von dem Schwarzw lder Schinken.
Verk uferin: Wie gro  soll das St ck sein?
Kunde: So um die dreihundert Gramm.
Verk uferin: Das macht dann acht Mark zwanzig.
Kunde: Bitte.
Verk uferin: Danke und einen sch nen Tag noch!
Kunde: Danke, ebenfalls!
```

Listing 1: Transkript 1 - Rohdaten

2.1.2 Transkript 2 - Marktplatz (Kirschen)

Datum: 28. Juni 1994, **Ort:** Marktplatz, Aachen

```
Verk ufer: Kirschen kann jeder probieren hier!
Kunde 1: Ein halbes Kilo Kirschen, bitte.
Verk ufer: Ein halbes Kilo? Oder ein Kilo?
Verk ufer: Drei Mark, bitte.
Kunde 1: Danke sch n!
Verk ufer: Kirschen kann jeder probieren hier!
Kunde 2: Ein halbes Kilo, bitte.
Verk ufer: Drei Mark, bitte.
Kunde 2: Danke sch n!
```

Listing 2: Transkript 2 - Rohdaten

2.1.3 Transkript 3 - Fischstand

Datum: 28. Juni 1994, **Ort:** Fischstand, Marktplatz, Aachen

```
Kunde: Ein Pfund Seelachs, bitte.
Verk ufer: Seelachs, alles klar.
Verk ufer: Vier Mark neunzehn, bitte.
Kunde: Danke sch n!
```

Listing 3: Transkript 3 - Rohdaten

2.1.4 Transkript 4 - Gemüsestand (ausführlich)

Datum: 28. Juni 1994, **Ort:** Gemüsestand, Aachen, Marktplatz, 11:00 Uhr

```
Kunde: H ren Sie, ich nehme ein paar Champignons mit.
Verk ufer: Braune oder helle?
Kunde: Nehmen wir die hellen.
Verk ufer: Die sind beide frisch, keine Sorge.
Kunde: Wie ist es mit Pfifferlingen?
Verk ufer: Ah, die sind super!
Kunde: Kann ich die in Reissalat tun?
Verk ufer: Eher kurz anbraten in der Pfanne.
Kunde: Okay, mache ich.
Verk ufer: Sch nen Tag noch!
Kunde: Gleichfalls!
```

Listing 4: Transkript 4 - Rohdaten

2.1.5 Transkript 5 - Gemüsestand (mit KAV zu Beginn)

Datum: 26. Juni 1994, **Ort:** Gemüsestand, Aachen, Marktplatz, 11:00 Uhr

```
Kunde 1: Auf Wiedersehen!
Kunde 2: Ich h tte gern ein Kilo von den Granny Smith  pfeln  hier
    .
Verk ufer: Sonst noch etwas?
Kunde 2: Ja, noch ein Kilo Zwiebeln.
Verk ufer: Sechs Mark f nfundzwanzig, bitte.
Kunde 2: Auf Wiedersehen!
```

Listing 5: Transkript 5 - Rohdaten

2.1.6 Transkript 6 - Käseverkaufsstand

Datum: 28. Juni 1994, **Ort:** Käseverkaufsstand, Aachen, Marktplatz

```
Kunde 1: Guten Morgen!
Verk ufer: Guten Morgen!
Kunde 1: Ich h tte gerne f nfhundert Gramm holl ndischen Gouda.
Verk ufer: Am St ck?
Kunde 1: Ja, am St ck , bitte.
```

Listing 6: Transkript 6 - Rohdaten

2.1.7 Transkript 7 - Bonbonstand

Datum: 28. Juni 1994, **Ort:** Bonbonstand, Aachen, Marktplatz, 11:30 Uhr

```
Kunde: Von den gemischten h tte ich gerne hundert Gramm.
Verk ufer: F r zu Hause oder zum Mitnehmen?
Kunde: Zum Mitnehmen, bitte.
Verk ufer: F nfzig Pfennig, bitte.
Kunde: Danke!
```

Listing 7: Transkript 7 - Rohdaten

2.1.8 Transkript 8 - Bäckerei

Datum: 9. Juli 1994, **Ort:** Bäckerei, Aachen, 12:00 Uhr

```
(Schritte h rbar , Hintergrundger usche , teilweise unverst ndlich
    )
Kunde: Guten Tag!
(Unverst ndliche Begr  ung im Hintergrund)
Verk uferin: Einmal unser bester Kaffee, frisch gemahlen, bitte.
(Ger usche der Kaffeem hle , Verpackungsger usche)
Verk uferin: Sonst noch etwas?
Kunde: Ja, noch zwei St ck Obstsalat und ein Sch lchen Sahne.
Verk uferin: In Ordnung!
(Ger usche der Kaffeem hle , Papierger usche)
Verk uferin: Ein kleines Sch lchen Sahne, ja?
Kunde: Ja, danke.
(T rger usch , Lachen, Papierger usche)
Verk uferin: Keiner k mmert sich darum, die T ren zu  len .
Kunde: Ja, das ist immer so.
(Lachen, Ger usche von M nzen und Verpackung)
Verk uferin: Das macht vierzehn Mark und neunzehn Pfennig, bitte.
Kunde: Ich zahle in Kleingeld.
(Lachen und Ger usche von M nzen)
Verk uferin: Vielen Dank, sch nen Sonntag noch!
Kunde: Danke, Ihnen auch!
```

Listing 8: Transkript 8 - Rohdaten

2.2 Die Terminalzeichenketten (ARS 3.0)

Für die ARS 3.0 wurden diese Rohdaten in Terminalzeichenketten überführt, die als Grundlage für die hierarchische Grammatikinduktion dienten:

Tabelle 1: Terminalzeichenketten der acht Transkripte

Transkript	**Terminalzeichenkette**
1 (Metzgerei)	KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV
2 (Kirschen)	VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA
3 (Fischstand)	KBBd, VBBd, VAA, KAA
4 (Gemüse)	KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV
5 (Gemüse KAV)	KAV, KBBd, VBBd, KBBd, VAA, KAV
6 (Käse)	KBG, VBG, KBBd, VBBd, KAA
7 (Bonbon)	KBBd, VBBd, KBA, VAA, KAA
8 (Bäckerei)	KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV

Die Bedeutung der Terminalzeichen:

- **KBG**: Kunden-Gruß
- **VBG**: Verkäufer-Gruß
- **KBBd**: Kunden-Bedarf (konkret)
- **VBBd**: Verkäufer-Nachfrage
- **KBA**: Kunden-Antwort
- **VBA**: Verkäufer-Reaktion
- **KAE**: Kunden-Erkundigung
- **VAE**: Verkäufer-Auskunft
- **KAA**: Kunden-Abschluss
- **VAA**: Verkäufer-Abschluss
- **KAV**: Kunden-Verabschiedung
- **VAV**: Verkäufer-Verabschiedung

3 Szenario C: Computerlinguistische Integration

Szenario C realisiert eine vollständig computerlinguistische Modellierung der acht Transkripte. Es umfasst vier Komponenten:

1. **Speech Act Recognition**: Automatische Erkennung der Sprechakte aus den Rohdaten
2. **Word Embeddings**: Vektorielle Repräsentation der Äußerungen
3. **Topic Modeling**: Identifikation thematischer Verschiebungen
4. **Rhetorical Structure Theory (RST)**: Analyse der argumentativen Struktur

3.1 Didaktische Augmentierung

Da neuronale Netze für ihr Training große Datenmengen benötigen, werden die acht Transkripte zu Demonstrationszwecken augmentiert:

```
def augment_transcripts_for_teaching(transcripts, factor=20):
    """
    Augmentiert die acht Transkripte f r didaktische Zwecke.

