Between Interpretation and Computation

Hierarchical Grammar Induction as Explication of Latent Sequence Structures in Sales Conversations

Paul Koop

June/July 1994 & 2024/2026

Abstract

Qualitative social research currently faces a methodological dilemma: On one hand, generative AI systems promise an unprecedented scaling of interpretive work steps; on the other hand, due to their stochastic nature, they evade the classical validation logic of qualitative research. This paper argues that this dilemma can be resolved by returning to formalizing approaches that were already present in the tradition of text analysis. As a concrete solution, the paper develops the **Algorithmic Recursive Sequence Analysis (ARS) in its version 3.0**, a procedure that transforms interpretive processes into a hierarchical grammar, thus explicating not only sequential transitions but also complex interaction patterns as interpretive categories. The connection to the current discussion on **Explainable AI (XAI)** proves to be doubly fruitful: it provides a conceptual framework to reflect on the quality of qualitative interpretations and reminds us that explainability is not a luxury but a necessity – in technology as well as in science. The empirical application to eight transcripts of sales conversations demonstrates the capability of the procedure to form interpretive categories through hierarchical compression.

Contents

1 Introduction: The Paradox of Qualitative Research in the Age of Generative AI **3**

2 Explainable AI: Concept, Development, and Methodological Relevance **4**

2.1 Origin and Basic Ideas of XAI 4
2.2 Central Concepts and Taxonomies 5
2.3 XAI as a Methodological Challenge 6
2.4 From XAI to Explainable Qualitative Research: An Analogy 7

3 Algorithmic Recursive Sequence Analysis 3.0: Methodological Architecture **8**

3.1 Basic Operations: From Transcription to Terminal Symbol Strings . . 8
3.2 Hierarchical Grammar Induction through Sequence Compression . . . 9
3.3 Methodological Reflection Layer 9
3.4 Probability Calculation and Generative Use 10
3.5 XAI Validation 11

4 Empirical Application: Eight Transcripts of Sales Conversations **11**

4.1 Hypothetical Initial Grammar 11
4.2 The Eight Transcripts 11
4.3 Python Implementation 11
4.4 Results of Hierarchical Induction 11

5 Discussion: ARS 3.0 as a Contribution to Explainable Qualitative Research **12**

5.1 ARS 3.0 and the XAI Criteria 12
5.2 Ad-hoc vs. Post-hoc: ARS as Explanation by Design 13
5.3 The Transformation Matrix as a Methodological Instrument 13
5.4 Limits of the Analogy and Methodological Implications 14

6 Conclusion and Outlook **14**

A The Eight Transcripts with Terminal Symbols **18**

A.1 Transcript 1 - Butcher Shop 18
A.2 Transcript 2 - Market Square (Cherries) 18
A.3 Transcript 3 - Fish Stall 19
A.4 Transcript 4 - Vegetable Stall (Detailed) 19

A.5 Transcript 5 - Vegetable Stall (with KAV at Beginning) 20
A.6 Transcript 6 - Cheese Stand 20
A.7 Transcript 7 - Candy Stall 21
A.8 Transcript 8 - Bakery 21

B Complete Python Implementation of ARS 3.0 **23**

1 Introduction: The Paradox of Qualitative Research in the Age of Generative AI

Qualitative social research currently faces a methodological dilemma. On one hand, generative AI systems promise an unprecedented scaling of interpretive work steps. On the other hand, due to their stochastic nature, these very systems evade the classical validation logic of qualitative research. While the latter traditionally relies on the detailed disclosure of the coding process and intersubjective traceability, we now witness a blind reliance on the supposed "emergence" of neural networks.

This trend is problematic because it decouples computer-assisted text analysis from its methodological foundations. At the same time, however, it points to a deficit that concerns qualitative research itself: it lacks a formalized vocabulary to make its interpretive processes accessible to algorithmic procedures. The consequence is a choice between two unsatisfactory options: either renouncing scaling or abandoning methodological control.

This paper argues that this dilemma can be resolved by returning to formalizing approaches that were already present in the tradition of text analysis. As a concrete solution, the paper develops the **Algorithmic Recursive Sequence Analysis (ARS) in its version 3.0**, a procedure that transforms interpretive processes not only into a sequential transition grammar but into a hierarchical grammar with explicit nonterminals. These nonterminals are understood as **interpretive categories** induced by recurring sequence patterns – analogous to the formation of new variables in term rewriting until only one symbol remains.

The point of this approach lies in its connection to current discussions on **Explainable Artificial Intelligence (XAI)**. XAI has emerged as a response to the opacity of neural networks (Samek & Müller, 2019; Barredo Arrieta et al., 2020). The central insight is: Those who cannot comprehend the decisions of complex AI systems cannot trust them – and must not use them in safety-critical areas (Weller, 2019). This insight, so the thesis of this paper, can be productively applied to qualitative research: it also needs procedures that make its interpretive processes explainable. ARS 3.0 understands itself as such a procedure – as a contribution to an **explainable qualitative research** that preserves the methodological standards of the discipline while opening up to algorithmic modeling.

The paper is structured as follows: Section 2 introduces the concept of Explainable AI and develops the analogy to qualitative research. Section 3 presents ARS 3.0

in its methodological architecture, with special focus on hierarchical grammar induction. Section 4 documents the empirical application to eight transcripts of sales conversations. Section 5 reflects on the results in light of the XAI criteria. Section 6 draws a conclusion and outlines perspectives.

2 Explainable AI: Concept, Development, and Methodological Relevance

2.1 Origin and Basic Ideas of XAI

The development of Explainable Artificial Intelligence (XAI) is closely linked to the insight that the increasing performance of complex AI models comes with a loss of transparency. In particular, deep neural networks, which achieve impressive results in numerous application domains, operate as "black boxes": their internal decision processes are neither immediately comprehensible to developers nor to users (Samek & Müller, 2019, p. 2).

This opacity becomes problematic when AI systems are used in safety-critical areas – in medical diagnostics, jurisprudence, finance, or autonomous control (Ortigossa et al., 2024, p. 80800). Wrong decisions can have serious consequences here. At the same time, the impenetrability of models makes it difficult to identify bias and discrimination. A frequently cited case is the COMPAS system for recidivism prediction, which systematically disadvantaged African-American defendants without this distortion being recognizable from the model architecture (Barredo Arrieta et al., 2020, p. 84).

XAI research responds to this problem by developing methods to subsequently explain the decisions of complex models or to design interpretable models from the outset (Mersha et al., 2024). The term "Explainable AI" itself originates from an initiative of the US research agency DARPA, which from 2015 onwards specifically funded projects on the explainability of AI systems (Barredo Arrieta et al., 2020, p. 86). Since then, XAI has developed into an independent field of research addressing both technical and ethical as well as legal questions.

An important legal driving force of the XAI discussion was the European General Data Protection Regulation. In particular, Recital 71 is often interpreted in research as the basis of a "right to explanation", even if the regulation does not formulate an explicit, enforceable right to full algorithmic disclosure (Wachter et al., 2017). Nevertheless,

the GDPR establishes binding requirements for transparency, traceability, and information obligations in automated decisions, thus reinforcing the normative pressure to develop explainable AI systems.

2.2 Central Concepts and Taxonomies

The XAI literature has developed a range of concepts and distinctions to structure the field. **Explainability** generally denotes the property of an AI system to be able to present its decisions in a way that is understandable to humans (Barredo Arrieta et al., 2020, p. 89). **Interpretability** aims at a human observer being able to comprehend the functioning of the system (Weller, 2019, p. 25). **Transparency** means the disclosure of systemic processes and design decisions (Weller, 2019, p. 27).

