

Template-Guided Rule Generation and Evaluation for Data Quality using Large Language Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Alexander Schneider

Matrikelnummer 01527238

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Em.O.Univ.Prof. Dipl.-Ing. Dr.techn. Georg Gottlob

Mitwirkung: Dr. Jinsong Guo

Wien, 10. Oktober 2025

Alexander Schneider

Georg Gottlob



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Template-Guided Rule Generation and Evaluation for Data Quality using Large Language Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Alexander Schneider

Registration Number 01527238

to the Faculty of Informatics

at the TU Wien

Advisor: Em.O.Univ.Prof. Dipl.-Ing. Dr.techn. Georg Gottlob

Assistance: Dr. Jinsong Guo

Vienna, October 10, 2025

Alexander Schneider

Georg Gottlob



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Alexander Schneider

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 10. Oktober 2025

Alexander Schneider



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich bedanke mich bei meinem Betreuer, Professor Georg Gottlob, für die wertvolle Unterstützung während der Entstehung dieser Arbeit. Sein fundiertes Wissen und seine Vorarbeiten bildeten eine essentielle Grundlage für die Entwicklung und Ausarbeitung meiner Forschung.

Des weiteren bedanke ich mich bei Dr. Jinsong Guo für die ausgezeichnete Zusammenarbeit und die Möglichkeit, mein Wissen in einem internationalen Forschungsteam einzubringen. Seine Führung und Unterstützung sowohl bei der praktischen Implementierung als auch bei der wissenschaftlichen Ausrichtung meiner Arbeit waren wesentlich. Ich bedanke mich recht herzlich beim gesamten Forschungsteam von Unlimidata für die Zusammenarbeit und die diversen Inputs und Hilfestellung während meiner Forschungszeit.

Ich danke meiner Familie für ihre Unterstützung während meines Studiums, insbesondere meiner Mutter, die immer an mich geglaubt hat und mir über all die Jahre stets den Mut und Zuspruch gegeben hat, weiterzumachen. Mein besonderer Dank gilt auch meiner Partnerin Karolyn, für ihre Geduld, ihre Unterstützung und ihr Verständnis, speziell in den letzten Monaten. Danke, dass ihr immer an mich geglaubt habt, auch in Zeiten, in denen ich selbst an mir gezweifelt habe.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to thank my supervisor, Professor Georg Gottlob, for his initial ideas and the support during the development of this work. His knowledge and preliminary work were essential for the creation and elaboration of my thesis.

Furthermore, I thank Dr. Jinsong Guo for the collaboration and the opportunity to contribute my knowledge in an international research team. His guidance and support, both in the practical implementation and in the academic direction of my work, were vital. I am very grateful to the entire research team at Unlimidata for the collaboration and the various inputs and assistance they gave me during my research period.

I thank my family for their support during my studies, especially my mother, who has always believed in me and given me the courage and encouragement to keep going throughout all these years. My special thanks also go to my partner Karolyn, for her patience, her support, and her understanding, especially in the last few months. Thank you for always believing in me, even in times when I doubted myself.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Hochwertige Daten sind essentiell für zuverlässige datengetriebene Systeme, dennoch stehen Organisationen vor erheblichen Herausforderungen bei der systematischen Sicherstellung der Datenqualität. Traditionelle Ansätze verlassen sich stark auf Domänenexperten, um Datenqualitätsregeln manuell zu definieren. Dieser Prozess ist kostspielig, zeitaufwändig und schwer zu skalieren ist. Bestehende automatisierte Verfahren sind oft auf enge statistische Muster beschränkt und verfügen nicht über das notwendige semantische Verständnis um domänenspezifische Regeln zu bewerten.

Diese Arbeit präsentiert ein Framework zur automatisierten Generierung und Evaluierung von Datenqualitätsregeln mit Hilfe von Large Language Models (LLMs). Das zentrale Element ist ein template-basierter Generierungsprozess, der LLMs durch strukturierte Vorgaben gezielt in der Regelbildung steuert und so die Grenzen naiver Prompting-Strategien überwindet.

Für die Bewertung wird ein mehrstufiger Prozess eingeführt. Die neu entwickelte Metrik Semantic Score misst die Effektivität von Regeln anhand tatsächlicher Regelverletzungen, während ein Business Domain Judge unangemessene oder semantisch bedeutungslose Regeln herausfiltert.

Die Evaluierung anhand von drei Datensätzen zeigt, dass der Ansatz 75-100% der expertendefinierten Regeln generiert und zusätzliche wertvolle Regeln entdeckt. Die Semantic-Score Metrik wurde durch synthetische Verletzungen validiert und zeigt eine durchschnittliche Erhöhung von 41% bei betroffenen Regeln gegenüber nur 1,7% Variation im Kontrollset. Zusätzlich wurde der Business Domain Judge im Bezug auf Zuverlässigkeit evaluiert und statistisch ausgewertet.

Diese Arbeit demonstriert, dass LLM-basierte Frameworks die Regelgenerierung und Evaluierung effektiv automatisieren können, wodurch die Abhängigkeit von Domänenexperten reduziert wird. Offene Fragen bestehen hinsichtlich der Verlässlichkeit der Evaluierungsprozesse in verschiedenen Anwendungsdomänen und der Vollständigkeit des Generierungsprozesses.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

High-quality data is fundamental to reliable data-driven systems, yet organizations face significant challenges in systematically ensuring data quality as datasets grow in volume and complexity. Traditional approaches rely heavily on domain experts to manually define data quality rules. This process is costly, time-consuming and difficult to scale. While automated rule mining methods exist, they are often limited to narrow statistical patterns and lack the semantic understanding necessary to capture meaningful, domain-specific constraints.

This thesis proposes a novel framework for automated data quality rule generation and evaluation by leveraging Large Language Models (LLMs). The core contribution is a template-driven rule generation approach that guides LLMs using structured templates specifying rule complexity and shape, systematically addressing the limitations of naive LLM prompting while potentially enabling the discovery of novel constraints beyond conventional categories.

To evaluate the many candidate rules generated, the framework introduces a multi-step evaluation process. A novel semantic-score metric measures a rule's effectiveness in identifying genuine data errors using an LLM-as-a-Judge system to classify rule violations. A subsequent Business Domain Judge filters out domain-inappropriate, overly specific, or semantically meaningless rules.

Evaluation on three datasets demonstrates that the template-driven approach successfully generates 75-100% of expert-defined golden rules while discovering additional valuable constraints. The semantic-score metric was validated by introducing synthetic violations, showing an average 41% improvement for rules detecting these violations versus only 1.7% variation in control rules. The Business Domain Judge demonstrated strong reliability on clearly defined domains while appropriately reflecting genuine domain uncertainties.

This work demonstrates that LLM-based frameworks can effectively automate both rule generation and evaluation when provided with appropriate structural guidance, reducing reliance on domain experts while maintaining rule quality and coverage. Open questions remain regarding judge reliability across different domains and the completeness guarantees of the generation process.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
Research Collaboration	1
1 Introduction	3
1.1 Motivation	3
1.2 Research Questions	4
1.3 Thesis Structure	5
2 Theoretical Background	7
2.1 Machine Learning	7
2.2 Large Language Models	15
2.3 Database Theory Fundamentals	33
3 Related Work	45
3.1 Traditional Rule Mining	45
3.2 LLMs and Data Quality Management	46
3.3 LLM-as-a-Judge	47
4 Framework and Implementation	49
4.1 Framework Architecture	49
4.2 Rule Generation	53
4.3 Rule Compilation and Execution	61
4.4 Rule Evaluation	61
5 Results and Discussion	71
5.1 Rule Generation Evaluation	71
5.2 Semantic Score Evaluation	78
5.3 Business Domain Judge Evaluation	81
5.4 Limitations and Challenges	85
	xv

6 Conclusion and Future Work	87
6.1 Summary	87
6.2 Future Research	88
Overview of Generative AI Tools Used	91
List of Figures	93
List of Tables	95
Bibliography	97

Research Collaboration

This research was conducted in collaboration with Unlimidata, a company specializing in automated data curation solutions. I worked closely with researchers from their team involved in the development of the **Chat2Data** system [47]. This collaboration was established solely for academic and research purposes.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Introduction

1.1 Motivation

High-quality data is the foundation of reliable data-driven systems, yet organizations continue to struggle with systematic data quality assurance as data volume and complexity grow. Organizations typically rely on domain experts to manually define data quality rules such as integrity constraints, business rules or domain-specific checks. This manual effort is not only costly and time-consuming, but also inherently limits scalability.

Automated approaches to rule discovery exist, for example mining algorithms to find functional dependencies or denial constraints. However, these methods are often restricted to narrow statistical or logical patterns and lack the semantic understanding needed to capture domain-specific and meaningful data quality rules.

The recent rise of modern LLMs provides a new opportunity to assist or automate data quality assurance tasks. LLMs can interpret dataset semantics and generate candidate rules in structured forms (e.g. as SQL constraints, Prolog rules or as natural language rules).

However, when used naively (e.g. simply prompting the LLM to "generate rules for this dataset"), LLMs often generate results of poor quality. They may produce rules that are too obvious (e.g. "column name should not be empty"), overly specific or even hallucinated rules. The inherent *laziness* of LLMs is also problematic, as they may omit useful rules or provide rules that are too generic. Even advanced prompting strategies that include few-shot examples, or prompts that specify the type of rule to be generated still face this problem. Thus, current LLMs need additional guidance with structured templates, smart interrogations or iterative generation workflows.