    Didaktischer Hinweis: Diese Augmentierung dient
       ausschlie lich der
    Veranschaulichung der Methodik. Die resultierenden Daten
       sind nicht
    empirisch valide, sondern erm glichen lediglich die
       Demonstration
    der Funktionsweise neuronaler Verfahren.
    """
    augmented = []

    # 1. Basis-Augmentierung: einfaches Kopieren
    for _ in range(factor):
        augmented.extend(transcripts)

    # 2. Syntaktische Variationen (didaktisch kontrolliert)
    import copy
    import random

    for transcript in transcripts:
```

```
        for _ in range(factor // 4):
            var = copy.deepcopy(transcript)
            # Vertausche zwei benachbarte    uerungen    (
               selten)
            if len(var) > 3 and random.random() < 0.1:
                idx = random.randint(0, len(var)-2)
                var[idx], var[idx+1] = var[idx+1], var[idx]
            augmented.append(var)

    # 3. Lexikalische Variationen (Synonyme)
    synonyms = {
        'Guten Tag': ['Guten Morgen', 'Hallo', 'Guten Abend'
           ],
        'Danke': ['Vielen Dank', 'Danke sch n', 'Merci'],
        'Bitte': ['Bitte sehr', 'Gern geschehen']
    }

    # Hier k nnten weitere Variationen implementiert werden

    return augmented
```

Listing 9: Data Augmentation für Lehrzwecke

3.2 Speech Act Recognition mit Transformer-Modellen

Die automatische Erkennung der Sprechakte erfolgt mit einem feinabgestimmten BERT-Modell:

```
"""
Speech Act Recognition mit transformer-basierten Modellen
Didaktische Implementierung f r Lehrzwecke
"""

import torch
import torch.nn as nn
from transformers import BertTokenizer, BertModel
import numpy as np
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader

class SpeechActDataset(Dataset):
```

```
    """Dataset f r Speech Act Recognition"""
    def __init__(self, utterances, labels, tokenizer,
        max_length=128):
        self.utterances = utterances
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.utterances)

    def __getitem__(self, idx):
        utterance = self.utterances[idx]
        label = self.labels[idx]

        encoding = self.tokenizer(
            utterance,
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].
                flatten(),
            'label': torch.tensor(label, dtype=torch.long)
        }

class BertSpeechActClassifier(nn.Module):
    """
    BERT-basierter Klassifikator f r Sprechakte
    Didaktisch vereinfachte Architektur
    """
    def __init__(self, num_classes=12, dropout=0.3):
        super().__init__()
        self.bert = BertModel.from_pretrained('bert-base-
            german-cased')
        self.dropout = nn.Dropout(dropout)
```

```
        self.classifier = nn.Linear(768, num_classes)

        # Freeze BERT layers f r didaktische Zwecke (schnelleres Training)
        for param in self.bert.parameters():
            param.requires_grad = False

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs.pooler_output
        dropped = self.dropout(pooled_output)
        logits = self.classifier(dropped)
        return logits

def prepare_speech_act_data(transcripts, terminal_chains):
    """
    Bereitet die Daten f r das Speech Act Training vor
    """
    utterances = []
    labels = []

    # Extrahiere alle    uerungen    aus den Rohdaten
    # Hier vereinfacht: Mapping der Terminalzeichen auf Sprechakte
    for trans, chain in zip(transcripts, terminal_chains):
        # In einer vollst ndigen Implementierung m ssten die Rohdaten geparst werden
        # F r didaktische Zwecke verwenden wir die Terminalzeichen direkt
        for symbol in chain:
            utterances.append(f"Beispiel u erung f r {symbol}")
            labels.append(symbol)

    # Label-Encoding
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(labels)

    return utterances, y_encoded, label_encoder
```

```python

def train_speech_act_model(utterances, labels, epochs=10):
    """
    Trainiert das Speech Act Recognition Modell
    """
    tokenizer = BertTokenizer.from_pretrained('bert-base-german-cased')
    dataset = SpeechActDataset(utterances, labels, tokenizer)
    dataloader = DataLoader(dataset, batch_size=8, shuffle=True)

    model = BertSpeechActClassifier(num_classes=len(set(labels)))
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
    criterion = nn.CrossEntropyLoss()

    print("\n=== Speech Act Recognition Training (didaktisch) ===")
    for epoch in range(epochs):
        total_loss = 0
        for batch in dataloader:
            optimizer.zero_grad()
            outputs = model(batch['input_ids'], batch['attention_mask'])
            loss = criterion(outputs, batch['label'])
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        print(f"Epoch {epoch+1}: Loss = {total_loss/len(dataloader):.4f}")

    return model, tokenizer, label_encoder

# Didaktischer Hinweis
print("\n" + "="*70)
print("DIDAKTISCHER HINWEIS ZUR SPEECH ACT RECOGNITION")
print("="*70)
print("Die hier gezeigte Implementierung verwendet augmentierte")
```

```
print("Daten und dient ausschlie lich Lehrzwecken. Die automatische")
print("Erkennung von Sprechakten w rde in der Praxis:")
print("      Millionen von annotierten Trainingsdaten ben tigen")
print("      Auf spezifische Dom nen (Verkaufsgespr che) feinabgestimmt werden")
print("      Mit erheblichen Unsicherheiten behaftet sein")
```

Listing 10: Speech Act Recognition mit BERT

3.3 Word Embeddings und semantische Ähnlichkeit

Für die Quantifizierung semantischer Ähnlichkeit werden vortrainierte Word Embeddings verwendet:

```
"""
Word Embeddings f r semantische    hnlichkeitsanalysen
Didaktische Implementierung mit vortrainierten Modellen
"""

from sentence_transformers import SentenceTransformer
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import seaborn as sns

class SemanticAnalyzer:
    """
    Analysiert semantische   hnlichkeiten   zwischen uerungen
    """
    def __init__(self, model_name='paraphrase-multilingual-MiniLM-L12-v2'):
        print(f"Lade vortrainiertes Modell: {model_name}")
        self.model = SentenceTransformer(model_name)
        self.embeddings = {}

    def encode_utterances(self, utterances):
        """
        Erzeugt Embeddings f r eine Liste von    uerungen
```

```
        """
        embeddings = self.model.encode(utterances)
        for utt, emb in zip(utterances, embeddings):
            self.embeddings[utt] = emb
        return embeddings

    def similarity_matrix(self, utterances):
        """
        Berechnet die hnlichkeitsmatrix f r alle
            uerungen
        """
        embeddings = self.encode_utterances(utterances)
        sim_matrix = cosine_similarity(embeddings)
        return sim_matrix

    def find_similar(self, query, utterances, top_k=5):
        """
        Findet die hnlichsten uerungen zu einer Query
        """
        query_emb = self.model.encode([query])[0]
        utt_embs = self.encode_utterances(utterances)

        similarities = cosine_similarity([query_emb],
            utt_embs)[0]
        top_indices = np.argsort(similarities)[-top_k:][::-1]

        results = []
        for idx in top_indices:
            results.append({
                'utterance': utterances[idx],
                'similarity': similarities[idx]
            })

        return results

    def visualize_similarity(self, utterances, labels=None):
        """
        Visualisiert die hnlichkeitsmatrix als Heatmap
        """
        sim_matrix = self.similarity_matrix(utterances)
```

```
62
63          plt.figure(figsize=(12, 10))
64          sns.heatmap(sim_matrix,
65                      xticklabels=labels if labels else range(
                            len(utterances)),
66                      yticklabels=labels if labels else range(
                            len(utterances)),
67                      cmap='viridis', vmin=0, vmax=1)
68          plt.title('Semantische   hnlichkeit   zwischen
                uerungen   ')
69          plt.tight_layout()
70          plt.savefig('semantic_similarity.png', dpi=150)
71          plt.show()
72
73  # Didaktisches Beispiel
74  def demonstrate_semantic_analysis():
75      """
76      Demonstriert die semantische Analyse an Beispielen
77      """
78      analyzer = SemanticAnalyzer()
79
80      # Beispiel-   uerungen    aus den Transkripten
81      utterances = [
82          "Guten Tag!",
83          "Guten Morgen!",
84          "Einmal Leberwurst, bitte.",
85          "Ich h tte gerne Wurst.",
86          "Danke sch n!",
87          "Vielen Dank!",
88          "Auf Wiedersehen!",
89          "Tsch ss!"
90      ]
91
92      print("\n=== Semantische    hnlichkeitsanalyse    ===")
93
94      #    hnlichkeitsmatrix    berechnen
95      sim_matrix = analyzer.similarity_matrix(utterances)
96
97      #   hnlichste      uerungen    f r "Guten Tag!"
98      similar = analyzer.find_similar("Guten Tag!", utterances,
```