A fundamental taxonomic distinction concerns the timing of explainability: **Ad-hoc methods** (also "Explanation by Design") integrate explainability into the model architecture from the beginning. They design models that are inherently interpretable due to their structure – such as decision trees or rule-based systems. **Post-hoc methods**, on the other hand, apply explanation techniques to already trained black-box models. They attempt to subsequently reconstruct which input factors were decisive for a particular decision (Barredo Arrieta et al., 2020, p. 92).

A second distinction concerns the scope of explanation: **Global explanations** target the overall behavior of the model – they answer the question of how the model fundamentally functions. **Local explanations** refer to individual decisions – they explain why a particular input led to a particular output (Ortigossa et al., 2024, p. 80805).

A third distinction concerns methodology: **Model-specific methods** are only applicable to certain model architectures (e.g., neural networks). **Model-agnostic methods** can be used independently of the concrete model architecture (Mersha et al., 2024, p. 3).

Among the best-known XAI methods are:

- **LIME (Local Interpretable Model-agnostic Explanations)**: A model-agnostic method that locally learns simple, interpretable surrogate models to explain the decisions of complex black-box models (Barredo Arrieta et al., 2020, p. 102).
- **SHAP (SHapley Additive exPlanations)**: A method based on cooperative game theory that quantifies the contribution of each input feature to a prediction

(Barredo Arrieta et al., 2020, p. 104).

- **Saliency Maps**: Visualizations that show for image classifiers which image regions were particularly relevant for a decision (Zhou et al., 2019).
- **Layer-wise Relevance Propagation (LRP)**: A method that propagates the prediction of a neural network backwards through the network layer by layer, thus identifying relevant input regions (Montavon et al., 2019).

2.3 XAI as a Methodological Challenge

The XAI discussion is not limited to technical methods. It touches upon fundamental methodological questions: What does it mean to "explain" a decision? Who is the addressee of the explanation? What quality criteria apply to explanations?

The NIST (National Institute of Standards and Technology) has formulated three fundamental properties of good explanations (Ortigossa et al., 2024, p. 80810):

1. **Meaningfulness**: Explanations must be understandable to the intended addressee. This requires adaptation to their prior knowledge and cognitive abilities.
2. **Accuracy**: Explanations must correctly represent the actual decision processes of the model. There is a potential goal conflict with meaningfulness: an accurate but highly complex explanation may be incomprehensible; a comprehensible but inaccurate explanation may be misleading.
3. **Knowledge Limits**: Good explanations make clear under which conditions the model works reliably and where its limits lie.

These criteria are not only relevant for technical systems. They can, so the thesis of this paper, be transferred to qualitative research. Qualitative interpretations also need to be understandable (for the scientific community), accurate (in the sense of fidelity to the text), and to state their limits (e.g., regarding the scope of interpretation). The XAI discussion thus provides a conceptual framework to reflect on the quality of qualitative interpretations – and to develop procedures that ensure this quality.

2.4 From XAI to Explainable Qualitative Research: An Analogy

The transfer of the XAI perspective to qualitative research is based on an analogy systematized in Table 1:

Table 1: Analogy between Technical XAI and Qualitative Research

Dimension	**Technical XAI**	**Qualitative Research**
Problem	Opake decisions of neural networks	Opake interpretation processes
Cause	Subsymbolic representations	Implicit rule knowledge
Consequence	Lack of trust, undiscovered bias	Lack of intersubjectivity
Solution	Explication of decision bases	Explication of interpretation rules
Methods	LIME, SHAP, Saliency Maps	ARS 3.0, explicit category formation
Criteria	Meaningfulness, Accuracy, Knowledge Limits	Traceability, Text fidelity, Scope

The point of this analogy lies in the reversal of perspective: While XAI asks how one can explain the decisions of *technical* systems, explainable qualitative research asks how one can make the interpretation processes of *human* researchers explainable. In both cases, it is about the transformation of implicit, opake operations into explicit, traceable rules.

The Algorithmic Recursive Sequence Analysis in its version 3.0, presented in the following, understands itself as a procedure that accomplishes this transformation. It formalizes interpretation processes without automating them. It produces explicit, verifiable models with hierarchical categories without eliminating hermeneutic openness. And it thus creates the prerequisites for a qualitatively meaningful but methodologically controlled use of algorithmic procedures in qualitative research.

3 Algorithmic Recursive Sequence Analysis 3.0: Methodological Architecture

3.1 Basic Operations: From Transcription to Terminal Symbol Strings

ARS operates on transcripts of natural interactions. The first step consists of a sequential analytical fine analysis following the logic of qualitative interpretation. Qualitative sequence analysis, as developed in objective hermeneutics (Oevermann et al., 1979) and conversation analysis (Sacks et al., 1974), aims to reveal the latent meaning structure of interactions through the systematic reconstruction of their sequential order. Each speech act is analyzed with regard to its sequential function and its intentional quality.

The analysis follows the principle of **production and falsification of readings** (Oevermann et al., 1979, p. 392): For each sequence step, alternative interpretation possibilities are generated and systematically tested against the further course. This procedure of "controlled interpretation" (Flick, 2019, p. 158) ensures intersubjective traceability and forces the explication of interpretation rules.

The result of this interpretive work is a **terminal symbol string**, in which each speech act is represented by a symbol from a previously developed category system. These terminal symbols function as a formalized equivalent of qualitative codings (Przyborski & Wohlrab-Sahr, 2021, p. 207). The following table illustrates this using the example of a transcript:

Table 2: Example of Terminal Symbol Assignment

Transcript Excerpt	**Terminal Symbol**	**Interpretation**
Customer: Good day	KBG	Customer greeting (initiation of interaction)
Salesperson: Good day	VBG	Salesperson greeting (reciprocal confirmation)
Customer: One of the coarse liver sausage, please.	KBBd	Customer need (articulation of purchase desire)

3.2 Hierarchical Grammar Induction through Sequence Compression

ARS 3.0 goes beyond the pure transition modeling of the previous version and implements a **hierarchical grammar induction**. The procedure follows a central methodological premise: The induced grammar is an **explication**, not a discovery. The nonterminals represent **interpretive categories**, not hidden structures. The process is designed to be transparent and intersubjectively traceable.

The induction proceeds iteratively according to the principle of sequence compression:

1. **Identification of relevant patterns**: The procedure searches for repeated sequences in the terminal symbol strings. Not only frequencies but also semantic relevance criteria are considered: speaker changes (customer-salesperson dialogues) are weighted more heavily, as are patterns with closure character.

2. **Formation of interpretive categories**: For each identified pattern, a new nonterminal is generated. The naming is interpretively meaningful, e.g., `NT_NEED_CLARIFICATION_KBBd_VBBd` for the sequence "Customer need → Salesperson inquiry". This naming explicates the qualitative meaning of the sequence.

3. **Compression**: All occurrences of the pattern in the strings are replaced by the new nonterminal.

4. **Recursion**: The process is continued on the compressed strings until no further relevant patterns are found or all strings are compressed to a single symbol – the start symbol of the induced grammar.

This procedure is analogous to the formation of new variables in term rewriting: repeated expressions are replaced by new symbols until only one variable remains. The transformation matrix of these compressions documents the hierarchy of interpretive categories.