Existing LLM-based approaches employ a knowledge-driven generation process that relies on a predefined list of constraint types to guide the LLM in generating domain-specific constraints and rules. For instance, an LLM may be directed to generate only

syntactic check constraints, functional dependencies or integrity constraints defined in a comprehensive knowledge base.

While effective, this paradigm has some limitations: (1) it requires continuous curation and maintenance of comprehensive knowledge bases containing rule type definitions and exemplars, and (2) the rule generation process is inherently constrained to the defined categories, potentially preventing the discovery of novel or unconventional rules. As a result, these approaches improve reliability but trade off flexibility and innovation.

To overcome these issues, the proposed framework introduces a template-driven rule generation approach designed to enhance the broader data curation system **Chat2Data**. The template-driven rule generation process guides the LLMs by using structured templates that enforce different rule shapes (e.g. specifying how many times a certain database relation predicate should appear in a rule's body). As this generation process leads to the generation of many rules, which also include useless and redundant rules, a multi-step evaluation process is employed to evaluate and filter out low quality rules. A novel *semantic-score* is introduced, which aims at measuring the semantic characteristics of a rule. This semantic-score is based on actual rule violations to assess a rule's effectiveness in identifying genuine data errors, which is one of the main goals for data quality checking rules. A LLM-as-a-Judge system then evaluates the domain-appropriateness and practical utility of the generated rules to filter out ineffective rules.

Thus, the motivation for this thesis is as follows:

1. **Scalability:** Enable scalability by reducing the dependence on domain-experts for rule definition.
2. **Reliability:** Address the inherent laziness of LLMs and guide the generation process by providing the LLM with structured templates.
3. **Innovation:** Explore whether or not the template-driven generation process can generate high-quality and semantically rich rules and experiment with novel, AI-based scoring metrics combined with LLM-as-a-Judge systems for rule evaluation.

By automating both rule generation and evaluation, this work aims to demonstrate that LLM-based frameworks can understand dataset semantics to generate useful data quality rules and evaluate them based on multiple criteria.

1.2 Research Questions

1. **How effective are LLMs in generating meaningful and accurate data quality rules from raw datasets?**

The first research question aims at investigating the baseline capability of LLMs to understand dataset semantics and propose rules that are both syntactically valid

and semantically meaningful. While prior research has shown that LLMs are capable of generating structured outputs, their effectiveness in producing meaningful data quality rules remains largely unexplored. The experiments conducted in this thesis aim at providing insights into both the strengths and limitations of LLM-based rule generation, highlighting areas where models excel and where additional guidance or post-processing may be necessary.

2. How does the template-driven rule generation approach compare to existing knowledge-driven generation approaches?

The second research question focuses on evaluating the main contribution of this thesis: the template-driven rule generation process. Unlike knowledge-driven approaches, that rely on predefined lists of rule types and exemplars, the template-driven method guides the LLM by enforcing specific rule structures and shapes. The question examines whether the template-driven method can expand the diversity of generated rules, potentially yielding novel or unconventional constraints, and whether or not it supports the generation of inter-table (multi-table) rules.

3. How reliable are LLM-as-a-Judge systems for evaluating the quality of generated rules compared to traditional, execution-based evaluation metrics?

Traditional rule mining methods evaluate rules primarily using statistical measures like support and confidence. While these metrics capture the statistical validity of the evaluated rules, they lack semantic meaning and thus fail to reflect domain-specific usefulness. The third question thus investigates whether LLM-as-a-Judge systems can serve as a reliable complementary evaluation mechanism.

4. To what extent can LLM-based rule generation and evaluation reduce reliance on domain experts while maintaining rule quality and coverage?

The final question connects the technical contributions of the thesis to a broader practical concern: the heavy reliance on domain experts in current data quality management practices. If LLM-based rule generation and evaluation can produce meaningful rules, organizations may reduce the time and cost required for manual rule generation.

1.3 Thesis Structure

This thesis is structured as follows to systematically explore LLM-based rule generation and evaluation with a focus on a template-driven generation approach.

- **Chapter 1: Introduction:** This chapter provides the foundational motivation for this research by presenting a comprehensive problem statement that highlights the need for automated rule generation and evaluation systems for data quality management. The limitations of more naive and existing approaches to rule generation are discussed to introduce the research questions that guide this work.

- **Chapter 2: Theoretical Background:** This chapter provides the theoretical foundations required for the thesis. It begins with an overview of machine learning paradigms (supervised, unsupervised and reinforcement learning) and introduces the basic concepts of neural networks and natural language processing (NLP). The focus then shifts to Large Language Models (LLMs), covering their architectures, scaling laws, training methods and prompting strategies. To connect LLMs to the application domain, the chapter introduces key concepts from database theory, data quality management and rule-based systems. This combination provides the necessary context to understand both the challenges and opportunities of applying LLMs to data quality tasks such as rule generation.
- **Chapter 3: Related Work:** This chapter gives an overview to relevant literature and state-of-the-art methods. It discusses traditional rule mining approaches and recent work in the field. The chapter then surveys related work that uses LLMs for data quality management. Finally, the chapter explores the emerging role of LLM-as-a-Judge systems, reviewing their strengths, challenges and open research questions.
- **Chapter 4: Framework Design and Implementation:** This chapter presents the design and implementation of the proposed framework, covering the system architecture, technologies used and the complete pipeline from data understanding to rule evaluation. The core contribution of this work, the template-driven rule generation process is discussed in detail, explaining how bag-templates systematically guide LLMs to generate structured rules. The chapter concludes with the two-stage evaluation framework, including the Semantic Score Judge, which classifies rule violations to calculate the novel semantic-score metric, and the Business Domain Judge, which filters out low quality and domain-inappropriate rules.
- **Chapter 5: Results and Discussion:** This chapter evaluates the rule generation process on three datasets that are provided with a corresponding list of golden-rules defined by domain-experts. The semantic-score metric is validated through controlled experiments with synthetic violations. The Business Domain Judge is evaluated with a focus on consistency and reliability. Key limitations are discussed, including generation reliability, computational cost management and judge consistency challenges.
- **Chapter 6: Conclusion and Future Work:** This chapter summarizes the findings of the conducted research. It revisits the research questions, provides evidence-based answers and reflects on the implications for automated data quality applications. Limitations of the current approach are discussed, along with possible solutions and improvements. Finally, the chapter outlines future research directions.

Theoretical Background

This chapter provides the theoretical foundations necessary to understand the concepts and methods used throughout this thesis. It begins with an introduction to the fundamentals of machine learning, including its main paradigms and underlying techniques in Section 2.1. After that, the principles of LLMs, covering their architectures, training strategies and prompting methods are discussed in Section 2.2. Finally, key aspects of database theory and data quality management are discussed in Section 2.3 to establish the link between LLMs and their application to data quality rule generation and evaluation.

2.1 Machine Learning

Machine learning is a subfield of AI that learns to make decision by fitting mathematical models to observed (training) data [110]. One important subcategory of machine learning is deep learning, which uses deep neural to solve more complex tasks. With increasing computational power and major research breakthroughs in the recent decade, these networks have become very powerful and are now the underlying technology to solve tasks in multiple fields including natural language processing, computer vision or even speech recognition [110]. Neural networks are discussed in detail in Subsection 2.1.1.

Machine learning can be roughly separated into three main categories, namely supervised, unsupervised and reinforcement learning. Figure 2.1 gives a high-level overview of how the multiple subcategories of artificial intelligence are related to each other.

The core component in machine learning application is the so-called *model*, which essentially is a **mathematical function** that learns a mapping from inputs to outputs by adjusting its internal parameters during a training phase. What distinguishes different models is how this function is parameterized (e.g. as weights in a neural network or as coefficients in linear regression) and how these parameters are learned from training data.

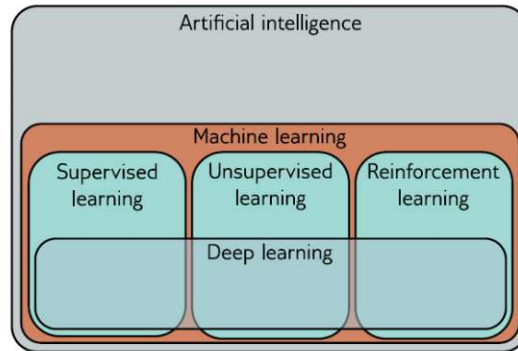


Figure 2.1: Overview of machine learning categories [110]

Supervised Learning In supervised learning, the model is trained on labeled data, where the input data comes paired with the correct output. The goal for the model is to learn a mapping from inputs to outputs, enabling it to make predictions on unseen data. One example use case of this is email spam detection, where the algorithm learns from a dataset of emails labeled either as spam or legitimate. This data is used to train a model, allowing it to categorize new emails into one of the mentioned categories. These types of problems are known as **classification** tasks, where the model has to categorize or label data based on its training data. These tasks come in form of binary classification, where the output can be one of two categories (e.g. positive or negative), and multi-class classification, where the output can be more than two categories [110]. Another problem type is the so-called **regression** task, where the model performs a prediction in form of a continuous number rather than a category or label assignment [110]. An example of regression tasks are house price prediction algorithms [110].

