Grammatikinduktion, Transduktion und Parsing

Die ARS als methodologischer Vorläufer erklärbarer neuro-symbolischer KI

Paul Koop

1994–2026

Zusammenfassung

Dieser Beitrag untersucht das historische und methodologische Verhältnis zwischen der Algorithmisch Rekursiven Sequenzanalyse (ARS) und der zeitgenössischen neuro-symbolischen KI. Auf der Grundlage von drei frühen ARS-Implementierungen – einem Induktor in Scheme, einem Parser in Pascal und einem Transduktor in Lisp (1994) – sowie einer Simulation eines großen Sprachmodells in Python (2023) argumentiere ich, dass die ARS eine *proto-neuro-symbolische* Methodologie darstellt. Im Gegensatz zu rein statistischen Sprachmodellen produziert die ARS explizite, falsifizierbare und intersubjektiv prüfbare Grammatiken. Der Beitrag zeigt, dass die zentralen Herausforderungen der heutigen neuro-symbolischen KI – die Integration von Mustererkennung und regelbasiertem Schließen, die Sicherung von Erklärbarkeit und die Wahrung methodologischer Kontrolle – in der ARS bereits vor Jahrzehnten adressiert wurden. Ich ordne die ARS in Henry Kautz' Taxonomie neuro-symbolischer Architekturen ein, evaluiere sie anhand der XAI-Kriterien (Verständlichkeit, Genauigkeit, Wissensgrenzen) und kontrastiere sie mit großen Sprachmodellen, die simulieren, ohne zu erklären. Der Beitrag schließt mit methodologischen Lehren für die zeitgenössische neuro-symbolische Forschung.

Inhaltsverzeichnis

1 Einleitung: Das verborgene Erbe der ARS

Der gegenwärtige Diskurs über neuro-symbolische KI ist von einer merkwürdigen Amnesie geprägt. Während Forscher über Architekturen debattieren, die neuronale Netze mit symbolischem Schließen integrieren (Hitzler & Sarker, 2022; Garcez & Lamb, 2020), ist ein methodologisch anspruchsvoller Vorläufer weitgehend in Vergessenheit geraten: die **Algorithmisch Rekursive Sequenzanalyse (ARS)**.

Die ARS, ursprünglich 1994 entwickelt und bis 2026 kontinuierlich verfeinert, stellt einen der frühesten systematischen Versuche dar, qualitative Hermeneutik mit formaler Grammatikinduktion zu verbinden. Im Gegensatz zu zeitgenössischen großen Sprachmodellen (LLMs), die statistische Muster aus massiven Korpora lernen, aber opak bleiben, produziert die ARS **explizite, falsifizierbare und intersubjektiv prüfbare Grammatiken**. Anders als rein symbolische Ansätze, die am Wissenserwerbsproblem leiden, induziert die ARS Regeln aus empirischen Protokollen.

Dieser Beitrag leistet drei Dinge:

1. Er rekonstruiert drei frühe ARS-Implementierungen – einen **Induktor** in Scheme, einen **Parser** in Pascal und einen **Transduktor** in Lisp – und zeigt, wie jede einen anderen Aspekt der Sequenzanalyse adressiert.

2. Er interpretiert diese Implementierungen als **proto-neuro-symbolische** Systeme und ordnet sie in Henry Kautz' Taxonomie neuro-symbolischer Architekturen ein (Kautz, 2020).

3. Er kontrastiert die ARS mit einem großen Sprachmodell, das auf demselben Korpus trainiert wurde, und zeigt, dass LLMs simulieren, aber nicht *erklären* – eine Unterscheidung, die für die XAI-Kriterien (Explainable AI) zentral ist (Ortigossa et al., 2024).

Der Beitrag behauptet nicht, dass die ARS ein neuro-symbolisches System im zeitgenössischen Sinne sei – ihr fehlen neuronale Komponenten. Vielmehr argumentiere ich, dass die ARS die *methodologische Logik* neuro-symbolischer Integration verkörpert: die Kombination von musterbasierter Induktion (System 1) mit regelbasierter Explikation (System 2) unter Wahrung der Erklärbarkeit durch Design.