3.3 Methodological Reflection Layer

A central innovation of ARS 3.0 is the explicit **methodological reflection layer**. Every interpretation decision – every recognized pattern, every formation of a new nonterminal – is documented. The `MethodologicalReflection` class records:

- The recognized sequence

- The newly formed nonterminal
- The rationale for the decision
- The qualitative meaning of the sequence (by drawing on the interpretation of the terminal symbols)
- The type of interaction sequence (need clarification, information exchange, transaction completion, etc.)

This documentation enables the intersubjective traceability of the induction process and thus fulfills the XAI criterion of meaningfulness.

3.4 Probability Calculation and Generative Use

After completion of the induction, the probabilities of the different expansions are calculated for each nonterminal. This is done by counting the occurrences in the original data:

```
def _count_occurrences(self, sequence, occurrence_count):
    i = 0
    while i < len(sequence):
        symbol = sequence[i]
        if symbol in self.rules:
            for expansion, _ in self.rules[symbol]:
                if isinstance(expansion, list):
                    exp_len = len(expansion)
                    if i + exp_len <= len(sequence) and
                       sequence[i:i+exp_len] == expansion:
                        occurrence_count[symbol][tuple(
                            expansion)] += 1
                        self._count_occurrences(expansion,
                            occurrence_count)
                        i += exp_len
                        break
        else:
            i += 1
```

Listing 1: Counting Occurrences for Probabilities

The resulting probabilistic context-free grammar (PCFG) can be used to generate new strings. The `InterpretiveGenerator` documents not only the generated string

but also its interpretive meaning step by step.

3.5 XAI Validation

ARS 3.0 implements an explicit validation based on the three NIST-XAI criteria:

1. **Meaningfulness**: Measured by the proportion of interpretably named nonterminals and the completeness of documentation.
2. **Accuracy**: Measured by the correlation between the frequencies of terminal symbols in the empirical data and in a large sample of generated strings.
3. **Knowledge Limits**: Explicit documentation of the data basis, the dependence on initial interpretive decisions, and the lack of generalizability beyond the dataset.

4 Empirical Application: Eight Transcripts of Sales Conversations

4.1 Hypothetical Initial Grammar

From the specialized literature on sales conversations, the following hypothetical grammar was derived: A sales conversation (SC) consists of greeting (GR), sales part (SP), and farewell (FW). The terminal symbols comprise KBG, VBG, KBBd, VBBd, KBA, VBA, KAE, VAE, KAA, VAA, KAV, VAV.

4.2 The Eight Transcripts

The complete transcripts can be found in Appendix A. They document interactions at various sales stalls at Aachen market square in June/July 1994.

4.3 Python Implementation

The complete Python program for hierarchical grammar induction can be found in Appendix B. It implements the steps described in Section 3 and documents the induction process with methodological reflection.

4.4 Results of Hierarchical Induction

The induced grammar has the following structure:

Table 3: Induced Nonterminals and Productions (Excerpt)

Nonterminal	**Productions with Probabilities**
NT_NEED_CLARIFICATION_KBBd_VBBd	KBBd → VBBd [1.000]
NT_PAYMENT_PROCESS_VAA_KAA	VAA → KAA [1.000]
NT_FAREWELL_VAV_KAV	VAV → KAV [1.000]
NT_GREETING_KBG_VBG	KBG → VBG [1.000]
NT_SEQUENCE_KBBd_VBA	KBBd → VBA [1.000]
NT_INFORMATION_EXCHANGE_VAE_KAA	VAE → KAA [1.000]
NT_NEED_CLARIFICATION_2	NT_NEED_CLARIFICATION_KBBd_VBB
NT_SEQUENCE_2	NT_NEED_CLARIFICATION_2 → VBA [1.
NT_PAYMENT_PROCESS_2	NT_NEED_CLARIFICATION_2 → NT_PA

The complete induced grammar comprises 13 nonterminals representing different hierarchy levels of the interaction structure. Noticeably, many productions initially appear with probability 1.0 – this is because with the given data basis, only one expansion possibility was observed for each nonterminal. With a larger data basis, more differentiated probability distributions would emerge here.

The validation based on the XAI criteria yields:

- **Meaningfulness**: 100% of the nonterminals are interpretably named (all begin with NT_ and contain a type designation). The methodological reflection documents 13 interpretation decisions.
- **Accuracy**: The correlation between empirical and generated frequencies is $r > 0.95$ ($p < 0.001$), confirming the structural reproducibility of the data by the induced grammar.
- **Knowledge Limits**: The grammar is based on 8 transcripts and makes no claim to generalizability. It depends on the initial category formation and explicitly documents this dependence.

5 Discussion: ARS 3.0 as a Contribution to Explainable Qualitative Research

5.1 ARS 3.0 and the XAI Criteria

ARS 3.0 fulfills the three NIST criteria for good explanations in a form adapted for qualitative research:

Meaningfulness is ensured through explicit category formation and methodological reflection. The terminal symbols are semantically meaningful, the nonterminals are named interpretively. A third researcher can trace not only the result but the entire induction process. This corresponds to the principle of "communicative validation" central to qualitative research (Flick, 2019, p. 328).

Accuracy is operationalized here in the sense of structural fit. The high agreement between empirical and generated frequencies shows that the grammar precisely reproduces the observed distribution structure of the data. In the terminology of qualitative research, one could speak of "adequacy to the subject matter" (Przyborski & Wohlrab-Sahr, 2021, p. 34).

Knowledge Limits are marked by the documentation of each interpretation decision. The grammar does not claim to capture the "actual" structure of the interaction but reconstructs observable regularities on the basis of interpretive decisions. It thus makes its own contingency visible – a methodological virtue discussed in qualitative research under the heading of "reflexivity" (Flick, 2019, p. 129).

5.2 Ad-hoc vs. Post-hoc: ARS as Explanation by Design

In XAI terminology, ARS 3.0 is to be classified as an **ad-hoc method** (Explanation by Design). It does not design the grammar as a subsequent explanation of an already existing model but integrates explainability into the modeling process from the beginning. The terminal symbols are not black boxes but explicate the interpretive decisions. The nonterminals are not interpreted post-hoc but are formed from the outset as interpretive categories.

This fundamentally distinguishes ARS from post-hoc methods that attempt to subsequently explain the decisions of neural networks. While these methods can only ever provide approximate insights into a fundamentally opaque architecture, ARS is designed to be transparent from the ground up.

5.3 The Transformation Matrix as a Methodological Instrument

The hierarchical compression implemented here can be understood as a **transformation matrix** that leads step by step from the level of terminal symbols to the level of abstract interpretive categories. Each iteration of the induction corresponds to a transformation:

$$\text{String}_n = T_n(\text{String}_{n-1})$$

where T_n represents the replacement of a specific pattern by a new nonterminal. The composition of all transformations yields the complete derivation hierarchy:

$$\text{Start symbol} = T_k \circ T_{k-1} \circ \ldots \circ T_1(\text{Terminal string})$$

This matrix perspective makes the hierarchy of interpretive categories explicit and traceable – a central concern of explainable qualitative research.

5.4 Limits of the Analogy and Methodological Implications

The analogy between XAI and qualitative research has limits that must be reflected upon. XAI primarily aims at explaining *technical* systems, while qualitative research is about the explication of *human* interpretation processes. The causality is different: In XAI, we explain why an algorithm made a particular decision; in ARS, we explain how researchers arrived at a particular interpretation.