Unsupervised Learning Unsupervised learning on the other hand, works with unlabeled data. The model itself has to find structures or patterns within the dataset without explicit guidance, thus the term *unsupervised*. Instead of learning a mapping from input to output data, the goal is to understand the structure of the data [110]. An example of an unsupervised learning algorithm are generative models like large language models (LLMs). These models can learn the concepts of grammar, semantics and the relationships between abstract concepts without explicit labeling of grammatical rules or other explicit relations. However, it should be noted that modern LLMs are very complex systems that combine supervised, unsupervised and even reinforcement learning strategies [5]. A detailed discussion on how LLMs work is done in Section 2.2.

Reinforcement Learning Another area of machine learning is reinforcement learning. It takes a fundamentally different approach to supervised and unsupervised learning, as the concept of reinforcement learning involves an agent that learns to make decisions by interacting with an environment. These agent actions can change the state of the system, potentially in a non-deterministic way [110]. The goal of reinforcement learning is for the

agent to learn to choose actions that produce the best rewards on average, and thus work towards the desired goal. Some challenges of reinforcement learning include the temporal credit assignment problem, in which a reward may occur delayed or the trade-off between exploration (e.g. to explore new strategies) and exploitation (e.g. work with the strategy it already knows) [110]. An example application of reinforcement learning is a chess playing agent. The agent has to interact with the environment (e.g. the chess field and the opponent), decide on the next move based on the current game state and work towards a goal (winning the game) [127].

2.1.1 Neural Networks

The Perceptron The first notion of an artificial neuron inspired by human biology was introduced by McCulloch and Pitts back in 1943 [110]. They proposed a mathematical model of a neural network consisting of multiple neurons that can potentially work together to perform computational tasks [91]. However, the model did not have a practical learning algorithm, resulting in limited practical use.

Following this groundwork, Rosenblatt [117] proposed the **Perceptron**, a probabilistic approach to information processing based on human brain neurons in 1958. He introduced a probabilistic learning algorithm which automatically adjusted the neurons connection weights depending on the training data. This was the first practical implementation of a supervised learning algorithm based on an artificial neural network.

Perceptrons are the building blocks of neural networks. They take inputs and multiply them by weights, sum them up and apply an activation function. The activation function produces a binary output, which enables the possibility to tackle linear classification tasks. Common activation functions include *ReLU* (handles vanishing gradients), *Sigmoid* (binary classification) and *Softmax* (multi-class classification). Table 2.1 summarizes the key properties of these activation functions and provides some insights on the primary use cases.

Table 2.1: Comparison of Common Activation Functions

Function	Definition	Primary Use Cases
ReLU	$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$	Hidden layers in deep networks; Prevents vanishing gradient problem
Sigmoid	$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$	Binary classification; Output layer for binary problems
Softmax	$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$	Multi-class classification; Output layer for probability distribution

From Perceptrons to Neural Networks

In 1969, Minsky and Papert [96] published their book *Perceptrons: An Introduction to Computational Geometry*, in which they discussed the limitations of the (single-layer)

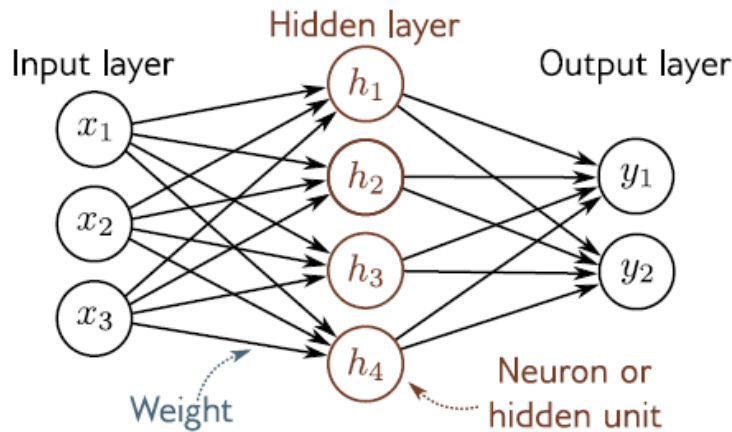


Figure 2.2: Figure of a Neural network consisting of an input, hidden and output layer by Prince [110]

perceptron model. They mathematically proved that single-layer perceptrons were unable to solve non-linearly separable problems (e.g. the XOR function). However, Minsky and Papert saw a potential in multi-layer networks with hidden layers that use non-linear activation functions. These multi-layered networks could theoretically learn more general input-output relations [110], but the authors were skeptical about implementing effective learning algorithms at that time, leading to a significant decline in neural network research during the 1970s known as the "AI winter".

Multi-Layer Perceptrons It was not until the 1980s when researchers like Hinton, LeCun and Parker revived interest in neural networks by introducing the multi-layer perceptron (MLP) and the backpropagation algorithm, which was first discussed by Werbos back in 1974 [135]. Connecting multiple MLPs in a network like structure results in a so-called "neural network". An example of a neural network consisting of an input layer, a hidden layer and an output layer can be seen in Figure 2.2.

Backpropagation Backpropagation is a supervised learning algorithm proposed by Rumelhart, Hinton and Williams [119]. It enables neural networks to iteratively adjust internal weights by minimizing a loss function using a gradient-based optimization algorithm. The backpropagation algorithm efficiently uses the chain rule of calculus to compute gradients of the error for each weight and propagating these gradients backwards through the network layers [119]. Backpropagation solved the **credit assignment problem**, which was the challenge of determining how much each weight in a neural network contributed to the final error in the output. This was a key achievement in the development of efficient learning algorithms that could also be used in larger (deeper) networks.

Gradient Descent A key part of the backpropagation algorithm is the optimization algorithm gradient descent. The algorithm iteratively tries to find the minimum of a function (in case of neural networks, the loss function) by taking small steps in the direction of the negative gradient [49]. The **learning rate** can be adjusted and impacts how large each step should be. The learning rate is crucial, as steps too large might overshoot the minimum and steps too small make the process slower.

To get a better intuitive understanding of gradient descent, imagine a scenario of trying to walk down a mountain in deep fog. The gradient descent approach would be to simply take small steps downhill until no more downhill steps can be found. A major problem that gradient descent algorithms have is that they can run into local minimum or saddle points [49]. Variations of gradient descent implementations include the following:

- **Full-batch gradient descent** is the simplest form of gradient descent that uses the entire dataset to compute the gradient. This results in good convergence performance but is computational intensive on larger datasets.
- **Stochastic gradient descent (SGD)** uses one training example per iteration, resulting in faster updates and local minima escapes through noise. LeCun et al. showed that SGD generalizes better than full-batch GD [81] but has a high variance in updates due to noisy gradients. A nice overview of how SGD is applied in large-scale machine learning can be seen in Bottou [17].
- **Mini-batch gradient descent** uses a small, randomized batch of data to balance between speed and stability. It is often used as a middle ground between full-batch GD and SGD.

There are many other optimization algorithms used in practice today, one common approach is the **Adaptive moment estimation** (Adam) optimizer [72]. For high precision tasks like fine-tuning, SGD variations incorporating a **momentum** factor that introduces a velocity variable to speed up learning are often used in practice [49]. For more details on gradient descent algorithms, see Ruder [118]. Fine-tuning techniques especially for language models are discussed in detail in Section 2.2.2.

Deep Learning

Deep learning refers to neural networks with multiple hidden layers, which are called **deep neural networks (DNNs)**. These networks can learn complex relationships between input and outputs and hierarchical representations of data [79]. Compared to traditional neural networks that usually contain one or two hidden layers, deep neural networks can consist of many more layers. The additional depth allows the network to learn increasingly abstract features from the data, where each layer builds upon representations learned in previous layers [49]. The resurgence of neural networks, particularly in the field of deep learning, was enabled by three key factors - the availability of large labeled datasets, increased computational power through modern GPU architectures and algorithmic innovations [74].

Convolutional Neural Networks (CNNs) Convolutional Neural Networks represent a specialized architecture designed to process grid-like data such as images. The architecture was popularized by LeCun et al. [80] in a system called *LeNet*, which was built to classify 28x28 grayscale images of handwritten digits. This laid the groundwork for modern CNNs, which had a resurgence in popularity by the introduction of *AlexNet* [74] in 2012. The proposed network was a major breakthrough in CNN architectures as it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [120] by a large margin. ImageNet was a dataset published in 2009 containing over 12 million high-scale images labeled into 22,000 categories. The ILSVRC drove further research progress particularly in the field of neural networks and computer vision for several years after the introduction of AlexNet.

The typical architecture of a CNN consists of alternating convolutional and pooling layers, followed by one or more fully connected layers [110]. Each layer has a different purpose, some of the most important ones are explained below.

- **Convolutional layers** apply filters (kernels) that can detect spatial patterns. Each filter extracts specific features such as edges, textures or shapes. With each new convolutional layer, the network can detect more abstract representations, eventually identifying concepts like digits, dogs or other objects.
- **Pooling layers** reduce spatial dimensions through operations like max pooling or average pooling. This reduces computational complexity while still maintaining important features to pass through the network.
- **Fully connected layers** are typically placed at the end of the network to integrate extracted features for the final output generation.

The main application areas for CNNs are image classification, object detection, object recognition and many other computer vision tasks.