```
        top_k=3)
    print("\n hnlichste zu 'Guten Tag!':")
    for r in similar:
        print(f"  {r['utterance']}: {r['similarity']:.3f}")

    # Visualisierung
    analyzer.visualize_similarity(utterances, utterances)

    return analyzer

# Didaktischer Hinweis
print("\n" + "="*70)
print("DIDAKTISCHER HINWEIS ZU WORD EMBEDDINGS")
print("="*70)
print("Die verwendeten Embeddings wurden auf gro en Korpora")
print("(Wikipedia, Nachrichten, Webtexte) vortrainiert. Sie")
print("erfassen allgemeinsprachliche  hnlichkeiten , nicht die")
print("spezifischen Kategorien der Verkaufsgespr che.")
```

Listing 11: Semantische Ähnlichkeit mit Word Embeddings

3.4 Topic Modeling mit BERTopic

Für die Identifikation thematischer Verschiebungen wird BERTopic verwendet:

```
"""
Topic Modeling zur Identifikation thematischer Verschiebungen
Didaktische Implementierung mit BERTopic
"""

from bertopic import BERTopic
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import matplotlib.pyplot as plt

class TranscriptTopicModeler:
    """
    F hrt Topic Modeling auf den Transkripten durch
    """
```

```
15      def __init__(self):
16          self.model = None
17          self.topics = None
18          self.probs = None
19
20      def prepare_documents(self, transcripts):
21          """
22          Bereitet die Transkripte als Dokumente f r Topic
                Modeling vor
23          """
24          documents = []
25          metadata = []
26
27          for i, transcript in enumerate(transcripts, 1):
28              # Jedes Transkript als ein Dokument
29              doc = ' '.join(transcript)
30              documents.append(doc)
31              metadata.append(f'Transkript {i}')
32
33              # Alternativ: Jede    uerung    als Dokument
34              # for j, utterance in enumerate(transcript):
35              #     documents.append(utterance)
36              #     metadata.append(f'T{i}_U{j}')
37
38          return documents, metadata
39
40      def fit_model(self, documents):
41          """
42          Trainiert das Topic Model
43          """
44          # Benutzerdefinierte Stopw rter
45          stopwords = ['bitte', 'danke', 'gern', 'mal', 'noch',
                'ja', 'nein']
46          vectorizer = CountVectorizer(stop_words=stopwords)
47
48          self.model = BERTopic(
49              embedding_model="paraphrase-multilingual-MiniLM-
                    L12-v2",
50              vectorizer_model=vectorizer,
51              verbose=True,
```

```
52              nr_topics='auto'
53          )
54
55          self.topics, self.probs = self.model.fit_transform(
                documents)
56          return self.topics, self.probs
57
58      def visualize_topics(self):
59          """
60          Visualisiert die gefundenen Themen
61          """
62          if self.model is None:
63              return
64
65          fig = self.model.visualize_topics()
66          fig.write_html("topic_visualization.html")
67
68          # Statistik
69          topic_counts = pd.Series(self.topics).value_counts()
70          print("\n=== Themenverteilung ===")
71          for topic, count in topic_counts.items():
72              if topic == -1:
73                  print(f"Outlier: {count} Dokumente")
74              else:
75                  words = self.model.get_topic(topic)[:5]
76                  words_str = ', '.join([w for w, _ in words])
77                  print(f"Thema {topic}: {count} Dokumente - {
                        words_str}")
78
79      def visualize_topics_over_time(self, documents,
            timestamps):
80          """
81          Visualisiert Themenentwicklung  ber  die Zeit
82          """
83          if self.model is None:
84              return
85
86          topics_over_time = self.model.topics_over_time(
87              documents,
88              timestamps,
```

```
            nr_bins=8
        )

        fig = self.model.visualize_topics_over_time(
            topics_over_time)
        fig.write_html("topics_over_time.html")

def demonstrate_topic_modeling(transcripts):
    """
    Demonstriert Topic Modeling an den Transkripten
    """
    modeler = TranscriptTopicModeler()
    documents, metadata = modeler.prepare_documents(
        transcripts)

    print("\n=== Topic Modeling der acht Transkripte ===")
    topics, probs = modeler.fit_model(documents)

    for i, (doc, topic, prob, meta) in enumerate(zip(
        documents, topics, probs, metadata)):
        if topic != -1:
            words = modeler.model.get_topic(topic)[:3]
            words_str = ', '.join([w for w, _ in words])
            print(f"{meta}: Thema {topic} (Confidence: {prob
                :.2f}) - {words_str}")
        else:
            print(f"{meta}: Kein klares Thema (Outlier)")

    modeler.visualize_topics()
    return modeler

# Didaktischer Hinweis
print("\n" + "="*70)
print("DIDAKTISCHER HINWEIS ZUM TOPIC MODELING")
print("="*70)
print("Topic Modeling identifiziert latente Themen in
    Textkorpora.")
print("Bei nur acht Dokumenten ist die Themenfindung instabil
    .")
print("Die Ergebnisse dienen daher nur der Veranschaulichung
```

```
    der")
print("Methodik, nicht der inhaltlichen Analyse.")
```

Listing 12: Topic Modeling mit BERTopic

3.5 Rhetorical Structure Theory (RST)

Für die Analyse der argumentativen Struktur wird ein RST-Parser implementiert:

```
"""
Rhetorical Structure Theory (RST) Analyse
Didaktische Implementierung f r Sequenzdaten
"""

import networkx as nx
import matplotlib.pyplot as plt
from collections import defaultdict

class RSTRelation:
    """Repr sentiert eine RST-Relation zwischen Textsegmenten"""
    def __init__(self, type_name, nucleus, satellite, direction='nucleus-satellite'):
        self.type = type_name  # z.B. 'Elaboration', 'Contrast', 'Cause'
        self.nucleus = nucleus  # Zentrales Segment
        self.satellite = satellite  # Unterst tzendes Segment
        self.direction = direction

class SimpleRSTParser:
    """
    Einfacher RST-Parser f r didaktische Zwecke
    Basierend auf cue phrases und strukturellen Mustern
    """

    # Cue Phrases f r verschiedene Relationen
    cue_phrases = {
        'Elaboration': ['zum Beispiel', 'insbesondere', 'n mlich', 'genauer'],
```

```
        'Contrast': ['aber', 'jedoch', 'hingegen', 'dagegen'],
        'Cause': ['weil', 'da', 'denn', 'deshalb', 'daher'],
        'Condition': ['wenn', 'falls', 'sofern'],
        'Purpose': ['um zu', 'damit'],
        'Sequence': ['dann', 'danach', 'zuerst', 'schlie lich']
    }

    def __init__(self):
        self.relations = []
        self.graph = nx.DiGraph()

    def segment_transcript(self, transcript):
        """
        Segmentiert ein Transkript in elementare Diskurseinheiten (EDUs)
        Vereinfacht: Jede    uerung    ist eine EDU
        """
        return transcript

    def identify_relations(self, segments):
        """
        Identifiziert RST-Relationen zwischen Segmenten
        Didaktisch vereinfachte Implementierung
        """
        relations = []

        for i in range(len(segments)-1):
            current = segments[i]
            next_seg = segments[i+1]

            # Pr fe auf Cue Phrases
            for rel_type, cues in self.cue_phrases.items():
                for cue in cues:
                    if cue in current.lower() or cue in next_seg.lower():
                        relations.append(RSTRelation(
                            type_name=rel_type,
                            nucleus=i,
```