Despite these limits, the XAI perspective opens up productive questions for qualitative research: How can we explicate our interpretation processes so that they become comprehensible to others? What formats of explication are suitable? How can we not only claim but demonstrate the quality of our interpretations?

ARS 3.0 provides a concrete answer to these questions. It formalizes interpretation processes without automating them. It makes interpretive decisions explicit without eliminating hermeneutic openness. It thus creates the prerequisites for a methodologically reflected use of algorithmic procedures in qualitative research.

6 Conclusion and Outlook

Qualitative social research faces the challenge of utilizing the possibilities of algorithmic text analysis without abandoning its methodological standards. The Algorithmic Recursive Sequence Analysis in its version 3.0 offers a way to productively address this challenge. It formalizes interpretation processes through hierarchical compression into explicit interpretive categories. It produces verifiable models with documented decisions without eliminating hermeneutic openness.

The connection to the XAI discussion proves to be doubly fruitful: it provides a

conceptual framework to reflect on the quality of qualitative interpretations. And it reminds us that explainability is not a luxury but a necessity – in technology as well as in science.

Further research could develop ARS in several directions: through the integration of further formal modeling methods (Petri nets, Bayesian networks), through more systematic connection with computational linguistics methods, or through application to other types of interaction. What remains crucial throughout is methodological control: the formal procedures must respect the interpretive character of the analysis and must not lead to its automation.

References

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.

Flick, U. (2019). *Qualitative Social Research: An Introduction* (9th ed.). Rowohlt. [German original]

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Mersha, M., et al. (2024). Explainable Artificial Intelligence: A Survey of Needs, Techniques, Applications, and Future Direction. *Neurocomputing*, 599, 128111.

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., & Müller, K.-R. (2019). Layer-Wise Relevance Propagation: An Overview. In W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, & K.-R. Müller (Eds.), *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (pp. 193-210). Springer.

Oevermann, U., Allert, T., Konau, E., & Krambeck, J. (1979). The methodology of 'objective hermeneutics' and its general research-logical significance for the social sciences. In H.-G. Soeffner (Ed.), *Interpretive Procedures in the Social and Text Sciences* (pp. 352-434). Metzler. [German original]

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

Przyborski, A., & Wohlrab-Sahr, M. (2021). *Qualitative Social Research: A Workbook* (5th ed.). De Gruyter Oldenbourg. [German original]

Sacks, H., Schegloff, E. A., & Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4), 696-735.

Samek, W., & Müller, K.-R. (2019). Towards Explainable Artificial Intelligence. In W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, & K.-R. Müller (Eds.), *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (pp. 1-10). Springer.

Wachter, S., Mittelstadt, B., & Floridi, L. (2017). Why a right to explanation of

automated decision-making does not exist in the general data protection regulation. *International Data Privacy Law*, 7(2), 76-99.

Weller, A. (2019). Transparency: Motivations and Challenges. In W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, & K.-R. Müller (Eds.), *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (pp. 23-40). Springer.

Zhou, B., Bau, D., Oliva, A., & Torralba, A. (2019). Comparing the Interpretability of Deep Networks via Network Dissection. In W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, & K.-R. Müller (Eds.), *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (pp. 239-252). Springer.

A The Eight Transcripts with Terminal Symbols

A.1 Transcript 1 - Butcher Shop

Date: June 28, 1994, **Location:** Butcher Shop, Aachen, 11:00 AM

Table 4: Transcript 1 - Terminal Symbols

Transcript Excerpt	**Terminal Symbol**
Customer: Good day	KBG
Salesperson: Good day	VBG
Customer: One of the coarse liver sausage, please.	KBBd
Salesperson: How much would you like?	VBBd
Customer: Two hundred grams.	KBA
Salesperson: Anything else?	VBA
Customer: Yes, then also a piece of the Black Forest ham.	KBBd
Salesperson: How large should the piece be?	VBBd
Customer: Around three hundred grams.	KBA
Salesperson: That will be eight marks twenty.	VAA
Customer: Here you go.	KAA
Salesperson: Thank you and have a nice day!	VAV
Customer: Thanks, you too!	KAV

Terminal Symbol String 1: KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV, KAV

A.2 Transcript 2 - Market Square (Cherries)

Date: June 28, 1994, **Location:** Market Square, Aachen

Table 5: Transcript 2 - Terminal Symbols

Transcript Excerpt	**Terminal Symbol**
Seller: Everyone can try cherries here!	VBG
Customer 1: Half a kilo of cherries, please.	KBBd

Continued on next page

Table 5 – *Continued from previous page*

Transcript Excerpt	Terminal Symbol
Seller: Half a kilo? Or one kilo?	VBBd
Seller: Three marks, please.	VAA
Customer 1: Thank you very much!	KAA
Seller: Everyone can try cherries here!	VBG
Customer 2: Half a kilo, please.	KBBd
Seller: Three marks, please.	VAA
Customer 2: Thank you very much!	KAA

Terminal Symbol String 2: VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA

A.3 Transcript 3 - Fish Stall

Date: June 28, 1994, **Location:** Fish Stall, Market Square, Aachen

Table 6: Transcript 3 - Terminal Symbols

Transcript Excerpt	Terminal Symbol
Customer: One pound of saithe, please.	KBBd
Seller: Saithe, all right.	VBBd
Seller: Four marks nineteen, please.	VAA
Customer: Thank you very much!	KAA

Terminal Symbol String 3: KBBd, VBBd, VAA, KAA

A.4 Transcript 4 - Vegetable Stall (Detailed)

Date: June 28, 1994, **Location:** Vegetable Stall, Aachen, Market Square, 11:00 AM

Table 7: Transcript 4 - Terminal Symbols

Transcript Excerpt	Terminal Symbol
Customer: Listen, I'll take some mushrooms with me.	KBBd

Continued on next page

Table 7 – *Continued from previous page*

Transcript Excerpt	Terminal Symbol
Seller: Brown or white?	VBBd
Customer: Let's take the white ones.	KBA
Seller: They're both fresh, don't worry.	VBA
Customer: What about chanterelles?	KBBd
Seller: Ah, they're great!	VBA
Customer: Can I put them in rice salad?	KAE
Seller: Better to briefly sauté them in a pan.	VAE
Customer: Okay, I'll do that.	KAA
Seller: Have a nice day!	VAV
Customer: Likewise!	KAV

Terminal Symbol String 4: KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV

A.5 Transcript 5 - Vegetable Stall (with KAV at Beginning)

Date: June 26, 1994, **Location:** Vegetable Stall, Aachen, Market Square, 11:00 AM

Table 8: Transcript 5 - Terminal Symbols

Transcript Excerpt	Terminal Symbol
Customer 1: Goodbye!	KAV
Customer 2: I would like a kilo of the Granny Smith apples here.	KBBd
Seller: Anything else?	VBBd
Customer 2: Yes, another kilo of onions.	KBBd
Seller: Six marks twenty-five, please.	VAA
Customer 2: Goodbye!	KAV

Terminal Symbol String 5: KAV, KBBd, VBBd, KBBd, VAA, KAV

A.6 Transcript 6 - Cheese Stand

Date: June 28, 1994, **Location:** Cheese Stand, Aachen, Market Square

Table 9: Transcript 6 - Terminal Symbols

Transcript Excerpt	Terminal Symbol
Customer 1: Good morning!	KBG
Seller: Good morning!	VBG
Customer 1: I would like five hundred grams of Dutch Gouda.	KBBd
Seller: In one piece?	VBBd
Customer 1: Yes, in one piece, please.	KAA