Recurrent Neural Networks (RNNs) Recurrent Neural Networks are designed to process sequential data by maintaining a hidden state that is updated at each time step [38]. This allows RNNs to capture temporal dependencies in the input data, making them especially useful in applications such as natural language processing, speech recognition or time-series analysis. However, RNNs had some major problems that were limiting practical use. One of the key problems of RNNs is that they can forget information further back in the sequence, which is known as the *vanishing gradient problem* [60]. The sequential nature of RNNs also had negative implications on computational efficiency (especially for long sequences) and it also limited parallel processing capabilities.

These limitations lead to the development of specialized architectures such as **Long Short-Term Memory (LSTM)** [61] or **Gated Recurrent Units (GRU)** [29], which partially addressed these problems. However, recurrent structures were ultimately replaced with the introduction of the transformer architecture, which is discussed in detail in Section 2.2.1.

2.1.2 Natural Language Processing

Natural Language Processing (NLP) or sometimes called computational linguistics is a field in computer science that involves the understanding, interpretation and generation of human-like text. Eisenstein describes NLP as the design and analysis of computational algorithms and representations for processing natural human language [37]. NLP tasks come in multiple forms, some examples include extracting information from text, translating texts from one language to another, question-answering, spell checking and correction or information summarization [24] [37]. In the rapidly evolving field of generative AI, NLP has risen to unprecedented importance, as the emerging LLMs turned out to be very good in processing and generating text, introducing a new form of powerful NLP applications.

Historical Context and Evolution Early NLP systems of the 1950s and 1960s relied heavily on handcrafted rules and limited vocabularies. One example of an early NLP system was *ELIZA*, a computer program designed to simulate a Rogerian psychotherapist by using pattern matching and substitution techniques to formulate responses to user inputs [12]. It basically was one of the first reliant implementations of a chat bot at that time.

With the statistical revolution of the 1990s, probabilistic models and machine learning approaches took over, as they were able to learn patterns directly from data rather than following explicit rules. This period saw the rise of techniques like **Hidden Markov Models (HMMs)** [113] or **Statistical Machine Translation (SMT)** [101] systems.

The neural revolution and the introduction of deep learning architectures transformed the NLP field further. With the implementation of word embeddings like *Word2Vec* [30] or *GloVe* [106] in the early 2010s, a new way of capturing semantic relationships between words by representing them in a higher dimensional vector space laid the groundwork for today's language models.

Text Embeddings

One of the major breakthroughs that enabled more sophisticated language processing architectures was the introduction of so-called **embedding models**. These models capture semantic relationships between words in a continuous vector space, enabling a deeper understanding and the possibility of similarity searches based on the semantic meaning of words. The basic idea is that words with similar meanings cluster together in the embedding space. Relationships between words can then be captured by applying vector arithmetic, enabling the possibility of synonym detection, analogy solving and semantic search. A visualization of how the embedding space can cluster similar words can be seen in Figure 2.3.

Some of the major architectures are explained briefly below.

(LSA) [35] and prediction-based methods like Word2Vec. The key innovation of GloVe is that it directly trains on global word co-occurrence statistics while also preserving linear relationships usually found in prediction-based models.

Since the first introductions of the above embedding models, the landscape of modern embedding models evolved significantly. The embedding models used nowadays are often pre-trained models based on transformer architectures (see 2.2.1) like ELMo [108], GPT [114] or BERT [36]. These LLMs have the capability to also generate representations for out-of-vocabulary words, resulting in higher performance compared to traditional embedding models [48].

2.2 Large Language Models

This section covers the theoretical fundamentals of how Large Language Models (LLMs) work. First, an introduction to LLMs is given with the historical context leading up to this technology. Important foundational concepts like the transformer architecture and the scaling laws that enabled breakthroughs in the field are discussed. Subsection 2.2.2 covers the training steps involved in building modern LLMs. Inference, prompting strategies and techniques to increase reasoning capabilities are covered in subsection 2.2.3. A paradigm for using LLMs as evaluators, called *LLM-as-a-Judge*, is covered in Subsection 2.2.4. Finally, biases, limitations and ethical considerations are covered in Subsection 2.2.5.

2.2.1 Introduction to Large Language Models

In recent years, LLMs have demonstrated remarkable capabilities in natural language processing tasks. These models have shown outstanding performance in understanding, generating and manipulating human language, allowing for a wide range of applications such as machine translation, chatbots, sentiment analysis or text summarization [86]. This subsection gives an overview of the historical context, important foundational architectures and the scaling laws that enabled this technology.

Historical Context Following the neural revolution and the developments of efficient word embeddings, some advancements have been made in Sequence-to-Sequence learning using RNNs and LSTMs in advanced tasks like language translation [126]. One of the first major works focusing on an attention mechanism to relevant context dynamically was proposed by Bahdanau et al. [9]. Following this, Vaswani et al. proposed the transformer architecture in their famous paper **Attention Is All You Need** [131]. It replaced recurrent network layers with a self-attention mechanism and positional encoding, enabling parallel training and long-range contextual captures [131]. This new type of architecture turned out to become the backbone of modern LLMs, as it is exceptionally well-suited for scaling up models and training data size to increase overall performance [86]. Earlier attempts at **pre-trained language models (PLMs)** included ELMo [109], which was built on a bidirectional LSTM architecture. However, with the release of BERT

[36], a transformer based pre-trained language model that also incorporated fine-tuning methods marked a new era of pre-trained models for natural language processing tasks. The GPT Series (GPT-1, GPT-2, GPT-3) developed by OpenAI demonstrated the power of scaling up auto-regressive models. In 2020, GPT-3 marked a break-through and demonstrated remarkable few-shot performance through scaling up transformer based language models [18].

The Transformer Architecture

The transformer architecture, first published by Vaswani et al. in 2017 [131], marks a significant departure from previous used sequence-to-sequence models that heavily relied on recurrent neural networks or convolutional neural networks. The key part of the transformer is its reliance on the self-attention mechanism, which allows the model to weigh the importance of different words in a sequence in relation to each other. In other words, it can measure the "attention" needed in order to understand the semantic meaning of the words in a sequence. Due to its suitability for parallel computing and the complexity of the model itself, the transformer architecture outperformed previous neural networks in terms of accuracy and performance [86]. The most important techniques and concepts used in transformer architectures are discussed below.

Tokenization To transform sequences of text into data suitable for LLMs, the data is split into non-decomposing units called tokens [99]. Tokens can be single characters, subwords, symbols or words. Tokenization converts variable-length text into discrete units (tokens) and creates a manageable vocabulary size that is computationally efficient while still providing enough deepness and complexity of the used language. Common techniques used in practice are **byte-pair encoding (BPE)** [123], **WordPiece** [136] or **SentencePiece** [75]. A detailed survey on tokenization strategies and how they emerged historically can be found in [94].

Self-Attention The attention mechanism aims at focusing on important information from a large amount of data while ignoring the majority of unimportant information [86]. The self-attention mechanism enables each element in a sequence to interact with every other element to help the model understand contextual relationships in the sequence. It does so by utilizing three components, namely the *query*, the *key* and the *value*. The query represents what information a word is looking for. The key represents what information a word can provide to other words in the sequence and the value contains the actual information to be passed over. The mathematical representation of the attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (2.1)$$

Where Q , K and V are representing the query, key and value matrices and d_k the dimension. Vaswani et al. [131] described the function as mapping a query and a set of

key-value pairs to an output, which is computed as a weighted sum of the values, where the weight of each value is computed by a compatibility function of the query with the corresponding key.

Multi-Head Attention An extension of the self-attention mechanism is the multi-head attention, which performs the attention mechanism multiple times in parallel [86]. This allows the model to capture different aspects of the input by looking at the data from different "perspectives". The original work by Vaswani et al. proposed a model with 8 attention heads, each having reduced dimensions, resulting in a more manageable computational cost [131]. The outputs of the heads are concatenated and linearly transformed to obtain a final representation of values [86]. Multi-head attention is defined as follows [131]:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (2.2)$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.3)$$

Positional Encoding As the tokens pass through the transformer encoder-decoder components, the data lacks any sense of position or ordering in the sequence [131]. To incorporate sequential information into the model a procedure known as positional encoding is applied to the input embeddings at the end of the encoder and decoder stacks [131]. The positional embeddings add a unique vector to each token position in the input sequence, allowing the model to differentiate between tokens based on their position in the sequence [86]. Positional encodings as described by Vaswani et al. [131] are defined as follows:

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (2.4)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (2.5)$$

Architecture Types

The original transformer architecture proposed by Vaswani et al. has since evolved into many distinct variations. The most important architecture types are discussed shortly below.

Encoder-Decoder models like T5 [115] or BART [82] combine both encoder and decoder components. The encoder processes the input and the decoder generates output. This makes the mixed architecture ideal for tasks like translation, summarization or other sequence-to-sequence tasks. The initial transformer model was built using an encoder-decoder architecture and can be seen in Figure 2.4. The illustration shows how the encoder applies layers of self-attention and feed-forward networks, while the decoder adds masked self-attention and cross-attention to the encoder outputs. A final linear layer and softmax generate the output probabilities of the model.