2 Drei Implementierungen, ein Korpus

2.1 Die empirische Grundlage: Ein Marktgespräch

Alle in diesem Beitrag analysierten Implementierungen basieren auf demselben empirischen Korpus: einem transkribierten Verkaufsgespräch, das am 28. Juni 1994 auf dem Aachener Marktplatz aufgenommen wurde. Das Transkript wurde einer qualitativen Sequenzanalyse nach der Methode der objektiven Hermeneutik (Oevermann et al., 1979) unterzogen, was zu einer Terminalzeichenkette mit 12 Kategorien führte (KBG, VBG, KBBd, VBBd, KBA, VBA, KAE, VAE, KAA, VAA, KAV, VAV).

Die durchgängig verwendete Terminalzeichenkette lautet:

```
KBG VBG KBBd VBBd KBA VBA KBBd VBBd KBA VBA KAE VAE KAE VAE KAA VAA KAV VAV
```

2.2 Induktor (Scheme, 1994): Vom Korpus zur Grammatik

Der Induktor, geschrieben in Scheme, ist die grundlegende Komponente der ARS. Seine Funktion ist es, ein Korpus von Terminalzeichen zu lesen und eine probabilistische kontextfreie Grammatik (PCFG) durch Zählung von Transitionen zu induzieren.

2.2.1 Zentrale Datenstrukturen

```
;; Lexikon: 12 Terminalzeichen
(define lexikon (vector 'KBG 'VBG 'KBBd 'VBBd 'KBA 'VBA
                        'KAE 'VAE 'KAA 'VAA 'KAV 'VAV))

;; Transformationsmatrix zur Z hlung von    bergngen
(define matrix (vector zeile0 zeile1 ... zeile17))

;; Funktion zum Z hlen der Transitionen
(define (transformationenZaehlen korpus)
  (vector-set! (vector-ref matrix (izeichen (car korpus)))
               (izeichen (car(cdr korpus)))
               (+ 1 (vector-ref (vector-ref matrix (izeichen
                  (car korpus)))
                                (izeichen (car(cdr korpus))))
                                  ))
  (if(not(null? (cdr (cdr korpus))))
     (transformationenZaehlen (cdr korpus))))
```

Listing 1: Lexikon und Transformationsmatrix in Scheme

2.2.2 Induzierte Grammatik

Die resultierende Grammatik lautet:

```
(KBG -> . VBG)
(VBG -> . KBBd)
(KBBd -> . VBBd)
(VBBd -> . KBA)
(KBA -> . VBA)
(VBA -> . KBBd) (VBA -> . KAE)
(KAE -> . VAE)
(VAE -> . KAE) (VAE -> . KAA)
(KAA -> . VAA)
(VAA -> . KAV)
(KAV -> . VAV)
```

2.2.3 Interpretation

Der Induktor transformiert das empirische Protokoll in ein **explizites Regelsystem**. Jede Produktionsregel ist mit ihrer empirischen Häufigkeit gewichtet. Diese Transformation ist reversibel: Aus der Grammatik können Sequenzen generiert werden, die die statistischen Eigenschaften des ursprünglichen Korpus reproduzieren.

In neuro-symbolischer Terminologie führt der Induktor eine **symbolische Abstraktion** aus diskreten Daten durch. Er lernt Gewichte nicht durch Backpropagation, sondern durch einfaches Zählen – ein transparenter, verifizierbarer Prozess.

2.3 Parser (Pascal, 1992): Validierung von Wohlgeformtheit

Der Parser, geschrieben in Pascal, implementiert einen Chart-Parser, der entscheidet, ob eine gegebene Terminalzeichenkette *wohlgeformt* im Sinne der induzierten Grammatik ist.

2.3.1 Wichtige Datentypen

```
1 TYPE
2   TKategorien = (Leer, VKG, BG, VT, AV, B, A, BBD, BA, AE, AA
      ,
3                  KBG, VBG, KBBD, VBBD, KBA, VBA, KAE, VAE,
4                  KAA, VAA, KAV, VAV);
5
```

```
  TKante = RECORD
    Kategorie : TKategorien;
    vor, nach, zeigt : PTKante;
    gefunden : PTKantenListe;
    aktiv : BOOLEAN;
    nummer : INTEGER;
    CASE Wort : BOOLEAN OF
      TRUE : (inhalt : STRING);
      FALSE : (gesucht : PTKategorienListe);
  END;
```

Listing 2: Parserdatenstrukturen in Pascal

2.3.2 Parsing-Algorithmus

Der Parser implementiert einen Standard-Chart-Parsing-Algorithmus mit drei Kernregeln:

1. **Initialisierung**: Terminalsymbole werden als aktive Kanten eingefügt.
2. **Prädiktion**: Neue Kanten werden für Nonterminale erzeugt, die an einer bestimmten Position beginnen können.
3. **Komplettierung**: Wenn ein Nonterminal vollständig erkannt ist, werden übergeordnete Regeln komplettiert.