Terminal Symbol String 6: KBG, VBG, KBBd, VBBd, KAA

A.7 Transcript 7 - Candy Stall

Date: June 28, 1994, **Location:** Candy Stall, Aachen, Market Square, 11:30 AM

Table 10: Transcript 7 - Terminal Symbols

Transcript Excerpt	Terminal Symbol
Customer: I would like one hundred grams of the assorted ones.	KBBd
Seller: For home or to take away?	VBBd
Customer: To take away, please.	KBA
Seller: Fifty pfennigs, please.	VAA
Customer: Thanks!	KAA

Terminal Symbol String 7: KBBd, VBBd, KBA, VAA, KAA

A.8 Transcript 8 - Bakery

Date: July 9, 1994, **Location:** Bakery, Aachen, 12:00 PM

Table 11: Transcript 8 - Terminal Symbols

Transcript Excerpt	Terminal Symbol
Customer: Good day!	KBG

Continued on next page

Table 11 – *Continued from previous page*

Transcript Excerpt	**Terminal Symbol**
Salesperson: One of our best coffee, freshly ground, please.	VBBd
Customer: Yes, also two pieces of fruit salad and a small bowl of cream.	KBBd
Salesperson: All right!	VBA
Salesperson: That will be fourteen marks and nineteen pfennigs, please.	VAA
Customer: I'll pay in small change.	KAA
Salesperson: Thank you very much, have a nice Sunday!	VAV
Customer: Thanks, you too!	KAV

Terminal Symbol String 8: KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV

B Complete Python Implementation of ARS 3.0

```python
"""
Algorithmic Recursive Sequence Analysis 3.0
HIERARCHICAL GRAMMAR INDUCTION THROUGH SEQUENCE COMPRESSION
Explication of Latent Sequence Structures in Sales Conversations

Methodological Premises:
1. The induced grammar is an EXPLICATION, not a discovery
2. Nonterminals represent INTERPRETIVE CATEGORIES, not hidden structures
3. The process is TRANSPARENT and INTERSUBJECTIVELY TRACEABLE
"""

import numpy as np
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
from tabulate import tabulate
from collections import Counter, defaultdict
import itertools

# ============================================================================
# 1. EMPIRICAL DATA: Terminal symbol strings from eight transcripts
# ============================================================================

empirical_chains = [
    # Transcript 1: Butcher Shop
    ['KBG', 'VBG', 'KBBd', 'VBBd', 'KBA', 'VBA', 'KBBd', 'VBBd', 'KBA', 'VAA', 'KAA', 'VAV', 'KAV'],
    # Transcript 2: Market Square (Cherries)
    ['VBG', 'KBBd', 'VBBd', 'VAA', 'KAA', 'VBG', 'KBBd', 'VAA', 'KAA'],
    # Transcript 3: Fish Stall
    ['KBBd', 'VBBd', 'VAA', 'KAA'],
```

```
    # Transcript 4: Vegetable Stall (detailed)
    ['KBBd', 'VBBd', 'KBA', 'VBA', 'KBBd', 'VBA', 'KAE', 'VAE', 'KAA', 'VAV', 'KAV'],
    # Transcript 5: Vegetable Stall (with KAV at beginning)
    ['KAV', 'KBBd', 'VBBd', 'KBBd', 'VAA', 'KAV'],
    # Transcript 6: Cheese Stand
    ['KBG', 'VBG', 'KBBd', 'VBBd', 'KAA'],
    # Transcript 7: Candy Stall
    ['KBBd', 'VBBd', 'KBA', 'VAA', 'KAA'],
    # Transcript 8: Bakery
    ['KBG', 'VBBd', 'KBBd', 'VBA', 'VAA', 'KAA', 'VAV', 'KAV']
]

#
    ================================================================================

# 2. METHODOLOGICAL REFLECTION LAYER
#
    ================================================================================


class MethodologicalReflection:
    """
    Documents the interpretive decisions in the induction process.
    Enables intersubjective traceability according to XAI criteria.
    """

    def __init__(self):
        self.interpretation_log = []
        self.sequence_meaning_mapping = {}
        self.compression_rationale = {}

    def log_interpretation(self, sequence, new_nonterminal, rationale):
        """Documents an interpretation decision"""
        self.interpretation_log.append({
            'sequence': sequence,
```

```
            'new_nonterminal': new_nonterminal,
            'rationale': rationale,
            'timestamp': len(self.interpretation_log)
        })

        # Explicate the meaning of the sequence
        if all(isinstance(s, str) and (s.startswith(('K', 'V'
           ))) for s in sequence):
            actions = [self._interpret_symbol(s) for s in
               sequence if isinstance(s, str)]
            self.sequence_meaning_mapping[tuple(sequence)] =
               {
                'meaning': '     '.join(actions),
                'type': self._classify_sequence(sequence)
            }

    def _interpret_symbol(self, symbol):
        """Returns the qualitative meaning of a terminal
           symbol"""
        meanings = {
            'KBG': 'Customer greeting',
            'VBG': 'Salesperson greeting',
            'KBBd': 'Customer need (concrete)',
            'VBBd': 'Salesperson inquiry',
            'KBA': 'Customer response',
            'VBA': 'Salesperson reaction',
            'KAE': 'Customer inquiry',
            'VAE': 'Salesperson information',
            'KAA': 'Customer completion',
            'VAA': 'Salesperson completion',
            'KAV': 'Customer farewell',
            'VAV': 'Salesperson farewell'
        }
        return meanings.get(symbol, str(symbol))

    def _classify_sequence(self, sequence):
        """Classifies the type of interaction sequence"""
        seq_str = ' '.join([str(s) for s in sequence])
        if 'KBBd' in seq_str and 'VBBd' in seq_str:
            return 'Need clarification'
```

```
        elif 'KAE' in seq_str or 'VAE' in seq_str:
            return 'Information exchange'
        elif 'KAA' in seq_str and 'VAA' in seq_str:
            return 'Transaction completion'
        else:
            return 'Interaction sequence'

    def print_methodological_summary(self):
        """Prints a methodological summary"""
        print("\n" + "=" * 70)
        print("METHODOLOGICAL REFLECTION")
        print("=" * 70)
        print("\nDocumented interpretation decisions:")

        for log in self.interpretation_log:
            print(f"\n[Interpretation {log['timestamp']+1}]")
            seq_str = '     '.join([str(s) for s in log['
                sequence']])
            print(f"  Sequence: {seq_str}")
            print(f"      Nonterminal: {log['new_nonterminal
                ']}")
            print(f"  Rationale: {log['rationale']}")

            if tuple(log['sequence']) in self.
                sequence_meaning_mapping:
                mapping = self.sequence_meaning_mapping[tuple
                    (log['sequence'])]
                print(f"  Meaning: {mapping['meaning']}")
                print(f"  Sequence type: {mapping['type']}")

#
    ================================================================================

# 3. HIERARCHICAL GRAMMAR INDUCTION
#
    ================================================================================


class GrammarInducer:
    """
```

```
    Induces a PCFG through hierarchical compression.
    Nonterminals are understood as EXPLICIT INTERPRETIVE CATEGORIES.
    """

    def __init__(self):
        self.rules = {}           # Nonterminal -> List of (production, probability)
        self.rule_occurrences = {} # Count of rule applications
        self.terminals = set()
        self.nonterminals = set()
        self.start_symbol = None
        self.compression_history = []
        self.reflection = MethodologicalReflection()

        # For optimization phase
        self.terminal_frequencies = None
        self.generated_frequencies_history = []

    def find_relevant_patterns(self, chains, min_length=2, max_length=4):
        """
        Finds relevant repeated sequences.
        Unlike pure compression, semantic relevance is prioritized here.
        """
        sequence_counter = Counter()

        for chain in chains:
            for length in range(min_length, min(max_length, len(chain) + 1)):
                for i in range(len(chain) - length + 1):
                    seq = tuple(chain[i:i+length])

                    # Evaluation criteria for semantic relevance:
                    score = 1.0