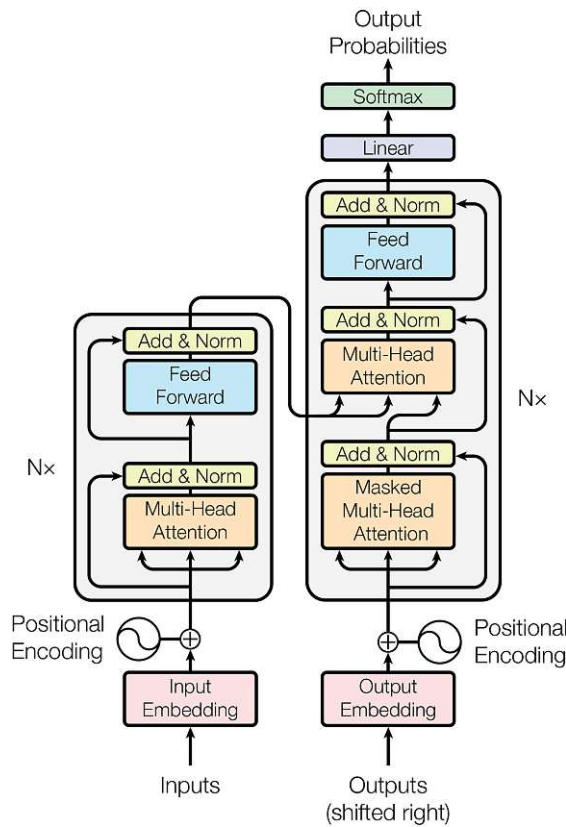


Figure 2.4: The Transformer Model (Encoder-Decoder) as proposed by Vaswani et al. [131]

Encoder-only models like BERT [36] or RoBERTa [87] focus on understanding the input text. Input sequences are processed bidirectionally, allowing the models to excel at classification and text interpretation tasks. While these types of models had their use cases in the past, modern LLMs are usually not built on encoder-only architectures.

Decoder-only models like the GPT series [18], [5] or LLaMA [129] solely focus on the decoding process. The models sequentially generate tokens only attending to preceding tokens in the sequence [86]. This autoregressive setup enables them to model the probability distribution over text sequences, making them highly effective at capturing fluency, coherence and long-range dependencies in generation tasks.

Scaling Laws and Recent Trends

Scaling laws are a foundational concept in understanding the relationship between model size, dataset size and model performance. Kaplan et al. [69] identified robust power laws that explain the relationship between model performance and three key factors - **model size** (e.g. number of parameters), **dataset size** and **computational budget**

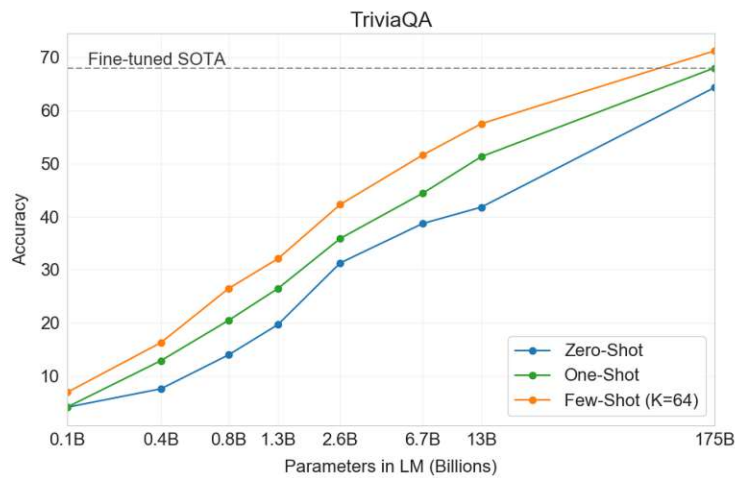


Figure 2.5: GPT-3 performance scaling on TriviaQA by Brown et al. [18]

used for training. Their work demonstrated that language model performance improves predictably by scaling up to larger models. The work was crucial to the field and justified the race towards more powerful and continuously growing models (focusing mostly on scaling up model parameters).

The release of GPT-3 [18] demonstrated that remarkable performance gains can be achieved primarily through scale. An example of the relation between model size and accuracy on a question-answer benchmark can be seen in Figure 2.5.

Following this work, Hoffmann et al. [62] introduced **compute-optimal** scaling laws, which essentially showed that the amount of training data also has to increase if model size is scaled up. How architectural choices influence scaling properties is discussed by Tay et al. [128]. The authors demonstrated that different architectures (e.g. Transformers vs. LSTMs) have different scaling curves and that some architectures benefit more efficiently from increased scale.

Wei et al. [133] observe **emergent abilities** appearing at certain scaling thresholds, challenging the idea of simple power scaling laws introduced prior. The authors define emergent abilities as *"an ability is emergent if it is not present in smaller models but it is present in larger models"*. This phenomena suggests that scaling laws might be more complex with discrete jumps in functionality occurring at critical scaling points.

Recent Challenges to Scaling Laws The introduction of post-training techniques like instruction tuning and reinforcement learning from human feedback (RLHF) [104] (which is discussed in Subsection 2.2.2) showed that relatively small amounts of high-quality human feedback can drastically improve model capabilities without increasing model size. These findings suggest that only scaling up training data and parameters might have diminishing returns and need to be supplemented with high-quality training

data. This is further supported by findings of Longpre et al. [89] showing that training on carefully selected, high-quality data achieves better results than simply scaling up data volume. Sorscher et al. [125] challenged the power law scaling through data pruning strategies, where they selectively removed low-quality or redundant examples to increase performance compared to the full dataset.

The Shift to Mixture-of-Experts (MoE) Architectures A significant development challenging traditional scaling laws has been the adoption of Mixture-of-Experts (MoE) architectures [44] in models like *Mixtral* [68] or *DeepSeek-R1* [53]. These architectures employ sparsely activated modules where only a subset of parameters is activated for a given input. This approach significantly reduces computational resources during inference even for very large models, creating a potential paradigm shift to the classical parameter-compute relationship.

Exhaustion of Web-Scale Data In February 2025, OpenAI released their biggest model yet, GPT 4.5. Compared to a prior reasoning model o3-mini, GPT-4.5 was over 100x more computationally expensive. For reference, GPT-1 (2018) had 117 million parameters, GPT-2 (2019) had 1.5 billion parameters and GPT-3 had 175 billion parameters. GPT-4 (2023) was estimated to have over 1 trillion parameters with an approximated increase of 5-10x over GPT-3. While detailed specifications about the latest models remain proprietary, GPT-4.5 is estimated to contain between 5-10 trillion parameters. Recent benchmarks indicate that GPT-4.5 achieves further improvements in factual reliability and response quality. However, its substantial computational requirements make it prohibitively expensive, and its cost-effectiveness remains to be determined.

During a talk at NeurIPS 2024 [116], Ilya Sutskever, an OpenAI co-founder, said that *"Pre-training as we know it will unquestionably end. We've achieved peak data and there will be no more. We have to deal with the data that we have. There's only one internet"*. This statement supports earlier predictions from researchers like Hoffman et al. [62], who anticipated that optimal model sizes would eventually be limited by data availability rather than computing power.

2.2.2 Model Training

The key part of any machine learning model is the training phase. This section explores the training process used by modern LLMs. It begins with pre-training, which is typically based on self-supervised learning over large-scale datasets. This is followed by post-training techniques such as supervised fine-tuning (SFT) and reinforcement learning techniques that further align model outputs. Finally, test-time scaling methods are discussed as a way to enhance model performance during inference without altering model weights.

Pre-training

The first step in training a language model is a self-supervised learning step, in which the model processes a vast amount of high quality text data to capture information related to vocabulary, grammar, semantics, and text structure [86].

Training data The datasets in pre-training include books, open source software repositories from GitHub [1] and other internet sources like Wikipedia [3] or Reddit [2]. Commonly used sources for training on books are BookCorpus [149] or the Gutenberg [111] data set. CommonCrawl [32] is often used to train on data crawled from the internet. It contains over 250 billion web pages that have been accumulated over a span of 16 years [86]. However, the dataset also contains a lot of redundant and low-quality data, thus some model providers use pre-processed or filtered sources that are based on CommonCrawl, like C4 [115] or CC-Stories [130].

Self-Supervised Learning At its core, there are two main techniques used in the self-supervised learning process. **Next token prediction** [18] is the task of predicting the next token (or word) in a sequence, allowing the model to learn language grammar, facts and reasoning patterns. Another technique mainly used in BERT type architectures is **masked language modeling** [36], where the model has to predict randomly masked words in sequences, forcing the model to understand bidirectional token context.

Pre-training is one of the most expensive tasks when training a language model. It requires a vast amount of computing power and the process takes a long time. Thus, pre-training is usually done only once per model. This also introduces a main limitation of the standalone models, the so-called **knowledge cut-off**. This is the date on which the dataset for pre-training has been collected. Without additional techniques like retrieval-augmented generation (see Subsection 2.2.3), re-training or fine-tuning, models only have access to data until a specific date.

Post-training techniques

After pre-training is done, the model usually has a good baseline for general language understanding (e.g. grammar, text structures, ..) and a compressed form of global knowledge (e.g. facts). However, the model is still "only" predicting next tokens and its ability to solve tasks or act as a helpful assistant is quite limited. To transform this limited model into a helpful assistant or a domain expert, post-training techniques like **fine-tuning** or **reinforcement learning** are applied.