```
                            satellite=i+1
                        ))
                        break

            # Standard: Sequenz-Relation
            if i < len(segments)-1:
                relations.append(RSTRelation(
                    type_name='Sequence',
                    nucleus=i,
                    satellite=i+1
                ))

        return relations

    def build_tree(self, segments, relations):
        """
        Baut einen RST-Baum aus den identifizierten
           Relationen
        """
        self.graph.clear()

        # Knoten hinzuf gen
        for i, seg in enumerate(segments):
            self.graph.add_node(i, text=seg[:30] + '...' if
               len(seg) > 30 else seg)

        # Kanten hinzuf gen
        for rel in relations:
            self.graph.add_edge(rel.nucleus, rel.satellite,
                                relation=rel.type)

        return self.graph

    def parse(self, transcript):
        """
        Vollst ndige RST-Analyse eines Transkripts
        """
        segments = self.segment_transcript(transcript)
        relations = self.identify_relations(segments)
        tree = self.build_tree(segments, relations)
```

```
101
102         return {
103             'segments': segments,
104             'relations': relations,
105             'tree': tree
106         }
107
108     def visualize(self, title="RST-Struktur"):
109         """
110         Visualisiert den RST-Baum
111         """
112         pos = nx.spring_layout(self.graph)
113         plt.figure(figsize=(12, 8))
114
115         # Knoten zeichnen
116         nx.draw_networkx_nodes(self.graph, pos, node_color='
                lightblue',
117                                node_size=500)
118
119         # Kanten zeichnen mit Relationstyp als Label
120         for edge in self.graph.edges(data=True):
121             nx.draw_networkx_edges(self.graph, pos, [(edge
                    [0], edge[1])])
122             nx.draw_networkx_edge_labels(
123                 self.graph, pos,
124                 {(edge[0], edge[1]): edge[2]['relation']}
125             )
126
127         # Knotenlabels
128         labels = {node: f"{node}: {self.graph.nodes[node]['
                text']}"
129                   for node in self.graph.nodes()}
130         nx.draw_networkx_labels(self.graph, pos, labels,
                font_size=8)
131
132         plt.title(title)
133         plt.axis('off')
134         plt.tight_layout()
135         plt.savefig('rst_structure.png', dpi=150)
136         plt.show()
```

```
137
138 def demonstrate_rst_analysis(transcripts):
139     """
140     Demonstriert RST-Analyse an den Transkripten
141     """
142     parser = SimpleRSTParser()
143
144     print("\n=== RST-Analyse der Transkripte ===")
145
146     for i, transcript in enumerate(transcripts, 1):
147         print(f"\nTranskript {i}:")
148         result = parser.parse(transcript)
149
150         # Zeige identifizierte Relationen
151         for rel in result['relations'][:5]:  # Nur erste 5
152             seg1 = result['segments'][rel.nucleus][:20] + '
                ...'
153             seg2 = result['segments'][rel.satellite][:20] + '
                ...'
154             print(f"  {rel.type}: {seg1}     {seg2}")
155
156         if i == 1:  # Nur erstes Transkript visualisieren
157             parser.visualize(f"RST-Struktur Transkript {i}")
158
159     return parser
160
161 # Didaktischer Hinweis
162 print("\n" + "="*70)
163 print("DIDAKTISCHER HINWEIS ZUR RST-ANALYSE")
164 print("="*70)
165 print("Die hier implementierte RST-Analyse ist stark
    vereinfacht.")
166 print("Ein vollst ndiger RST-Parser w rde:")
167 print("      Aufw ndige manuelle Annotation erfordern")
168 print("      Mit trainierten neuronalen Modellen arbeiten")
169 print("      Mehrere Hierarchieebenen von Diskursrelationen
    ber cksichtigen")
```

Listing 13: Rhetorical Structure Theory Parser

3.6 Integration der Komponenten in Szenario C

Die vollständige Integration aller Komponenten in Szenario C:

```
"""
Szenario C: Vollst ndige computerlinguistische Integration
Didaktische Implementierung f r Lehrzwecke
"""

import os
import json
from datetime import datetime

class ScenarioC:
    """
    Integriert alle computerlinguistischen Komponenten:
    - Speech Act Recognition
    - Word Embeddings / Semantische Analyse
    - Topic Modeling
    - RST-Analyse
    """

    def __init__(self, transcripts, terminal_chains):
        self.transcripts = transcripts
        self.terminal_chains = terminal_chains
        self.results = {}

        print("\n" + "="*70)
        print("SZENARIO C: COMPUTERLINGUISTISCHE INTEGRATION")
        print("="*70)
        print("\nDieses Szenario demonstriert die Anwendung")
        print("computerlinguistischer Verfahren auf die acht")
        print("Transkripte. Alle Ergebnisse dienen")
        print("Zwecken und sind nicht empirisch valide.\n")

    def run_speech_act_recognition(self):
        """
        F hrt die Speech Act Recognition aus
```

```
        """
        print("\n--- Speech Act Recognition ---")
        utterances, labels, encoder = prepare_speech_act_data(
            self.transcripts, self.terminal_chains
        )

        model, tokenizer, label_encoder =
            train_speech_act_model(
            utterances, labels, epochs=5
        )

        self.results['speech_act'] = {
            'model': model,
            'tokenizer': tokenizer,
            'label_encoder': label_encoder,
            'num_classes': len(label_encoder.classes_)
        }

        return self.results['speech_act']

    def run_semantic_analysis(self):
        """
        F hrt die semantische   hnlichkeitsanalyse   aus
        """
        print("\n--- Semantische   hnlichkeitsanalyse   ---")
        analyzer = SemanticAnalyzer()

        # Sammle alle   uerungen
        all_utterances = []
        for transcript in self.transcripts:
            all_utterances.extend(transcript)

        #   hnlichkeitsmatrix
        sim_matrix = analyzer.similarity_matrix(
            all_utterances[:20])  # Nur erste 20

        self.results['semantic'] = {
            'analyzer': analyzer,
            'utterances': all_utterances,
```

```
72              'similarity_matrix': sim_matrix
73          }
74
75          return self.results['semantic']
76
77      def run_topic_modeling(self):
78          """
79          F hrt das Topic Modeling aus
80          """
81          print("\n--- Topic Modeling ---")
82          modeler = TranscriptTopicModeler()
83          documents, metadata = modeler.prepare_documents(self.
                transcripts)
84          topics, probs = modeler.fit_model(documents)
85          modeler.visualize_topics()
86
87          self.results['topic'] = {
88              'modeler': modeler,
89              'topics': topics,
90              'probabilities': probs,
91              'documents': documents,
92              'metadata': metadata
93          }
94
95          return self.results['topic']
96
97      def run_rst_analysis(self):
98          """
99          F hrt die RST-Analyse aus
100         """
101         print("\n--- RST-Analyse ---")
102         parser = SimpleRSTParser()
103
104         rst_results = []
105         for i, transcript in enumerate(self.transcripts, 1):
106             result = parser.parse(transcript)
107             rst_results.append({
108                 'transcript_id': i,
109                 'segments': result['segments'],
110                 'relations': [(r.type, r.nucleus, r.satellite
```

```
                    ) for r in result['relations']]
            })

            if i == 1:
                parser.visualize(f"RTS-Struktur Transkript {i}")

        self.results['rst'] = rst_results
        return rst_results

    def run_all(self):
        """
        F hrt alle Analysen aus
        """
        self.run_speech_act_recognition()
        self.run_semantic_analysis()
        self.run_topic_modeling()
        self.run_rst_analysis()