2.3.3 Interpretation

Der Parser operationalisiert den Begriff der **strukturellen Wohlgeformtheit**. Eine Sequenz ist nicht nur "plausibel", sondern formal entscheidbar. Dies antizipiert den deterministischen endlichen Automaten (DFA), der später in `ARS_XAI_Aut_Ger.tex` formalisiert wurde.

In XAI-Terminologie verkörpert der Parser **Erklärbarkeit durch Design**: Jede Entscheidung, eine Sequenz zu akzeptieren oder zu verwerfen, kann auf explizite Regeln zurückgeführt werden.

2.4 Transduktor (Lisp, 1994): Generierung neuer Protokolle

Der Transduktor, geschrieben in Lisp, generiert aus der induzierten Grammatik neue Terminalzeichenketten und simuliert so mögliche Verkaufsgespräche.

2.4.1 Generierungsalgorithmus

```
;; Generiert die Sequenz
(defun gs (st r)
  (cond
    ((equal st nil) nil)
    ((atom st) (cons st (gs (next st r (random 101)) r)))
    (t (cons (eval st) (gs (next st r (random 101)) r)))
  )
)

;; W hlt das n chste Symbol basierend auf gewichteten
   Wahrscheinlichkeiten
(defun next (st r z)
  (cond
    ((equal r nil) nil)
    ((and (<= z (car (cdr (car r))))
          (equal st (car (car r))))
     (car (reverse (car r))))
    (t (next st (cdr r) z))
  )
)
```

Listing 3: Transduktor in Lisp

2.4.2 Beispielausgabe

Eine typische generierte Sequenz (Klammern zur Lesbarkeit entfernt):

```
KBG VBG KBBD VBBD KBA VBA KAE VAE KAA VAA
KBBD VBBD KBA VBA KBBD VBBD KBA VBA KBBD VBBD KBA VBA KAE VAE KAA VAA
KAV VAV
```

2.4.3 Interpretation

Der Transduktor ist ein **generatives Modell** – aber anders als ein LLM ist sein Generierungsprozess vollständig transparent. Jedes Symbol wird durch eine Regel produziert, die inspiziert, zurückverfolgt und gerechtfertigt werden kann. Der Transduktor halluziniert nicht; er folgt der Grammatik.

2.5 Das große Sprachmodell (Python, 2023): Simulation ohne Erklärung

Zum Vergleich wurde ein tiefes Sprachmodell (LSTM-basiert) auf demselben Korpus trainiert. Die Modellarchitektur folgt der in Trask (2020) beschriebenen Implementierung.

2.5.1 Modellarchitektur

```
class LSTMCell(Layer):
    def __init__(self, n_inputs, n_hidden, n_output):
        self.xf = Linear(n_inputs, n_hidden)
        self.xi = Linear(n_inputs, n_hidden)
        self.xo = Linear(n_inputs, n_hidden)
        self.xc = Linear(n_inputs, n_hidden)
        self.hf = Linear(n_hidden, n_hidden, bias=False)
        self.hi = Linear(n_hidden, n_hidden, bias=False)
        self.ho = Linear(n_hidden, n_hidden, bias=False)
        self.hc = Linear(n_hidden, n_hidden, bias=False)
        self.w_ho = Linear(n_hidden, n_output, bias=False)
```

Listing 4: LSTM-Sprachmodell in Python

2.5.2 Beispielausgabe

```
KBG VBG
KBBD VBBD KBA VBA KAE VAE KAA VAA
KBBD VBBD KBA VBA KBBD VBBD KBA VBA KBBD VBBD KBA VBA KAE VAE
KAA VAA
KAV VAV
KBG VBG
KBBD VBBD KBA VBA KAE VAE KAE VAE KAE VAE KAE VAE KAA VAA
```

2.5.3 Interpretation

Die LLM-Ausgabe ist **oberflächlich nicht unterscheidbar** von der Ausgabe des Transduktors. Beide generieren plausible Sequenzen von Terminalzeichen. Die Ähnlichkeit trügt jedoch:

- Die **Ausgabe des Transduktors** wird durch explizite, inspizierbare Regeln

erzeugt. Die Produktion jedes Symbols kann auf eine Grammatikregel zurückgeführt werden.