Supervised Fine-Tuning (SFT) Supervised fine-tuning adapts pre-trained models for specific tasks or domains by updating the model weights based on specifically crafted datasets [76]. This process can increase model performance in areas like sentiment analysis, question-answering or in domain-specific tasks. Potential downsides of fine-tuning techniques are the inherent risks of overfitting and required computational cost during fine-tuning. Tuned models have also been shown to struggle with out-of-domain

generalization, hinting at a potential trade-off between specificity and versatility [76]. Some variations of SFT include the following [76]:

- **Instruction fine-tuning** involves training the model on collected pairs of instructions (prompts) and their corresponding responses (completions). This can help the model to learn to follow user instructions and act in an assistant style. Example tasks include summarization, question-answering or creative writing [76].
- **Dialogue / Multi-turn fine-tuning** can increase the model performance in longer (multi-turn) conversations. The data for this type of fine-tuning consists of chat transcripts with multiple user queries and system responses [76].
- **Domain-specific fine-tuning** is necessary when the model needs to excel in a specific domain (e.g. legal, finance or medical). A dataset consisting of domain relevant information and demonstrations of potential tasks is required to fine-tune the model. It has been shown that this technique can achieve remarkable performance in various areas, including legal services [141] [22] or medical applications [65] [84] [124].
- **Chain-of-Thought (CoT) reasoning fine-tuning** is used to teach models how to produce step-by-step reasoning traces instead of final outputs [76]. A more detailed discussion on CoT reasoning can be seen in Section 2.2.3.
- **Alignment or Preference fine-tuning** is used to align models towards desirable or safer outputs. Alignment strategies are discussed in detail in Section 2.2.5.
- **Parameter-efficient fine-tuning (PEFT)** techniques reduce the computational overhead of classical fine-tuning by focusing on a subset of trainable parameters. **Low-Rank Adaptation (LoRA)** [64] achieves task-specific adaptations by updating only specific parameters, significantly reducing computational cost and memory requirements [76]. **Prefix-tuning** adapts models by learning task-specific prefix embeddings prepended to input tokens while keeping model parameters frozen [85]. It is now primarily used for controlling output style and format. **Adapters** similarly freeze original parameters but add small, task-specific neural network layers into the architecture [63], though they typically require more parameter updates than LoRA for comparable performance [57].

Reinforcement Learning from Human Feedback (RLHF) The most common form of reinforcement learning is reinforcement learning from human feedback [104]. RLHF tunes language models by incorporating human preferences, making them more aligned with human expectations. Kumar et al. [76] describe the process in three main steps. In the first step, SFT is applied to a pre-trained model to establish a baseline of linguistic and factual capabilities. Then, a reward function is trained using human rankings of the generated responses. In the last step, a policy optimization algorithm is applied to guide the policy updates based on the reward function.

The technique was successfully used in OpenAI’s InstructGPT [104]. The authors used a dataset of human-written demonstrations of desired prompt responses to fine-tune an instance of GPT-3. With the help of human evaluators, they created a dataset containing multiple model outputs ranked on their quality. This data was used to train a reward model (RM) that could predict which responses humans usually prefer. In a final step, they used their **proximal policy optimization (PPO)** [122] algorithm to further fine-tune the model by rewarding the human preferred outputs. Evaluations showed improvements in truthfulness and preferred outputs in human evaluation while also reducing model size significantly [104].

Reinforcement Learning from AI Feedback (RLAIF) An alternative to RLHF that does not need human labeled data is reinforcement learning from AI feedback. In this technique, a second language model is used to generate preference labels. Just like in RLHF, the generated dataset is then used to train a reward model which guides the reinforcement learning of the target model [76]. In contrast to RLHF, RLAIF reduces the cost and time needed to prepare a human labeled dataset, which enables large-scale model alignment. However, the model that does the labeling needs to be comparable to human labeler’s to achieve good results.

2.2.3 Inference and Prompting Strategies

LLMs can also be enhanced with techniques that do not strictly fall into pre-training and post-training categories. These techniques can enhance model performance without updating the model itself. Enhancements that take place during inference are called **test-time scaling (TTS)** methods [76] (sometimes called inference-time scaling). The most important techniques are discussed below.

Prompting Strategies

The term prompting is quickly evolving due to the rapidly developing field. In general, prompting is the process of providing a prompt to a generative AI model, which then generates a response based on that prompt [121]. There are many different types of prompting strategies, some of which are already built into modern LLMs. Prompting strategies that provide the LLM with instructions or exemplars to solve a certain task in context are also-called **In-Context Learning (ICL)** strategies [121].

Few-shot prompting vs. Zero-shot prompting *Few-shot prompting* is a prompting technique where the LLM learns how to solve a task by including some examples in the prompt. These prompts contain exemplars that demonstrate how to approach a specific task, showing the desired format or reasoning pattern needed for the solution. Brown et al. [18] demonstrated how powerful few-shot prompting can already be with just a few examples. However, the selection of the exemplars is a difficult task that can vastly impact performance. Schulhoff et al. [121] discuss multiple design decisions when selecting exemplars for few-shot prompting. These include exemplar quality (where more examples

generally improve performance), exemplar ordering (which can dramatically influence performance on some tasks) and exemplar label distribution, where imbalanced class representation can bias the model outputs. The authors also emphasize the importance of label quality, format consistency and exemplar similarity to the test samples.

Zero-shot prompting on the other hand involves giving the model a task with no examples, thus relying entirely on its pre-trained knowledge. This strategy is often used in simple tasks or question-answer settings that do not require very specific instructions. Both zero- and few-shot prompting can be combined with other, more advanced prompting techniques discussed below.

Template-based prompting Prompts are often constructed dynamically by using so-called prompt-templates [121]. A prompt template is a pre-defined structure or function containing one or more variables that will be replaced dynamically to create the final prompt. An example of a simple prompt for a binary classification task can be seen in listing 2.1.

Listing 2.1: Binary Classification Prompt Template

```
Classify the tweet as positive or negative:  
{TWEET}
```

Template-based prompting will be one of the core prompting strategies used in the implementation of the rule generation framework.

Role-based prompting Assigning a model a specific persona or expertise (e.g. "You are an expert data quality analyst, ..") to leverage domain-specific knowledge in the training data is called role-based prompting. This technique can also be used to set the tone or style in which the model should response. Recent work has shown that role-based prompting can increase model performance on certain domains, however, choosing the correct role is a challenging task and highly depends on the use case [147]. The prompts used in the framework implementation also utilize role-based prompting techniques.

Reasoning Improvements

Reasoning models are language models designed specifically to engage in a step-by-step "thinking" process before generating answers. Unlike traditional models, reasoning models use an internal chain-of-thought to guide the model through the task. This also means that inference takes longer and the model does not instantly provide an output. These models perform better on tasks that require a deeper "thinking" process like logical reasoning, mathematical problem solving or coding tasks. Some of the most prominent techniques used in improving a models reasoning capabilities are listed below. A visual overview of the strategies discussed can be seen in Figure 2.6. The figure compares different prompting strategies: Direct prompting, Chain-of-Thought (CoT), Self-Consistency, Tree-of-Thought (ToT) and Graph-of-Thought (GoT). Each method

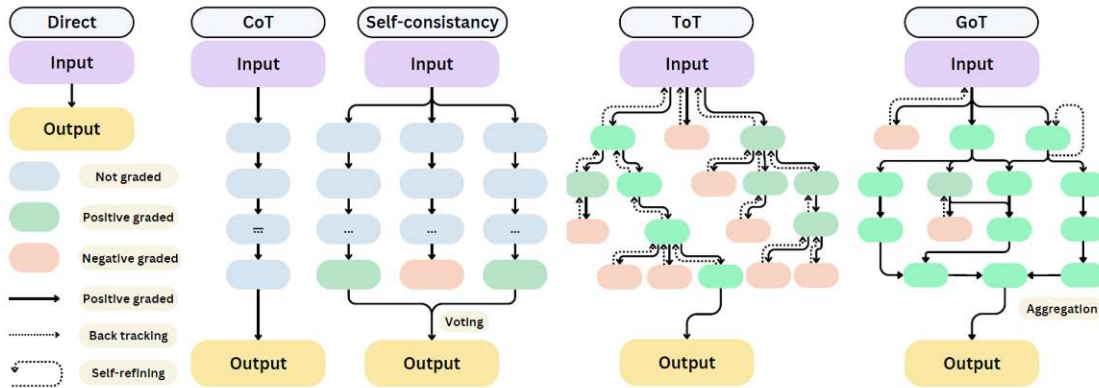


Figure 2.6: Overview of reasoning strategies based on Kumar et al. [76]

introduces different strategies for structuring, exploring or aggregating intermediate results to improve reasoning quality or reliability.

Chain-of-Thought (CoT) prompting A simple, yet effective technique of increasing reasoning performance is the so-called chain-of-thought prompting. The idea is quite simple - provide the model with examples of how to work through tasks step-by-step. The examples can then be either used in-context through prompting strategies or in fine-tuning steps during post-training. The technique was originally proposed by Wei et al. [134] and showed that breaking down tasks into smaller, intermediate reasoning steps can increase reasoning performance. The authors added demonstrations of chain-of-thought exemplars into the prompt (i.e. in-context) and were able to measure increased performance across a variety of tasks, including arithmetic, commonsense and symbolic reasoning. It has also been shown that simply adding phrases like "think step by step" to the prompts can already increase zero-shot performance [73], without the requirement of specific exemplars.