        # Zusammenfassung
        print("\n" + "="*70)
        print("ZUSAMMENFASSUNG SZENARIO C")
        print("="*70)
        print(f"    Speech Act Recognition: {self.results['speech_act']['num_classes']} Klassen")
        print(f"    Semantische Analyse: {len(self.results['semantic']['utterances'])}    uerungen   ")
        print(f"    Topic Modeling: {len(set(self.results['topic']['topics']))} Themen")
        print(f"    RST-Analyse: {len(self.results['rst'])} Transkripte analysiert")

        return self.results

# Didaktische Ausf hrung
def run_scenario_c_demonstration():
    """
    F hrt die vollst ndige Demonstration von Szenario C aus
    """
    # Lade die Transkripte
```

```
145     from ars_data import transcripts, terminal_chains
146
147     # Augmentiere die Daten f r didaktische Zwecke
148     augmented_transcripts = augment_transcripts_for_teaching(
            transcripts, factor=10)
149     augmented_chains = augment_transcripts_for_teaching(
            terminal_chains, factor=10)
150
151     print("\n" + "="*70)
152     print("DIDAKTISCHE AUGMENTIERUNG")
153     print("="*70)
154     print(f"Original: {len(transcripts)} Transkripte")
155     print(f"Augmentiert: {len(augmented_transcripts)}
            Transkripte")
156
157     # F hre Szenario C aus
158     scenario = ScenarioC(augmented_transcripts,
            augmented_chains)
159     results = scenario.run_all()
160
161     # Speichere Ergebnisse
162     with open('scenario_c_results.json', 'w') as f:
163         # Konvertiere nicht-serialisierbare Objekte
164         serializable = {
165             'speech_act': {'num_classes': results['speech_act
                ']['num_classes']},
166             'semantic': {'num_utterances': len(results['
                semantic']['utterances'])},
167             'topic': {'num_topics': len(set(results['topic'][
                'topics']))},
168             'rst': results['rst']
169         }
170         json.dump(serializable, f, indent=2)
171
172     print("\nErgebnisse gespeichert in 'scenario_c_results.
            json'")
173
174     return results
175
176 if __name__ == "__main__":
```

```
177     run_scenario_c_demonstration()
```

Listing 14: Szenario C - Vollständige Integration

4 Szenario D: Hybride Modellierung

Szenario D integriert computerlinguistische Verfahren mit den interpretativ gebildeten Kategorien der ARS 3.0. Es überspringt die vollständige Automatisierung der Kategorienbildung (Szenario C) und nutzt die neuen Verfahren komplementär.

4.1 CRF für sequenzielle Abhängigkeiten

Conditional Random Fields modellieren Abhängigkeiten der Sprechakte vom weiteren Kontext:

```
"""
Conditional Random Fields (CRF) f r sequenzielle
   Abh ngigkeiten
Didaktische Implementierung mit sklearn-crfsuite
"""

import sklearn_crfsuite
from sklearn_crfsuite import metrics
import numpy as np

class CRFSequenceModel:
    """
    CRF-Modell f r die Sequenzmodellierung der
       Terminalzeichen
    """

    def __init__(self):
        self.crf = sklearn_crfsuite.CRF(
            algorithm='lbfgs',
            c1=0.1,  # L1-Regularisierung
            c2=0.1,  # L2-Regularisierung
            max_iterations=100,
            all_possible_transitions=True
        )
        self.label_encoder = None
```

```
24
25      def word2features(self, tokens, i):
26          """
27          Erzeugt Features f r Position i in der Sequenz
28          """
29          word = tokens[i]
30
31          features = {
32              'bias': 1.0,
33              'word': word,
34              'word.is_first': i == 0,
35              'word.is_last': i == len(tokens) - 1,
36              'word.prefix_K': word.startswith('K'),
37              'word.prefix_V': word.startswith('V'),
38              'word.suffix_A': word.endswith('A'),
39              'word.suffix_B': word.endswith('B'),
40              'word.suffix_E': word.endswith('E'),
41              'word.suffix_G': word.endswith('G'),
42              'word.suffix_V': word.endswith('V'),
43          }
44
45          # Kontext-Features
46          if i > 0:
47              word_prev = tokens[i-1]
48              features.update({
49                  '-1:word': word_prev,
50                  '-1:word.prefix_K': word_prev.startswith('K')
                        ,
51                  '-1:word.prefix_V': word_prev.startswith('V')
                        ,
52                  '-1:word.suffix_A': word_prev.endswith('A'),
53              })
54          else:
55              features['BOS'] = True
56
57          if i < len(tokens) - 1:
58              word_next = tokens[i+1]
59              features.update({
60                  '+1:word': word_next,
61                  '+1:word.prefix_K': word_next.startswith('K')
```

```
                ,
            '+1:word.prefix_V': word_next.startswith('V')
                ,
            '+1:word.suffix_A': word_next.endswith('A'),
        })
    else:
        features['EOS'] = True

    return features

def extract_features(self, sequences):
    """
    Extrahiert Features f r alle Sequenzen
    """
    X = []
    for seq in sequences:
        X.append([self.word2features(seq, i) for i in
            range(len(seq))])
    return X

def fit(self, sequences, labels):
    """
    Trainiert das CRF-Modell
    """
    X = self.extract_features(sequences)
    self.crf.fit(X, labels)
    return self

def predict(self, sequences):
    """
    Sagt Labels f r neue Sequenzen vorher
    """
    X = self.extract_features(sequences)
    return self.crf.predict(X)

def evaluate(self, test_sequences, test_labels):
    """
    Evaluiert das Modell
    """
    pred = self.predict(test_sequences)
```

```
99
100         # Flatten f r Metriken
101         y_true = [label for seq in test_labels for label in
                seq]
102         y_pred = [label for seq in pred for label in seq]
103
104         return {
105             'accuracy': np.mean(np.array(y_true) == np.array(
                    y_pred)),
106             'classification_report': metrics.
                    flat_classification_report(
107                 test_labels, pred, labels=sorted(set(y_true))
108             )
109         }
110
111 def demonstrate_crf(terminal_chains):
112     """
113     Demonstriert CRF-Modellierung auf den Terminalzeichen
114     """
115     print("\n=== CRF-Modellierung der Terminalzeichen ===")
116
117     # Train-Test-Split (didaktisch)
118     train_size = int(len(terminal_chains) * 0.7)
119     train_chains = terminal_chains[:train_size]
120     test_chains = terminal_chains[train_size:]
121
122     # Features extrahieren
123     model = CRFSequenceModel()
124     X_train = model.extract_features(train_chains)
125
126     # Training
127     print(f"Trainiere CRF mit {len(train_chains)} Sequenzen
            ...")
128     model.fit(train_chains, train_chains)  # Labels sind die
            Sequenzen selbst
129
130     # Evaluation
131     results = model.evaluate(test_chains, test_chains)
132     print(f"\nGenauigkeit: {results['accuracy']:.3f}")
133
```

```
    return model
```

Listing 15: CRF für sequenzielle Abhängigkeiten

4.2 Transformer-Embeddings als Ergänzung

Transformer-Embeddings werden zusätzlich zu den kategorialen Terminalzeichen verwendet:

```
"""
Transformer-Embeddings als Erg nzung zu kategorialen
   Terminalzeichen
"""

import torch
import numpy as np
from sentence_transformers import SentenceTransformer

class TerminalEmbeddingEnricher:
    """
    Erg nzt Terminalzeichen um semantische Embeddings der
       zugrundeliegenden    uerungen
    """

    def __init__(self, model_name='paraphrase-multilingual-
       MiniLM-L12-v2'):
        self.model = SentenceTransformer(model_name)
        self.symbol_to_embedding = {}
        self.symbol_to_text = self._create_symbol_mapping()

    def _create_symbol_mapping(self):
        """
        Erstellt eine Mapping von Terminalzeichen zu
           Beispieltexten
        """
        return {
            'KBG': ['Guten Tag', 'Guten Morgen', 'Hallo'],
            'VBG': ['Guten Tag', 'Guten Morgen', 'Hallo
               zur ck'],
            'KBBd': ['Einmal Leberwurst', 'Ich h tte gerne
               K se', 'Ein Kilo  pfel  bitte'],
```