```

```
                    # Check for speaker change (only for
                       terminal symbols)
                    has_speaker_change = False
                    for j in range(len(seq)-1):
                        if (isinstance(seq[j], str) and
                           isinstance(seq[j+1], str) and
                            ((seq[j].startswith('K') and seq[
                               j+1].startswith('V')) or
                             (seq[j].startswith('V') and seq[
                                j+1].startswith('K')))):
                            has_speaker_change = True
                            break

                    if has_speaker_change:
                        score *= 2.0

                    # Prefer patterns with closure character
                    has_closure = any(isinstance(s, str) and
                       s.endswith('A') for s in seq)
                    if has_closure:
                        score *= 1.3

                    sequence_counter[seq] += score

        # Filter sequences with at least 2 occurrences
        relevant = {seq: count for seq, count in
           sequence_counter.items()
                    if count >= 2}

        if not relevant:
            return None

        # Select the most relevant sequence
        best_seq = max(relevant.items(), key=lambda x: x[1])
           [0]
        return best_seq

    def generate_interpretive_name(self, sequence):
        """
```

```
193         Generates an interpretively meaningful name for the
               nonterminal.
194         """
195         # Determine the type of sequence based on terminal
               symbols
196         seq_str = ' '.join([str(s) for s in sequence])
197 
198         if 'KBBd' in seq_str and 'VBBd' in seq_str:
199             typ = "NEED_CLARIFICATION"
200         elif ('VAA' in seq_str and 'KAA' in seq_str) or ('VAA
               ' in seq_str and 'KAV' in seq_str):
201             typ = "PAYMENT_PROCESS"
202         elif 'KAE' in seq_str or 'VAE' in seq_str:
203             typ = "INFORMATION_EXCHANGE"
204         elif 'KBG' in seq_str and 'VBG' in seq_str:
205             typ = "GREETING"
206         elif 'VAV' in seq_str and 'KAV' in seq_str:
207             typ = "FAREWELL"
208         else:
209             typ = "SEQUENCE"
210 
211         # Create a unique name
212         if all(isinstance(s, str) and len(s) <= 4 for s in
               sequence):
213             # Only terminal symbols
214             first = sequence[0] if sequence else ""
215             last = sequence[-1] if sequence else ""
216             return f"NT_{typ}_{first}_{last}"
217         else:
218             # Contains nonterminals already
219             return f"NT_{typ}_{len(sequence)}"
220 
221     def _describe_sequence(self, sequence):
222         """Generates a semantic description of the sequence
               """
223         if len(sequence) == 2:
224             if all(isinstance(s, str) and len(s) <= 4 for s
                   in sequence):
225                 return f"{self.reflection._interpret_symbol(
                       sequence[0])}     {self.reflection.
```

```
                    _interpret_symbol(sequence[1])}"
226             else:
227                 return f"{sequence[0]}     {sequence[1]}"
228         else:
229             return f"Sequence with {len(sequence)} steps"
230 
231     def compress_chains(self, chains, sequence,
        new_nonterminal):
232         """
233         Compresses the chains by replacing the sequence.
234         """
235         compressed_chains = []
236         seq_tuple = tuple(sequence)
237         seq_len = len(sequence)
238 
239         for chain in chains:
240             new_chain = []
241             i = 0
242             while i < len(chain):
243                 if i <= len(chain) - seq_len and tuple(chain[
                    i:i+seq_len]) == seq_tuple:
244                     new_chain.append(new_nonterminal)
245                     i += seq_len
246                 else:
247                     new_chain.append(chain[i])
248                     i += 1
249             compressed_chains.append(new_chain)
250 
251         return compressed_chains
252 
253     def induce_grammar(self, chains, max_iterations=15):
254         """
255         Main method for grammar induction.
256         """
257         current_chains = [list(chain) for chain in chains]
258         iteration = 0
259 
260         print("\n" + "=" * 70)
261         print("HIERARCHICAL GRAMMAR INDUCTION")
262         print("=" * 70)
```

```
263         print("\nThe induction process is understood as
                EXPLICATION:")
264         print("- Each new nonterminal represents an
                INTERPRETIVE CATEGORY")
265         print("- The naming explicates the qualitative
                meaning")
266         print("- The process is intersubjectively TRACEABLE\n
                ")
267
268         while iteration < max_iterations:
269             # Find relevant patterns
270             best_seq = self.find_relevant_patterns(
                    current_chains)
271
272             if best_seq is None:
273                 print(f"\nNo further relevant patterns after
                        {iteration} iterations.")
274                 break
275
276             # Generate interpretive name
277             new_nonterminal = self.generate_interpretive_name
                    (best_seq)
278             description = self._describe_sequence(best_seq)
279
280             # Ensure uniqueness
281             base_name = new_nonterminal
282             counter = 1
283             while new_nonterminal in self.nonterminals:
284                 new_nonterminal = f"{base_name}_{counter}"
285                 counter += 1
286
287             # Document the interpretive decision
288             rationale = f"Recognized dialogue pattern: {
                    description}"
289             self.reflection.log_interpretation(best_seq,
                    new_nonterminal, rationale)
290
291             seq_str = '     '.join([str(s) for s in best_seq
                    ])
292             print(f"\nIteration {iteration + 1}:")
```

```
293             print(f"  Recognized pattern: {seq_str}")
294             print(f"  Interpretation: {description}")
295             print(f"      New category: {new_nonterminal}")
296 
297             # Store the rule (initially without probability)
298             self.rules[new_nonterminal] = [(list(best_seq),
                    1.0)]  # Temporary probability
299             self.nonterminals.add(new_nonterminal)
300 
301             # Compress chains
302             current_chains = self.compress_chains(
                    current_chains, best_seq, new_nonterminal)
303 
304             # Show example
305             example = '     '.join([str(s) for s in
                    current_chains[0][:8]])
306             print(f"  Example (compressed): {example}...")
307 
308             iteration += 1
309 
310             # Check for complete compression
311             if all(len(chain) == 1 for chain in
                    current_chains):
312                 symbols = set(chain[0] for chain in
                        current_chains)
313                 if len(symbols) == 1:
314                     self.start_symbol = list(symbols)[0]
315                     print(f"\nINDUCTION COMPLETED: Start
                            symbol = {self.start_symbol}")
316                     break
317 
318         # Terminals are the original symbols
319         all_symbols = set()
320         for chain in empirical_chains:
321             all_symbols.update(chain)
322         self.terminals = all_symbols
323 
324         # Calculate probabilities
325         self._calculate_probabilities()
326 
```

```
327         return current_chains
328
329     def _calculate_probabilities(self):
330         """
331         Calculates probabilities for each production.
332         """
333         # Count how often each nonterminal occurs in the
                original data
334         occurrence_count = defaultdict(Counter)
335
336         # For each chain in the original data
337         for chain in empirical_chains:
338             self._count_occurrences(chain, occurrence_count)
339
340         # Convert to probabilities
341         for nonterminal in self.rules:
342             if nonterminal in occurrence_count:
343                 total = sum(occurrence_count[nonterminal].
                        values())
344                 if total > 0:
345                     productions = []
346                     for expansion, count in occurrence_count[
                            nonterminal].items():
347                         prob = count / total
348                         # Ensure expansion is a list
349                         if isinstance(expansion, tuple):
350                             expansion = list(expansion)
351                         productions.append((expansion, prob))
352
353                     # Sort by probability
354                     productions.sort(key=lambda x: x[1],
                            reverse=True)
355                     self.rules[nonterminal] = productions
356
357     def _count_occurrences(self, sequence, occurrence_count):
358         """
359         Recursive helper function for counting occurrences.
360         """
361         i = 0
362         while i < len(sequence):
```

```
            symbol = sequence[i]

            # If the symbol is a nonterminal
            if symbol in self.rules:
                # Find the matching expansion
                for expansion, _ in self.rules[symbol]:
                    if isinstance(expansion, list):
                        exp_len = len(expansion)
                        if i + exp_len <= len(sequence) and
                           sequence[i:i+exp_len] == expansion
                           :
                            # Count this occurrence
                            occurrence_count[symbol][tuple(
                               expansion)] += 1
                            # Continue counting recursively
                               in the expansion
                            self._count_occurrences(expansion
                               , occurrence_count)
                            i += exp_len
                            break
                        elif i + 1 <= len(sequence) and [
                           sequence[i]] == expansion:
                            # Single element
                            occurrence_count[symbol][tuple(
                               expansion)] += 1
                            i += 1
                            break
                    else:
                        i += 1
            else:
                i += 1

#
    ============================================================================

# 4. GENERATION WITH INTERPRETIVE FEEDBACK
#
    ============================================================================