Tree-of-Thoughts (ToT) The Tree-of-Thoughts framework [138] is an extension of chain-of-thought that enhances model performance by exploring multiple reasoning paths simultaneously. Instead of focusing on a single chain-of-thought, the model can branch out into multiple possible thought sequences [76]. The framework creates a tree structure where each node represents a partial solution (or a "thought"). The model can then consider multiple branches, backtrack when needed and finally select the most promising path. This essentially maps the model reasoning into a tree search task. ToT excels at more complex problems that require multiple steps or strategic explorations like puzzle solving, planning or game simulations [76].

Compared to traditional CoT, ToT requires more computational resources but can provide improved problem-solving abilities, enhanced error recovery and increased adaptability (e.g. different search strategies can be used). However, challenges include the evaluation

and selection of the correct branches and scaling can become problematic for difficult tasks due to the increased computational cost [138].

Another extension of the ToT framework is **Graph-of-Thoughts (GoT)** [13]. The framework extends ToT by utilizing a graph-based structure for the reasoning process rather than a strictly hierarchical tree [76]. This allows for more flexible reasoning paths where new edges can be formed dynamically, thought branches can merge or form cycles.

Self-Consistency Decoding Self-Consistency decoding was proposed by Wang et al. [132] as an alternative decoding strategy to the naive greedy decoding used in regular chain-of-thought prompting. By letting the model sample multiple distinct reasoning paths and then marginalizing over the paths, the strategy showed improvements across a variety of reasoning tasks. The different reasoning chains are created by using prompt engineering and temperature sampling to encourage the creation of different CoTs [76]. After the chains finished and the model outputs a result for each chain, the most consistent result is chosen (essentially a majority vote is taken). The technique is especially useful for arithmetic and commonsense reasoning tasks where multiple paths can lead to a solution [76].

Reasoning: Less is More? Scaling up reasoning capabilities was usually achieved with massive training datasets consisting of complex tasks and their corresponding reasoning chains. Ye et al. [139] challenge this assumption with their **Less is More for Reasoning (LIMO)** hypothesis. The authors show that remarkable mathematical reasoning capabilities can be achieved with only 817 carefully selected high-quality training examples. Their model outperformed other reasoning models that were trained on datasets much larger (> 100.000 examples).

According to their findings, the complex reasoning capabilities of a model are fundamentally determined by two key factors:

- The completeness of the model’s knowledge gathered during pre-training.
- The effectiveness of high-quality post-training examples that serve as cognitive templates to guide the model through complex reasoning tasks [139].

The findings suggest that complex reasoning is more about the correct selection of high-quality examples rather than sheer quantity. LIMO could potentially revolutionize reasoning training and make it more accessible for a broader audience due to the significantly reduced computational and data resources needed. The paper also draws parallels to previous work by Zhou et al. [148], which proposed a **Less is More for Alignment** strategy that showed similar findings for general alignment (especially for response format and style) could be achieved with very few examples.

Retrieval-Augmented Generation (RAG)

Retrieval-augmented generation [83] is a framework used to enhance LLMs by adding the ability to retrieve external knowledge before or during the generation process. It has been shown that RAG can enhance answer accuracy and reduce model hallucinations particularly for knowledge intensive tasks [46]. One of the key limitations that current language models face is that they are trained on potentially outdated or obsolete information. RAG can provide the models with reliable and up-to-date data from an external knowledge base [43].

Core Concept The RAG framework consists of several processes, namely retrieval, generation and augmentation, and a mechanism that determines whether or not the retrieval is needed for the current task [43].

- **Retrieval:** The first step in the RAG pipeline is the retrieval process, which extracts relevant information from an external data source. This is usually done by doing semantic search on a vector database that has been initialized with external data. The retrieved data (usually called "documents") is then used in the next step for augmentation.

There are two main implementations of retrieval, namely *sparse* and *dense* retrievers. **Sparse retrieval** is word-based and mostly used for text retrieval [43]. **Dense retrieval** utilizes vector space embeddings for enhanced (semantic) similarity searches [70]. Dense retrieval methods use embedding models to embed documents and queries into vector space. These embedding models are trainable, which can increase overall performance of the RAG application in certain scenarios [100] [71] [143].

- **Augmentation:** Augmentation is the step that combines the data from the retrieval step with the generation step. Augmentation can be done in essentially three steps: *input-, output- and intermediate-layer integration* [43]. Input-layer augmentation takes the data returned from retrieval and adds it to the prompt or context before calling the generation model. Output-layer augmentation adds the retrieved data to the output of the generation model. Intermediate-layer integrations directly integrate the retrieved results through internal layers of the generation model, thus requiring high access to the generation model.
- **Generation:** Calling the generation model to finish the task is called generation. Just like the embedding model, the generation model can also be fine-tuned or trained to increase performance in the downstream task. Generally speaking, generation can be done using two kinds of generation models, *parameter-accessible (open-weight)* and *parameter-inaccessible (closed-weight)* models [43].

RAG systems can significantly increase performance and accuracy while simultaneously reducing factual hallucinations. RAG is especially useful in tasks where factual, domain-

specific knowledge or up-to-date information is necessary. A common use for RAG is to enhance LLMs with the ability to fetch internet data [98].

Despite the many upsides, RAG has limitations that need to be considered. First of all, fetching data from an external source can increase inference time significantly. It has also been shown that retrieval of irrelevant data can lead to false and misguided responses, even when the inherent knowledge of the model would have been sufficient to answer the question [140]. Determining if retrieval is even necessary, which and how many documents need to be retrieved and how to formulate the final prompt for the generation tasks are open research questions and highly dependent on the task to be solved.

2.2.4 LLM-as-a-Judge

While LLMs are mostly known for their abilities to generate content, they can also be used as evaluators or decision makers for complex tasks. This paradigm is known as LLM-as-a-Judge (LaaJ). With their ability to process diverse data types and provide scalable and cost-effective assessments, LLMs can present an alternative to traditional expert-driven evaluations [51]. The main benefits of LaaJ systems are scalability and explainability. Forcing the LLM to also explain the reason for the decision can provide valuable insights on the task and the models internal knowledge leading to the decision.

Pairwise comparisons The most fundamental paradigm involves presenting the judge with two outputs and asking it to determine which one is better. This mirrors human preference evaluation and tends to be more reliable, as it does not require absolute scoring. The technique is often used for ranking multiple options or prioritizing them. It has been shown that LLMs and human evaluations are more aligned in pairwise comparison settings than in score-based assessments [51]. An example of a simple prompt template for pairwise comparison task can be seen in Listing 2.2.

Listing 2.2: Pairwise comparison prompt template [51]

```
Given a new article, which summary is better? Answer "Summary 0" or "
  Summary 1".
You do not need to explain the reason

Article: {Article}
Summary 0: {Summary_0}
Summary 1: {Summary_1}
```

Binary classification and question-answering Binary classification or yes/no question-answering is another popular technique commonly used in LLM judges. It is often used as a pre-processing step before more complex evaluations are done in order to filter out problematic or clearly low-quality inputs. This concept can also be mapped to other custom phrases beyond simply "yes" and "no" answers. The final step of

the evaluation phase in the proposed framework will utilize a classification judge that approves or discards a rule based on specified criteria.

Output scoring In scoring judges, the model evaluates a given input against specific criteria by using numerical scales (e.g. 1-10) or categorical ratings (good/bad). Scores can either be discrete, with ranges like 1-5, or continuous values ranging from 0.0 to 1.0 [51]. Scoring can also be applied to multiple dimensions, for example, a judge prompt could include dimensions like helpfulness, relevance or confidence of the judgment. The dimensions can then be scored using a similar scale with an overall score based on the assigned dimension scores. An example of such a scoring prompt with multiple dimensions can be seen in Listing 2.3.

Listing 2.3: Scoring prompt based on templates used by Gu et al. [51]

```
Evaluate the quality of summaries written for a news article.
Rate each summary on the four dimensions: factual correctness,
    readability, conciseness and objectivity.
Rate each dimension on a scale from 1 (worst) to 5 (best).
Based on these scores, provide an overall score reflecting your
    judgment.

Article: {Article}
Summary: {Summary}
```

Limitations of LLM-as-a-Judge Although LaaJ systems have many benefits, they also have limitations that need to be considered. One major limitation is that evaluating models that lack adequate knowledge about the domain in question make incorrect judgments. This phenomenon is closely related to the well known hallucination problem. It has also been shown that LLM judges have limited math and reasoning capabilities, which also results in wrong judgments when they are asked to grade these types of tasks. However, Zheng et al. [146] observed that some models fail at correctly grading tasks that they would be able to solve when asked separately, which means that the other provided answers that were indeed wrong somehow mislead the judge.

Bai et al. [11] tackle some of these issues with a concept called **LLM-as-an-Examiner**, which expands on simple evaluation prompts by having the LLM generate additional questions to test if it is fit for the given domain. Additionally, follow-up questions are given to the evaluator to get a more in-depth assessment of the judgment. The authors also discuss another limitation of LaaJ system which is the individual model bias. Certain models can have internal biases that may influence the judgments. To tackle this, multiple (different) models can be used in a peer-examination setting to reduce individual biases.

Zheng et al. [146] observed additional limitations and biases.