- Die **Ausgabe des LLM** wird durch interne Gewichte erzeugt, die nicht direkt interpretierbar sind. Man kann nicht erklären, *warum* ein bestimmtes Symbol gewählt wurde.

Wie im Original-Notebook vermerkt:

> Im Gegensatz zu kognitivistischen Modellen (ARS, Grammar Induction, Parser, Grammar Transduction) erklärt ein solches Großes Sprachmodell nichts und deshalb werden Große Sprachmodelle von Postmodernismus, Posthumanismus und Transhumanismus mit parasitärer Intention gefeiert.

3 ARS als proto-neuro-symbolische KI

3.1 Das neuro-symbolische Forschungsprogramm

Neuro-symbolische KI integriert neuronale Methoden (Mustererkennung, Lernen aus Daten) mit symbolischen Methoden (Logik, Regeln, Schließen). Henry Kautz' Taxonomie (Kautz, 2020) unterscheidet mehrere Architekturmuster:

Tabelle 1: Kautz' neuro-symbolische Architekturen

Architektur	**Beschreibung**
Neural | Symbolic	Neuronale Wahrnehmung, symbolisches Schließen
Neural: Symbolic → Neural	Symbolische Generierung von Trainingsdaten
NeuralSymbolic	Aus symbolischen Regeln generierte neuronale Netze
Neural[Symbolic]	In neuronale Netze eingebettetes symbolisches Schließen

3.2 Verortung der ARS in der Taxonomie

Die ARS passt nicht genau in eine einzelne Kategorie, da sie unabhängig vom neuronalen Paradigma entwickelt wurde. Wenn wir jedoch den qualitativen Interpretationsprozess als eine Form der **Mustererkennung** (System 1) und die

Grammatikinduktion als **symbolisches Schließen** (System 2) auffassen, nähert sich die ARS dem Muster **Neural | Symbolic** an:

- **Mustererkennung** (System 1): Der menschliche Interpret identifiziert wiederkehrende Muster im Transkript, produziert Lesarten und falsifiziert Alternativen – eine Form der musterbasierte Kognition.
- **Symbolisches Schließen** (System 2): Die induzierte Grammatik, der Parser und der Transduktor bilden ein formales symbolisches System, das ausgeführt, inspiziert und validiert werden kann.

Was die ARS von zeitgenössischen neuro-symbolischen Systemen unterscheidet, ist dass die Mustererkennungskomponente **menschlich**, nicht neuronal ist. Dies ist keine Schwäche, sondern eine bewusste methodologische Entscheidung: Sie stellt sicher, dass die Mustererkennung interpretierbar bleibt und der intersubjektiven Validierung unterliegt.

3.3 Die drei Komponenten als komplementäre neuro-symbolische Funktionen

Tabelle 2: ARS-Komponenten und ihre neuro-symbolischen Funktionen

Komponente	**Sprache**	**Neuro-symbolische Funktion**
Induktor	Scheme	Symbolabstraktion aus diskreten Daten
Parser	Pascal	Strukturelle Validierung, Wohlgeformtheitsprüfung
Transduktor	Lisp	Generative Regelanwendung
LLM (Kontrast)	Python	Reine Mustererkennung ohne Erklärung

Zusammen bilden diese drei Komponenten eine **vollständige Pipeline** von empirischen Daten zum generativen Modell – eine Pipeline, die in jedem Schritt vollständig transparent ist.

4 XAI-Validierung der ARS

Die drei NIST-XAI-Kriterien (Ortigossa et al., 2024) bieten einen Rahmen für die Bewertung von Erklärbarkeit:

4.1 Verständlichkeit

- **Induktor**: Die Transformationsmatrix und die Produktionsregeln sind direkt interpretierbar. Jede Regel entspricht einem beobachteten Übergang im Korpus.
- **Parser**: Zustände (KBG, VBG, VKG usw.) sind semantisch gehaltvolle Kategorien, die aus qualitativer Interpretation abgeleitet sind.
- **Transduktor**: Die Generierung folgt expliziten Regeln, die inspiziert werden können.
- **LLM**: Gewichte und verborgene Zustände sind nicht direkt interpretierbar.