```
            'VBBd': ['Wie viel darf es sein?', 'Welche Sorte?
                ', 'Sonst noch etwas?'],
            'KBA': ['Zweihundert Gramm', 'Die hellen bitte',
                'Ja, gerne'],
            'VBA': ['In Ordnung', 'Kommt sofort', 'Alles klar
                '],
            'KAE': ['Kann ich das in Salat tun?', 'Woher
                kommen die?', 'Ist das frisch?'],
            'VAE': ['Eher anbraten', 'Aus der Region', 'Ja,
                ganz frisch'],
            'KAA': ['Bitte', 'Danke', 'Ja, danke'],
            'VAA': ['Das macht 8 Mark 20', '3 Mark bitte', '
                14 Mark 19'],
            'KAV': ['Auf Wiedersehen', 'Tsch ss', 'Einen
                sch nen Tag'],
            'VAV': ['Vielen Dank', 'Sch nen Tag noch', 'Auf
                Wiedersehen']
        }

    def get_embedding(self, symbol):
        """
        Gibt das Embedding f r ein Terminalzeichen zur ck
        """
        if symbol in self.symbol_to_embedding:
            return self.symbol_to_embedding[symbol]

        # Durchschnitt der Beispieltext-Embeddings
        texts = self.symbol_to_text.get(symbol, [symbol])
        embeddings = self.model.encode(texts)
        avg_embedding = np.mean(embeddings, axis=0)

        self.symbol_to_embedding[symbol] = avg_embedding
        return avg_embedding

    def enrich_sequence(self, sequence):
        """
        Erweitert eine Sequenz von Terminalzeichen um
            Embeddings
        """
        symbols = sequence
```

```
58          embeddings = np.array([self.get_embedding(sym) for
                sym in symbols])
59
60          return {
61              'symbols': symbols,
62              'embeddings': embeddings,
63              'combined': np.column_stack([
64                  self._one_hot_encode(symbols),
65                  embeddings
66              ]) if len(symbols) > 0 else np.array([])
67          }
68
69      def _one_hot_encode(self, symbols):
70          """
71          One-Hot-Encoding der Terminalzeichen
72          """
73          unique_symbols = sorted(set(self.symbol_to_text.keys
                ()))
74          symbol_to_idx = {sym: i for i, sym in enumerate(
                unique_symbols)}
75
76          one_hot = np.zeros((len(symbols), len(unique_symbols)
                ))
77          for i, sym in enumerate(symbols):
78              if sym in symbol_to_idx:
79                  one_hot[i, symbol_to_idx[sym]] = 1
80
81          return one_hot
82
83  def demonstrate_embedding_enrichment():
84      """
85      Demonstriert die Anreicherung von Terminalzeichen mit
            Embeddings
86      """
87      enricher = TerminalEmbeddingEnricher()
88
89      print("\n=== Anreicherung der Terminalzeichen mit
            Embeddings ===")
90
91      # Beispielsequenz
```

```
    sequence = ['KBG', 'VBG', 'KBBd', 'VBBd', 'KBA']

    enriched = enricher.enrich_sequence(sequence)

    print(f"\nSequenz: {'     '.join(sequence)}")
    print(f"Embedding-Dimension: {enriched['embeddings'].
       shape[1]}")
    print(f"One-Hot-Dimension: {enriched['combined'].shape[1]
        - enriched['embeddings'].shape[1]}")
    print(f"Kombinierte Dimension: {enriched['combined'].
       shape[1]}")

    return enricher
```

Listing 16: Transformer-Embeddings für Terminalzeichen

4.3 Graph Neural Networks für die Nonterminal-Hierarchie

Die Nonterminal-Hierarchie wird als Graph Neural Network modelliert:

```
"""
Graph Neural Network f r die Nonterminal-Hierarchie
Didaktische Implementierung mit PyTorch Geometric
"""

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, GATConv
from torch_geometric.data import Data
import networkx as nx

class GrammarGNN(nn.Module):
    """
    Graph Neural Network f r die Grammatik-Hierarchie
    """

    def __init__(self, input_dim, hidden_dim=64, num_classes
       =12):
        super().__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
```

```
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.classifier = nn.Linear(hidden_dim, num_classes)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)
        x = F.relu(x)
        x = self.classifier(x)
        return F.log_softmax(x, dim=1)

class GrammarHierarchyGNN:
    """
    Verwaltet das GNN f r die Nonterminal-Hierarchie
    """

    def __init__(self, grammar_rules):
        self.grammar = grammar_rules
        self.graph = self._build_graph()
        self.model = None

    def _build_graph(self):
        """
        Baut einen Graphen aus der Grammatik-Hierarchie
        """
        G = nx.DiGraph()

        # Knoten: Terminale und Nonterminale
        all_symbols = set()

        # Nonterminale als Knoten
        for nt, productions in self.grammar.items():
            all_symbols.add(nt)
            for prod, _ in productions:
                for sym in prod:
                    all_symbols.add(sym)

        # Kanten: Ableitungsrelationen
        for nt, productions in self.grammar.items():
```

```
            for prod, prob in productions:
                for sym in prod:
                    G.add_edge(nt, sym, weight=prob)

        return G

    def prepare_data(self):
        """
        Bereitet die Daten f r das GNN vor
        """
        # Knoten-Indizes
        nodes = list(self.graph.nodes())
        node_to_idx = {node: i for i, node in enumerate(nodes
            )}

        # Feature-Matrix (vereinfacht: One-Hot)
        x = torch.eye(len(nodes))

        # Kanten-Index
        edge_index = []
        for u, v, data in self.graph.edges(data=True):
            edge_index.append([node_to_idx[u], node_to_idx[v
                ]])

        edge_index = torch.tensor(edge_index, dtype=torch.
            long).t().contiguous()

        return Data(x=x, edge_index=edge_index)

    def train(self, epochs=100):
        """
        Trainiert das GNN
        """
        data = self.prepare_data()
        self.model = GrammarGNN(input_dim=data.x.shape[1])

        optimizer = torch.optim.Adam(self.model.parameters(),
            lr=0.01)

        print("\n=== Training des Grammar GNN ===")
```

```
        for epoch in range(epochs):
            self.model.train()
            optimizer.zero_grad()
            out = self.model(data.x, data.edge_index)

            # Selbst berwachtes Lernen: Rekonstruktion des Graphen
            # Vereinfacht: Vorhersage der Nachbarn
            loss = F.nll_loss(out[data.edge_index[0]], data.edge_index[1])

            loss.backward()
            optimizer.step()

            if epoch % 20 == 0:
                print(f"Epoch {epoch}: Loss = {loss.item():.4f}")

        return self.model

def demonstrate_gnn(grammar_rules):
    """
    Demonstriert das GNN f r die Grammatik-Hierarchie
    """
    print("\n=== Graph Neural Network f r Nonterminal-Hierarchie ===")

    gnn = GrammarHierarchyGNN(grammar_rules)
    print(f"Graph: {gnn.graph.number_of_nodes()} Knoten, "
          f"{gnn.graph.number_of_edges()} Kanten")

    model = gnn.train(epochs=100)

    return gnn, model
```

Listing 17: Graph Neural Network für Nonterminal-Hierarchie

4.4 Attention-Mechanismen für relevante Vorgänger

Attention-Mechanismen identifizieren besonders relevante Vorgänger für aktuelle Entscheidungen:

```
"""
Attention-Mechanismen f r die Identifikation relevanter
   Vorg nger
"""

import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

class SequenceAttention(nn.Module):
    """
    Attention-Mechanismus f r Sequenzmodellierung
    """

    def __init__(self, embedding_dim, hidden_dim=64):
        super().__init__()
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim

        # Attention-Parameter
        self.W_q = nn.Linear(embedding_dim, hidden_dim, bias=
           False)
        self.W_k = nn.Linear(embedding_dim, hidden_dim, bias=
           False)
        self.W_v = nn.Linear(embedding_dim, hidden_dim, bias=
           False)
        self.scale = hidden_dim ** 0.5

    def forward(self, x, mask=None):
        """
        x: (seq_len, batch, embedding_dim)
        """
        # Query, Key, Value berechnen
        Q = self.W_q(x)  # (seq_len, batch, hidden_dim)
        K = self.W_k(x)  # (seq_len, batch, hidden_dim)
```