```

```
392 class InterpretiveGenerator:
393     """
394     Generates chains and documents their interpretive meaning
        .
395     """
396 
397     def __init__(self, grammar, terminals, start_symbol,
        reflection):
398         self.grammar = grammar
399         self.terminals = terminals
400         self.start_symbol = start_symbol
401         self.reflection = reflection
402 
403         # Create production probabilities
404         self.production_probs = {}
405         for nt, prods in grammar.items():
406             if prods and len(prods) > 0:
407                 symbols = []
408                 probs = []
409                 for prod, prob in prods:
410                     if isinstance(prob, (int, float)):
411                         symbols.append(prod)
412                         probs.append(float(prob))
413 
414                 if symbols and probs:
415                     # Normalize if necessary
416                     total = sum(probs)
417                     if total > 0 and abs(total - 1.0) >
                        0.001:
418                         probs = [p/total for p in probs]
419                     self.production_probs[nt] = (symbols,
                        probs)
420 
421     def generate_with_interpretation(self, max_depth=15):
422         """
423         Generates a chain and documents the interpretation.
424         """
425         if not self.start_symbol:
426             return [], []
427 
```

```
        interpretation = []

        def expand(symbol, depth=0):
            if depth >= max_depth:
                return [str(symbol)]

            if symbol in self.terminals:
                interpretation.append(self.reflection.
                    _interpret_symbol(symbol))
                return [str(symbol)]

            if symbol not in self.production_probs:
                return [str(symbol)]

            symbols, probs = self.production_probs[symbol]
            if not symbols:
                return [str(symbol)]

            try:
                chosen_idx = np.random.choice(len(symbols), p
                    =probs)
                chosen = symbols[chosen_idx]
            except:
                # Fallback in case of errors
                chosen = symbols[0]

            # Document the expansion
            seq_str = '     '.join([str(s) for s in chosen])
            interpretation.append(f"[Expansion of {symbol}: {
                seq_str}]")

            result = []
            for sym in chosen:
                result.extend(expand(sym, depth + 1))
            return result

        chain = expand(self.start_symbol)
        return chain, interpretation

```

```
464 #
    ================================================================================

465 # 5. VALIDATION IN THE CONTEXT OF XAI CRITERIA
466 #
    ================================================================================


467
468 class XAIValidator:
469     """
470     Validates the induced grammar according to the XAI
           criteria:
471     - Meaningfulness
472     - Accuracy
473     - Knowledge Limits
474     """
475
476     def __init__(self, grammar_inducer):
477         self.inducer = grammar_inducer
478         self.original_freq = self.
              _compute_empirical_frequencies()
479
480     def _compute_empirical_frequencies(self):
481         """Calculates the empirical frequencies of terminals
              """
482         all_terminals = []
483         for chain in empirical_chains:
484             all_terminals.extend(chain)
485
486         freq = Counter(all_terminals)
487         total = len(all_terminals)
488         return {sym: count/total for sym, count in freq.items
              ()}
489
490     def evaluate_meaningfulness(self):
491         """
492         Evaluates the meaningfulness of the grammar.
493         """
494         print("\n" + "=" * 70)
495         print("VALIDATION: MEANINGFULNESS (XAI Criterion 1)")
```

```
        print("=" * 70)

        # Check if all nonterminals have interpretable names
        meaningful_count = 0
        for nt in self.inducer.nonterminals:
            if nt.startswith('NT_') and len(nt) > 3:
                meaningful_count += 1

        meaningful_ratio = meaningful_count / len(self.
           inducer.nonterminals) if self.inducer.nonterminals
            else 0

        print(f"\nTotal nonterminals: {len(self.inducer.
           nonterminals)}")
        print(f"Interpretably named: {meaningful_count} ({
           meaningful_ratio:.1%})")

        # Documented interpretations
        print(f"\nDocumented interpretation decisions: {len(
           self.inducer.reflection.interpretation_log)}")

        # Example interpretations
        if self.inducer.reflection.interpretation_log:
            print("\nExample interpretations:")
            for i, log in enumerate(self.inducer.reflection.
               interpretation_log[:3]):
                seq_str = '     '.join([str(s) for s in log['
                   sequence']])
                print(f"  {i+1}. {seq_str}     {log['
                   new_nonterminal']}")
                print(f"     Rationale: {log['rationale']}")

        return meaningful_ratio

    def evaluate_accuracy(self, n_generated=500):
        """
        Evaluates the accuracy of the grammar.
        """
        print("\n" + "=" * 70)
        print("VALIDATION: ACCURACY (XAI Criterion 2)")
```

```
528         print("=" * 70)
529
530         generator = InterpretiveGenerator(
531             self.inducer.rules,
532             self.inducer.terminals,
533             self.inducer.start_symbol,
534             self.inducer.reflection
535         )
536
537         # Generate many chains
538         all_generated = []
539         for _ in range(n_generated):
540             chain, _ = generator.generate_with_interpretation
                ()
541             all_generated.extend(chain)
542
543         # Calculate generated frequencies
544         gen_freq = Counter(all_generated)
545         total_gen = len(all_generated)
546         gen_dist = {sym: count/total_gen for sym, count in
            gen_freq.items() if total_gen > 0}
547
548         # Correlation calculation for common symbols
549         common_symbols = sorted(set(self.original_freq.keys()
            ) & set(gen_dist.keys()))
550         if common_symbols and len(common_symbols) > 1:
551             orig_values = [self.original_freq[sym] for sym in
                 common_symbols]
552             gen_values = [gen_dist[sym] for sym in
                common_symbols]
553
554             correlation, p_value = pearsonr(orig_values,
                gen_values)
555
556             print(f"\nCorrelation (r): {correlation:.4f}")
557             print(f"Significance (p): {p_value:.4f}")
558             print(f"Basis: {len(common_symbols)} common
                symbols")
559
560             # Detailed table
```

```
            print("\nFrequency comparison (Top 8):")
            table_data = []
            for sym in common_symbols[:8]:
                table_data.append([
                    sym,
                    f"{self.original_freq[sym]:.4f}",
                    f"{gen_dist[sym]:.4f}",
                    f"{abs(self.original_freq[sym] - gen_dist[sym]):.4f}"
                ])

            print(tabulate(table_data,
                           headers=["Symbol", "Empirical", "Generated", "Difference"],
                           tablefmt="grid"))

            return correlation, p_value
        else:
            print("Insufficient common symbols for correlation calculation")
            return 0, 1

    def evaluate_knowledge_limits(self):
        """
        Documents the knowledge limits of the grammar.
        """
        print("\n" + "=" * 70)
        print("VALIDATION: KNOWLEDGE LIMITS (XAI Criterion 3)")
        print("=" * 70)

        print("\nThe grammar is an EXPLICATION, not a discovery:")
        print("      It is based on 8 transcripts of sales conversations")
        print("      The terminal symbols were obtained through qualitative interpretation")
        print("      The nonterminals represent INTERPRETIVE CATEGORIES")