4.2 Genauigkeit

- **Induktor**: Die induzierte Grammatik reproduziert die empirischen Übergangshäufigkeiten mit hoher Korrelation ($r = 0{,}9999$).
- **Parser**: Wohlgeformtheitsentscheidungen sind deterministisch und verifizierbar.
- **Transduktor**: Generierte Sequenzen folgen der statistischen Verteilung des Korpus.
- **LLM**: Der Trainingsverlust sinkt, aber das Modell produziert keine expliziten Regeln, die gegen die Daten validiert werden können.

4.3 Wissensgrenzen

- **ARS**: Die Grammatik dokumentiert explizit ihre Datenbasis (8 Transkripte, 59 Interakte). Sie erhebt keinen Anspruch auf Generalisierung über das Korpus hinaus.
- **LLM**: Die Grenzen des Modells sind nicht explizit repräsentiert. Es kann halluzinieren oder plausible, aber ungültige Sequenzen produzieren, ohne Unsicherheit zu signalisieren.

5 Simulation vs. Erklärung: Die fundamentale Unterscheidung

5.1 Was LLMs tun: Statistische Simulation

Große Sprachmodelle lernen die statistische Verteilung von Token-Sequenzen aus Trainingsdaten. Bei der Generierung sampeln sie aus dieser gelernten Verteilung. Dies ist **Simulation**: Das Modell produziert Ausgaben, die der Trainingsverteilung ähneln.

Entscheidend ist, dass Simulation kein Verständnis der *Regeln* erfordert, die die Daten erzeugen. Ein LLM, das auf einem Korpus von Verkaufsgesprächen trainiert wurde, kann plausible neue Gespräche generieren, ohne jemals Konzepte wie "Begrüßung", "Bedarfsklärungöder "Verabschiedungßu repräsentieren.

5.2 Was die ARS tut: Explikative Rekonstruktion

Die ARS zielt dagegen auf **explikative Rekonstruktion**. Sie induziert explizite Regeln, die die beobachteten Regularitäten *konstituieren*. Diese Regeln sind nicht nur statistische Zusammenfassungen, sondern **generative Mechanismen**, die:

1. **inspiziert** werden können (die Regeln sind in einer formalen Sprache geschrieben – Scheme, Pascal, Lisp)
2. **zurückverfolgt** werden können (jeder Generierungsschritt kann auf eine Regel zurückgeführt werden)
3. **falsifiziert** werden können (ein Gegenbeispiel kann eine Regel widerlegen)
4. **kommuniziert** werden können (die Regeln können mit anderen Forschern geteilt, diskutiert und kritisiert werden)

5.3 Die Cargo-Kult-Kritik

Das Original-Notebook enthält einen provokativen Passus:

„Wenn man ein Lehrbuch über die Regeln von Verkaufsgesprächen schreiben will, aber einen Softwareagenten erhält, der gerne Verkaufsgespräche führt, hat man auf sehr hohem Niveau schlechte Arbeit gemacht.“

Diese Kritik ist nicht anti-KI. Sie warnt vor **Kategorienfehlern**: Ein Werkzeug, das für einen Zweck entwickelt wurde (statistische Simulation), für ein anderes

Problem zu verwenden (explikative Rekonstruktion). Ein LLM ist ein ausgezeichneter Simulator, aber ein schlechter Erklärer. Die ARS ist ein ausgezeichneter Erklärer, aber ein weniger skalierbarer Simulator. Diese Komplementarität zu erkennen, ist der erste Schritt zu einer methodologisch soliden Integration.

6 Hin zu einer methodologischen Synthese

6.1 Komplementarität statt Konkurrenz

Die obige Analyse legt eine Arbeitsteilung nahe:

- **LLMs für Skalierung** nutzen: Neuronale Mustererkennung kann erste Kategoriezuordnungen vorschlagen, Kandidatenmuster identifizieren und große Korpora verarbeiten.
- **ARS für Validierung** nutzen: Die symbolische Grammatik kann die Wohlgeformtheit neuronaler Vorschläge prüfen, interpretative Entscheidungen dokumentieren und Erklärungen liefern.
- **Den Menschen in der Schleife behalten**: Die endgültige Validierungs- und Interpretationsautorität verbleibt beim menschlichen Forscher.

Dies ist genau der Ansatz, der später als **CGTI (Computational Grounded Theory Integration)** und **AQSA (Adversarial Qualitative Sequence Analysis)** formalisiert wurde.