```
33          V = self.W_v(x)  # (seq_len, batch, hidden_dim)
34
35          # Attention-Scores
36          scores = torch.matmul(Q.transpose(0, 1), K.transpose
                (0, 1).transpose(1, 2))
37          scores = scores / self.scale
38
39          if mask is not None:
40              scores = scores.masked_fill(mask == 0, -1e9)
41
42          # Attention-Gewichte
43          attention_weights = F.softmax(scores, dim=-1)
44
45          # Gewichtete Summe
46          context = torch.matmul(attention_weights, V.transpose
                (0, 1))
47
48          return context, attention_weights
49
50  class SymbolPredictorWithAttention(nn.Module):
51      """
52      Sagt das n chste Symbol mit Attention auf die Vorg nger
            vorher
53      """
54
55      def __init__(self, num_symbols, embedding_dim=50,
            hidden_dim=64):
56          super().__init__()
57          self.embedding = nn.Embedding(num_symbols,
                embedding_dim)
58          self.attention = SequenceAttention(embedding_dim,
                hidden_dim)
59          self.lstm = nn.LSTM(embedding_dim, hidden_dim,
                batch_first=True)
60          self.classifier = nn.Linear(hidden_dim +
                embedding_dim, num_symbols)
61
62      def forward(self, x):
63          """
64          x: (batch, seq_len) mit Symbol-Indizes
```

```
        """
        # Embeddings
        embedded = self.embedding(x)  # (batch, seq_len,
            embedding_dim)

        # LSTM f r sequenzielle Abh ngigkeiten
        lstm_out, (hidden, cell) = self.lstm(embedded)

        # Attention  ber  die Sequenz
        # Transponieren f r Attention (seq_len, batch,
            embedding_dim)
        context, attention_weights = self.attention(embedded.
            transpose(0, 1))

        # Kombiniere letzten LSTM-State mit Attention-Context
        last_hidden = hidden[-1]  # (batch, hidden_dim)
        last_context = context[-1]  # (batch, hidden_dim)

        # Vorhersage
        combined = torch.cat([last_hidden, last_context], dim
            =-1)
        logits = self.classifier(combined)

        return logits, attention_weights

def demonstrate_attention(terminal_chains, symbol_to_idx):
    """
    Demonstriert Attention-Mechanismen auf den Sequenzen
    """
    print("\n=== Attention-Mechanismen f r relevante
        Vorg nger ===")

    # Daten vorbereiten
    sequences = []
    for chain in terminal_chains:
        seq = [symbol_to_idx[sym] for sym in chain]
        sequences.append(seq)

    # Padding f r Batch-Verarbeitung
    from torch.nn.utils.rnn import pad_sequence
```

```
100     sequences_padded = pad_sequence([torch.tensor(seq) for
            seq in sequences],
101                                     batch_first=True,
                                            padding_value=0)
102
103     # Modell initialisieren
104     model = SymbolPredictorWithAttention(num_symbols=len(
            symbol_to_idx))
105
106     # Forward-Pass
107     logits, attention_weights = model(sequences_padded[:2])
            # Nur erste 2 Sequenzen
108
109     print(f"\nInput-Shape: {sequences_padded[:2].shape}")
110     print(f"Attention-Weights-Shape: {attention_weights.shape
            }")
111     print(f"Logits-Shape: {logits.shape}")
112
113     # Visualisierung der Attention-Gewichte
114     plot_attention_weights(attention_weights[0].detach().
            numpy(),
115                            sequences[0], sequences[0])
116
117     return model
118
119 def plot_attention_weights(attention, source_tokens,
        target_tokens):
120     """
121     Visualisiert Attention-Gewichte als Heatmap
122     """
123     import matplotlib.pyplot as plt
124     import seaborn as sns
125
126     plt.figure(figsize=(10, 8))
127     sns.heatmap(attention[:len(target_tokens), :len(
            source_tokens)],
128                 xticklabels=source_tokens,
129                 yticklabels=target_tokens,
130                 cmap='viridis', annot=True, fmt='.2f')
131     plt.title('Attention-Gewichte zwischen Vorg ngern und
```

```
        Vorhersage')
    plt.xlabel('Vorg nger-Symbole')
    plt.ylabel('Vorhersage-Position')
    plt.tight_layout()
    plt.savefig('attention_weights.png', dpi=150)
    plt.show()
```

Listing 18: Attention-Mechanismen für Sequenzmodellierung

4.5 Integration der Komponenten in Szenario D

Die vollständige Integration aller Komponenten in Szenario D:

```
"""
Szenario D: Hybride Modellierung
Integration computerlinguistischer Verfahren mit interpretativen Kategorien
"""

import json
import numpy as np

class ScenarioD:
    """
    Integriert computerlinguistische Verfahren komplement r zu den
    interpretativ gebildeten Kategorien der ARS 3.0
    """

    def __init__(self, terminal_chains, grammar_rules, reflection_log):
        self.terminal_chains = terminal_chains
        self.grammar_rules = grammar_rules
        self.reflection_log = reflection_log
        self.results = {}

        print("\n" + "="*70)
        print("SZENARIO D: HYBRIDE MODELLIERUNG")
        print("="*70)
        print("\nDieses Szenario integriert computerlinguistische")
```

```
        print("Verfahren KOMPLEMENT R zu den interpretativen ")
        print("Kategorien der ARS 3.0. Die interpretative Basis")
        print("bleibt erhalten, wird aber durch neue Verfahren")
        print("angereichert.\n")

    def run_crf_modeling(self):
        """
        F hrt CRF-Modellierung auf den Terminalzeichen durch
        """
        print("\n--- CRF-Modellierung ---")
        crf_model = demonstrate_crf(self.terminal_chains)
        self.results['crf'] = {'model': crf_model}
        return crf_model

    def run_embedding_enrichment(self):
        """
        Reichert Terminalzeichen mit Transformer-Embeddings an
        """
        print("\n--- Embedding-Anreicherung ---")
        enricher = demonstrate_embedding_enrichment()

        # Beispiel f r angereicherte Sequenz
        example_seq = self.terminal_chains[0][:5]
        enriched = enricher.enrich_sequence(example_seq)

        self.results['embeddings'] = {
            'enricher': enricher,
            'example': enriched
        }

        return enricher

    def run_gnn_hierarchy(self):
        """
        Modelliert die Nonterminal-Hierarchie als GNN
        """
```

```
61          print("\n--- GNN f r Nonterminal-Hierarchie ---")
62          gnn, model = demonstrate_gnn(self.grammar_rules)
63          self.results['gnn'] = {'gnn': gnn, 'model': model}
64          return gnn, model
65
66      def run_attention_analysis(self):
67          """
68          Analysiert Attention-Mechanismen auf den Sequenzen
69          """
70          print("\n--- Attention-Analyse ---")
71
72          # Symbol zu Index Mapping
73          all_symbols = set()
74          for chain in self.terminal_chains:
75              all_symbols.update(chain)
76          symbol_to_idx = {sym: i for i, sym in enumerate(
              sorted(all_symbols))}
77
78          model = demonstrate_attention(self.terminal_chains,
              symbol_to_idx)
79          self.results['attention'] = {'model': model}
80
81          return model
82
83      def run_all(self):
84          """
85          F hrt alle Analysen aus (komplement r, nicht
              ersetzend)
86          """
87          self.run_crf_modeling()
88          self.run_embedding_enrichment()
89          self.run_gnn_hierarchy()
90          self.run_attention_analysis()
91
92          # Zusammenfassung
93          print("\n" + "="*70)
94          print("ZUSAMMENFASSUNG SZENARIO D")
95          print("="*70)
96          print("    CRF-Modellierung: Sequenzielle
              Abh ngigkeiten modelliert")
```