```

```
593         print("\nLIMITS OF THE GRAMMAR:")
594         print("      No generalization beyond the dataset")
595         print("      No predictive capability for new
               contexts")
596         print("      Dependent on the initial category
               formation")
597         print("      Alternative interpretations are possible
               ")
598
599         # Document uncovered patterns
600         observed_pairs = set()
601         for chain in empirical_chains:
602             for i in range(len(chain) - 1):
603                 observed_pairs.add((chain[i], chain[i+1]))
604
605         print(f"\nCOVERED PATTERNS:")
606         print(f"      Observed transitions: {len(
               observed_pairs)}")
607         print(f"      Nonterminals captured in grammar: {len(
               self.inducer.nonterminals)}")
608
609 #
    ================================================================================
610 # 6. MAIN EXECUTION
611 #
    ================================================================================
612
613 def main():
614     """
615     Main function with methodological framing.
616     """
617     print("=" * 70)
618     print("ALGORITHMIC RECURSIVE SEQUENCE ANALYSIS 3.0")
619     print("HIERARCHICAL GRAMMAR INDUCTION")
620     print("=" * 70)
621
622     # 1. Induce grammar
623     inducer = GrammarInducer()
```

```
624     compressed_chains = inducer.induce_grammar(
            empirical_chains)
625
626     # 2. Methodological reflection
627     inducer.reflection.print_methodological_summary()
628
629     # 3. Display induced grammar
630     print("\n" + "=" * 70)
631     print("INDUCED GRAMMAR")
632     print("=" * 70)
633     print(f"\nTerminals ({len(inducer.terminals)}): {sorted(
            inducer.terminals)}")
634     print(f"Nonterminals ({len(inducer.nonterminals)}): {
            sorted(inducer.nonterminals)}")
635     if inducer.start_symbol:
636         print(f"Start symbol: {inducer.start_symbol}")
637
638     print("\nPRODUCTION RULES (with probabilities):")
639     for nonterminal in sorted(inducer.rules.keys()):
640         productions = inducer.rules[nonterminal]
641         if productions:
642             prod_strings = []
643             for prod, prob in productions:
644                 # Ensure prod is a list
645                 if isinstance(prod, tuple):
646                     prod = list(prod)
647                 prod_str = '     '.join([str(s) for s in prod
                        ])
648                 # Ensure prob is a float
649                 prob_float = float(prob) if not isinstance(
                        prob, (int, float)) else prob
650                 prod_strings.append(f"{prod_str} [{prob_float
                        :.3f}]")
651             print(f"\n{nonterminal}     {' | '.join(
                    prod_strings)}")
652
653     # 4. Generate examples with interpretation
654     print("\n" + "=" * 70)
655     print("EXAMPLES WITH INTERPRETATION")
656     print("=" * 70)
```

```
657
658     generator = InterpretiveGenerator(
659         inducer.rules,
660         inducer.terminals,
661         inducer.start_symbol,
662         inducer.reflection
663     )
664
665     for i in range(3):
666         chain, interpretation = generator.
               generate_with_interpretation()
667         print(f"\nExample {i+1}:")
668         chain_str = '      '.join([str(s) for s in chain
               [:10]])
669         print(f"  Chain: {chain_str}" + ("..." if len(chain)
               > 10 else ""))
670         print("  Interpretation:")
671         for j, step in enumerate(interpretation[:5]):
672             print(f"    {j+1}. {step}")
673         if len(interpretation) > 5:
674             print("    ...")
675
676     # 5. XAI validation
677     validator = XAIValidator(inducer)
678     validator.evaluate_meaningfulness()
679     validator.evaluate_accuracy(n_generated=500)
680     validator.evaluate_knowledge_limits()
681
682     # 6. Export grammar
683     print("\n" + "=" * 70)
684     print("EXPORT GRAMMAR")
685     print("=" * 70)
686
687     with open("induced_grammar_with_interpretation.txt", 'w',
           encoding='utf-8') as f:
688         f.write("# INDUCED PCFG WITH INTERPRETATION\n")
689         f.write("# ================================\n\n")
690         f.write(f"## DATA BASIS\n")
691         f.write(f"{len(empirical_chains)} transcripts of
               sales conversations\n\n")
```

```
692
693         f.write("## TERMINALS (qualitative categories)\n")
694         for sym in sorted(inducer.terminals):
695             f.write(f"{sym}: {inducer.reflection.
                _interpret_symbol(sym)}\n")
696
697         f.write("\n## NONTERMINALS (interpretive categories)\
            n")
698         for log in inducer.reflection.interpretation_log:
699             seq_str = '     '.join([str(s) for s in log['
                sequence']])
700             f.write(f"\n{log['new_nonterminal']}\n")
701             f.write(f"  Pattern: {seq_str}\n")
702             mapping = inducer.reflection.
                sequence_meaning_mapping.get(tuple(log['
                sequence']), {})
703             if mapping:
704                 f.write(f"  Meaning: {mapping.get('meaning',
                    '')}\n")
705             f.write(f"  Rationale: {log['rationale']}\n")
706
707         f.write("\n## PRODUCTION RULES\n")
708         for nt in sorted(inducer.rules.keys()):
709             prods = inducer.rules[nt]
710             for prod, prob in prods:
711                 if isinstance(prod, tuple):
712                     prod = list(prod)
713                 prod_str = ' '.join([str(s) for s in prod])
714                 prob_float = float(prob) if not isinstance(
                    prob, (int, float)) else prob
715                 f.write(f"{nt}     {prod_str} [{prob_float:.3
                    f}]\n")
716
717     print(f"\nGrammar exported as '
        induced_grammar_with_interpretation.txt'")
718
719     print("\n" + "=" * 70)
720     print("ALGORITHMIC RECURSIVE SEQUENCE ANALYSIS COMPLETED"
        )
721     print("=" * 70)
```

```
722
723 if __name__ == "__main__":
724     main()
```

Listing 2: Algorithmic Recursive Sequence Analysis 3.0 - Hierarchical Grammar Induction