6.2 Lehren für die zeitgenössische neuro-symbolische KI

Aus der ARS-Erfahrung kann die zeitgenössische neuro-symbolische Forschung lernen:

1. **Erklärbarkeit durch Design**: Symbolische Komponenten sollten von Grund auf interpretierbar sein, nicht als nachträgliche Ergänzungen.
2. **Mehrere Formalismen**: Unterschiedliche Aufgaben (Induktion, Parsing, Generierung) können unterschiedliche formale Sprachen erfordern. Scheme, Pascal und Lisp dienten jeweils einem eigenen Zweck.
3. **Methodologische Kontrolle vor Skalierung**: Ein kleines, gut verstandenes Korpus (8 Transkripte) bietet mehr methodologische Einsicht als ein großes, opakes Korpus.

4. **Der Mensch als System 1**: In manchen Kontexten ist menschliche Mustererkennung neuronalen Netzen überlegen – nicht weil sie schneller ist, sondern weil sie interpretierbar ist und kommuniziert werden kann.

7 Fazit

Dieser Beitrag hat drei frühe Implementierungen der Algorithmisch Rekursiven Sequenzanalyse (ARS) rekonstruiert – einen Induktor in Scheme, einen Parser in Pascal und einen Transduktor in Lisp – und sie mit einem großen Sprachmodell kontrastiert, das auf demselben Korpus trainiert wurde. Ich habe argumentiert, dass:

1. Die ARS eine **proto-neuro-symbolische** Methodologie darstellt, die zentrale Anliegen der zeitgenössischen neuro-symbolischen KI um Jahrzehnte antizipiert.

2. Die drei Komponenten (Induktor, Parser, Transduktor) komplementäre Funktionen adressieren: Symbolabstraktion, strukturelle Validierung und generative Regelanwendung.

3. Im Gegensatz zu LLMs, die statistische Verteilungen ohne Erklärung simulieren, produziert die ARS **explizite, falsifizierbare und intersubjektiv prüfbare Grammatiken**.

4. Die ARS die XAI-Kriterien der Verständlichkeit, Genauigkeit und Wissensgrenzen auf Weisen erfüllt, die rein neuronale Modelle nicht können.

Der historische Record zeigt, dass die Herausforderungen der neuro-symbolischen Integration lange vor der aktuellen Forschungswelle erkannt und bearbeitet wurden. Die ARS bietet eine methodologische Vorlage, die zeitgenössische Forscher gut daran täten zu studieren – nicht als historisches Artefakt, sondern als lebendigen Ansatz für **erklärbare, kontrollierte und verifizierbare** Sequenzanalyse.

Die Frage für die neuro-symbolische KI ist nicht, ob Mustererkennung mit regelbasiertem Schließen integriert werden soll. Die Frage ist, wie dies ohne Aufgabe der methodologischen Standards geschehen kann, die wissenschaftliche Erkenntnis erst möglich machen. Die ARS gibt eine Antwort.

Literatur

Garcez, A. d'Avila, & Lamb, L. C. (2020). Neurosymbolic AI: The 3rd wave. *arXiv preprint arXiv:2012.05876.*

Hitzler, P., & Sarker, M. K. (Hrsg.). (2022). *Neuro-Symbolic Artificial Intelligence: The State of the Art.* IOS Press.

Kautz, H. (2020). The third AI summer: AAAI Robert S. Engelmore Memorial Award Lecture. *AI Magazine*, 43(1), 93-104.

Koop, P. (1992). *Demo-Parser Chart-Parser Version 1.0.* Pascal-Quellcode.

Koop, P. (1994). *Grammatikinduktion empirisch gesicherter Verkaufsgespräche.* Scheme-Quellcode.

Koop, P. (1994). *Sequenzanalyse empirisch gesicherter Verkaufsgespräche.* Lisp-Quellcode.

Koop, P. (2023). *Qualitative Sozialforschung und Große Sprachmodelle.* Jupyter Notebook.

Oevermann, U., Allert, T., Konau, E., & Krambeck, J. (1979). Die Methodologie einer objektiven Hermeneutik. In H.-G. Soeffner (Hrsg.), *Interpretative Verfahren in den Sozial- und Textwissenschaften* (S. 352-434). Metzler.

Ortigossa, E. S., Gonçalves, T., & Nonato, L. G. (2024). Explainable Artificial Intelligence (XAI)—From Theory to Methods and Applications. *IEEE Access*, 12, 80799-80846.

Trask, A. W. (2020). *Neuronale Netze und Deep Learning kapieren: Der einfache Praxiseinstieg mit Beispielen in Python.* dpunkt.