```
97          print("    Embedding-Anreicherung: Terminalzeichen
               semantisch angereichert")
98          print("    GNN-Hierarchie: Nonterminal-Struktur als
               Graph modelliert")
99          print("    Attention-Analyse: Relevante Vorg nger
               identifiziert")
100         print("\nDie interpretativen Kategorien der ARS 3.0
               bleiben")
101         print("die Grundlage aller Analysen. Die
               computerlinguistischen")
102         print("Verfahren dienen der komplement ren
               Erkenntnisgewinnung.")
103
104         return self.results
105
106 def run_scenario_d_demonstration(terminal_chains,
    grammar_rules, reflection_log):
107     """
108     F hrt die vollst ndige Demonstration von Szenario D aus
109     """
110     scenario = ScenarioD(terminal_chains, grammar_rules,
           reflection_log)
111     results = scenario.run_all()
112
113     # Speichere Ergebnisse
114     with open('scenario_d_results.json', 'w') as f:
115         # Vereinfachte serialisierbare Version
116         serializable = {
117             'crf': {'status': 'completed'},
118             'embeddings': {'status': 'completed'},
119             'gnn': {'num_nodes': results['gnn'][0].graph.
                   number_of_nodes()},
120             'attention': {'status': 'completed'}
121         }
122         json.dump(serializable, f, indent=2)
123
124     print("\nErgebnisse gespeichert in 'scenario_d_results.
           json'")
125
126     return results
```

```
127
128 # Didaktischer Hinweis
129 print("\n" + "="*70)
130 print("METHODOLOGISCHER HINWEIS ZU SZENARIO D")
131 print("="*70)
132 print("Szenario D beh lt die interpretative Basis der ARS
    3.0 bei.")
133 print("Die computerlinguistischen Verfahren werden
    KOMPLEMENT R")
134 print("eingesetzt, nicht als Ersatz der manuellen
    Kategorienbildung.")
135 print("Dies entspricht der methodologischen Forderung nach")
136 print("Kontrolle und Transparenz im Sinne der XAI-Kriterien."
    )
```

Listing 19: Szenario D - Vollständige hybride Integration

5 Vergleich der Szenarien und methodologische Reflexion

5.1 Gegenüberstellung der Ansätze

Tabelle 2: Vergleich der Szenarien C und D

Kriterium	**Szenario C**	**Szenario D**
Kategorienbildung	Automatisch (Speech Act Recognition)	Interpretativ (ARS 3.0)
Datenbasis	Augmentierte Rohdaten	Terminalzeichenketten
Repräsentation	Vektorielle Embeddings	Diskrete Kategorien + Embeddings
Hierarchie	Automatisch gelernt	Explizit induziert (ARS 3.0)
Transparenz	Gering (Black Box)	Hoch (dokumentierte Entscheidungen)
Didaktischer Wert	Funktionsweise neuronaler Verfahren	Integration alter und neuer Methoden
Empirische Validität	Nicht gegeben	Eingeschränkt (basierend auf Interpretation)
Methodologische Kontrolle	Verloren	Erhalten

5.2 Didaktische Erkenntnisse aus Szenario C

Die Implementierung von Szenario C hat gezeigt:

1. **Notwendigkeit großer Datenmengen**: Neuronale Verfahren benötigen für valide Ergebnisse Datenmengen, die weit über die acht Transkripte hinausgehen. Die Augmentierung ermöglicht zwar die Demonstration der Funktionsweise, ersetzt aber keine echten Daten.

2. **Opazität der Entscheidungen**: Die automatisch gelernten Kategorien und Attention-Gewichte sind für Dritte nicht ohne weiteres nachvollziehbar. Die XAI-Kriterien der Verständlichkeit und Transparenz werden verletzt.

3. **Verlust der interpretativen Basis**: Die automatische Speech Act Recognition bildet nicht die qualitativ gehaltvollen Unterscheidungen der ARS ab (z.B. zwischen KBA und KAA), sondern lernt statistische Korrelationen im Vektorraum.

5.3 Didaktische Erkenntnisse aus Szenario D

Die Implementierung von Szenario D hat gezeigt:

1. **Komplementarität statt Substitution**: Die computerlinguistischen Verfahren können wertvolle Zusatzinformationen liefern (z.B. semantische Ähnlichkeiten zwischen verschiedenen Äußerungen), ohne die interpretative Basis zu ersetzen.

2. **Validierungsmöglichkeiten**: Die Embedding-Ähnlichkeiten können zur Validierung der interpretativen Kategorienbildung genutzt werden: Ähnliche Äußerungen sollten ähnliche Terminalzeichen erhalten.

3. **Visualisierung von Abhängigkeiten**: Attention-Mechanismen und CRF-Modelle visualisieren, welche Vorgänger für aktuelle Entscheidungen besonders relevant sind – dies kann die sequenzielle Struktur der Gespräche veranschaulichen.

4. **Methodologische Kontrolle bleibt erhalten**: Da die interpretativen Kategorien die Grundlage bilden, bleiben alle Ergebnisse an die qualitativen Entscheidungen rückgebunden und intersubjektiv prüfbar.

5.4 Fazit für die Lehrpraxis

Die didaktische Exploration der Szenarien C und D führt zu folgenden Schlussfolgerungen:

1. **Szenario C eignet sich zur Demonstration der Funktionsweise** neuronaler Verfahren, sollte aber mit explizitem Hinweis auf die fehlende empirische Validität und die methodologischen Probleme eingesetzt werden.

2. **Szenario D ist methodologisch vorzuziehen**, da es die interpretative Basis erhält und computerlinguistische Verfahren komplementär nutzt. Es vermittelt, wie alte und neue Methoden produktiv verbunden werden können.

3. **Data Augmentation ist ein wertvolles didaktisches Werkzeug**, um mit kleinen Datensätzen die Funktionsweise von Verfahren zu demonstrieren. Der augmentierte Charakter der Daten muss dabei stets transparent gemacht werden.

4. **Die XAI-Kriterien** (Verständlichkeit, Genauigkeit, Wissensgrenzen) bieten einen geeigneten Rahmen, um die verschiedenen Ansätze zu bewerten und ihre Stärken und Schwächen zu reflektieren.

6 Ausblick

Die hier vorgestellten didaktischen Implementierungen können in mehreren Richtungen weiterentwickelt werden:

1. **Erweiterung der Augmentierungsstrategien**: Über einfaches Kopieren hinaus könnten komplexere Augmentierungen (Paraphrasierung, kontrollierte Variation) implementiert werden.

2. **Integration weiterer Verfahren**: z.B. PETRI-Netze für Nebenläufigkeit, Bayessche Netze für Inferenz, oder formale Verifikationsmethoden.

3. **Entwicklung von Vergleichsmetriken**: Wie kann man die Ergebnisse der verschiedenen Szenarien quantitativ vergleichen, ohne die qualitative Basis zu verlieren?

4. **Übertragung auf andere Datensätze**: Die Methodik lässt sich auf andere Interaktionstypen übertragen (Arzt-Patient-Gespräche, Unterrichtsinteraktionen, etc.).

Entscheidend bleibt dabei stets die methodologische Kontrolle: Die formalen Verfahren müssen den interpretativen Charakter der Analyse respektieren und dürfen nicht zu dessen Automatisierung führen.

Literatur

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82-115.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of NAACL-HLT 2019*, 4171-4186.

Flick, U. (2019). *Qualitative Sozialforschung: Eine Einführung* (9. Aufl.). Rowohlt.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of ICML 2001*, 282-289.

Mann, W. C., & Thompson, S. A. (1988). Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, 8(3), 243-281.

Przyborski, A., & Wohlrab-Sahr, M. (2021). *Qualitative Sozialforschung: Ein Arbeitsbuch* (5. Aufl.). De Gruyter Oldenbourg.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of EMNLP-IJCNLP 2019*, 3982-3992.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems 30*, 5998-6008.