- **Position bias** occurs when certain models prefer a specific position in a ranking evaluation setting. The authors observed that *GPT-4* considered one answer better

when first placed, but then favored the other answer when their positions were swapped. Similar results were also observed for other models.

- **Verbosity bias** references to the tendency of LLMs to favor longer and more verbose responses, even when they are not of higher quality. A *repetitive list* attack was performed where responses containing numbered lists were taken and then rephrased to make them unnecessarily verbose [146]. This new, rephrased (and longer) list was then inserted to the initial numbered list. Even though the new list does not contain any new information, the LLM judges preferred the new list over the original, more precise list.
- **Self-enhancement bias** describes the phenomena that LLM judges may favor answers generated by themselves. It has been observed that *GPT-4* favors itself with a 10% increase and *claude-v1* favors itself with a 25% increase [146]. However, due to the small measured differences and the observation that certain models also prefer outputs from other specific models (not just from themselves), it is hard to say to what degree self-enhancement bias is happening.

2.2.5 Ethical Considerations

This section covers some ethical implications and challenges of LLMs and more broadly speaking artificial intelligence as a whole. The potential benefits in areas like healthcare, entertainment, transport, education and many others are obvious. However, there are also risks involved and the potential of harm could be enormous, if the technology is abused by malicious actors or totalitarian regimes.

Liu [86] highlights five major concerns of artificial intelligence:

- **Bias and Fairness:** A major problem of machine learning algorithms that train on specific data is the so-called **bias propagation mechanism**. If a system is trained to predict salaries for individuals based on historical data, then the system will reproduce historical biases [86]. It has also been shown that AI systems can reinforce or empower stereotypes [90] or that LLMs implicitly exhibit user discrimination in recommender tasks [142]. Gabriel et al. [45] argue, that AI systems (and technology in general) are implicitly non-neutral and that alignment with human values is necessary. However, as human values themselves are complex, context-dependent and sometimes difficult to specify, this problem is not easy to solve. The strategy of aligning a model to certain values or a specific behavior is called *Model Alignment*. For a more detailed discussion on fairness including examples of potential biased and unfair systems take a look at [14].
- **Explainability and Lack of Transparency:** With the arising complexity and size of modern deep learning systems, it is hard to know exactly how or based on what information these systems behave [86]. Understanding and interpreting why a LLM answered in a certain way is currently very limited, but necessary to

build reliable and non-harmful systems. This has led to the field of **Explainable Artificial Intelligence (XAI)**, a research field that aims at explaining and increasing transparency in modern AI systems [52] [33].

- **Weaponizing AI:** Unfortunately, violent conflicts seem to be an inevitable feature of human behavior. Many major new technologies have been applied directly or indirectly towards war [86]. Due to the power of AI systems it will be (and already is) used in human conflicts. AI systems have already been shown to be weaponized in multiple areas such as cyber attacks [137], physical warfare (e.g. autonomous drones) [19], disinformation/propaganda [16] and information warfare [66].
- **Concentrating power:** For the last couple of years, the majority of breakthroughs in large scale generative AI applications have been made by a handful of companies (e.g. OpenAI [103], Google [50], Meta [93]). To power these systems, huge amounts of money have been invested [40] [92]. It is clear that the world's most powerful companies do not invest this amount of money out of philanthropical reasons. They know that these technologies will allow them to make enormous profits [86]. If some of the currently used models outperform the majority of the competitors, a significant concentration of power could emerge, enabling the dominant players to consolidate immense influence. Whether or not this will happen is currently unclear. However, the substantial computational resources required to develop and maintain state-of-the-art LLMs already acts as a barrier to entry, potentially limiting competition and reinforcing the dominance of big companies.
- **Existential risk:** Liu [86] claims that many existential risks to human race resulted from technology. For example, climate change has been driven by industrialization, nuclear weapons derived from research in physics, pandemics are more likely to occur due to the globalized world that was built on innovations in transport, agriculture and construction [86]. AI brings a new existential risk, in which we should be very cautious about building systems that are more capable than human beings. Carlsmith [20] examined whether or not AI systems that pursue power pose an existential risk to humanity and estimated a probability of 5% for an existential catastrophe by 2100. He argues that such systems will likely emerge in this century and that they will likely seek power to achieve their objectives, potentially leading to catastrophic outcomes if their goals are not aligned with human values. This work highlights the serious attention we should give towards building aligned models and robust safety protocols for AI development.

These aspects are just a few potential risks that AI could be used for or lead to. There are many more negative impacts AI could have, from mass surveillance, disinformation creation and spreading or the enormous amount of energy consumption required to train AI systems that might contribute to climate change [86].

Safety Alignment

Safety Alignment refers to the challenge of aligning AI systems with human values and intentions. It is an important part of ensuring that system harmfulness is reduced and usability in assistance tasks is increased. Ji et al. [67] characterized the goals of alignment strategies with four principles called the *RICE principles*:

- **Robustness** refers to the overall resilience of AI systems when operating across diverse scenarios or under adversarial pressure. For example, models might seem to be well aligned and refuse harmful requests initially but be prone to jailbreak attacks, a type of attack where the limitations set by alignment can be completely circumvented [150]. In these attacks, prompt injection techniques are used to trick or hijack the model into answering harmful requests [67].
- **Interpretability** is closely connected to transparency and demands that we can understand a model's internal reasoning. To achieve this, alignment has to aim at making AI systems honest, non-concealing and non-manipulative [67].
- **Controllability** ensures that the actions and decision-making processes of a system remain subject to human oversight and intervention [67]. With the growing power and scale of modern models, many researchers express growing concern about whether or not controllability is possible [34] [58]. Controllability is focused on enabling scalable human oversight during the training process and corrigibility (e.g. not resisting shutdown or objective modifications) of AI systems [67].
- **Ethicality** ensures that a system is committed to uphold human values, moral guidelines or social norms and values [67]. To ensure this goal, ethical frameworks for AI systems have been proposed [54].

Reinforcement Learning from Human Feedback One of the most important strategies to increase model alignment is reinforcement learning from human feedback [25]. The technology involves model fine-tuning based on human feedback to increase model performance (see Subsection 2.2.2) and alignment. InstructGPT [104] is a prominent example of successfully using RLHF to increase alignment. An evaluation of the reinforced model InstructGPT (1.3B parameters) against the original GPT-3 (175B parameters) model showed improvements in truthfulness, reductions in harmful output and preferred outputs in a human evaluation [104]. The strategy has become fundamental for the development and alignment of modern language models.

Constitutional AI (CAI) A major downside of RLHF is the need for human evaluators or human labeled data. Creating these datasets is costly and involves human experts and labor. Constitutional AI is an approach developed by Anthropic [8] that substitutes human evaluation with AI-generated feedback based on some core constitutional principles [10]. The process involves two key-phases:

1. Red-Teaming Phase: One system generates potentially harmful outputs to various prompts.
2. Constitutional Phase: Another system then evaluates these outputs against a set of constitutional principles and then revises harmful responses to create safer alternatives.

Similar to RLHF, the created set of safer response alternatives can then be used in a reinforcement learning phase to fine-tune the model and thus increase overall safety alignment. This approach introduced a self-improving behavior and removed the dependency on human labeled data, thus increasing scalability.

2.3 Database Theory Fundamentals

In order to design a framework for automated data quality rule generation, the fundamental concepts of databases, data quality and rule mining have to be discussed. This section gives a short introduction to databases and data quality management in Subsection 2.3.1. Afterwards, the most important constraint types, metrics for evaluating them and traditional methods for rule mining are covered in Subsection 2.3.2. Finally, Subsection 2.3.4 gives a short introduction to logic programming languages like Prolog or Datalog, which will be one of the formalisms used in the rule generation framework.

2.3.1 Introduction to Databases and Data Quality Management

Modern information systems rely heavily on databases to store, organize and manage vast amounts of data. As organizations increasingly depend on data-driven decision making, ensuring data quality and integrity is a crucial task. This subsection introduces important theoretical background on relational database systems including the ACID properties, integrity constraints and gives a short introduction to data quality management.

The relational model

The relational database model has been the foundation of modern data management since it was introduced by Codd [31] in 1970. It formalizes data as a collection of relations (tables), consisting of tuples (rows) over a set of attributes (columns). The model is based on a theoretical foundation using relational algebra and calculus, which enables both data independence and declarative querying through query languages like *SQL* (*Structured Query Language*). SQL is the de-facto standard language for interacting with relational databases, allowing users to *create*, *read*, *update* and *delete* data through a human-readable query language. Due to its strong mathematical foundation, the model ensures predictable behavior and efficient query optimization methods, which lead to the broad adoption of relational database systems (DBMS) [4].

Entity Relationships The relational model relies on so-called **keys** to maintain data integrity and establish relations between tables. A key uniquely identifies each row in a table. A process called *normalization* is used to eliminate data redundancy and ensure consistency between tables. Relationships between tables are defined by referencing other tables through a foreign key.

SQL (Structured Query Language) SQL emerged as the standard declarative query language used in most relational databases. Declarative means that users specify what data they want rather than how to retrieve it. Complex data retrieval or modifications can be done using human-readable statements which the DBMS query optimizer translates into efficient execution plans. SQL is also relevant for data quality, as it allows for defining constraints that restrict data before it can be inserted into the table. An example of a CREATE statement for a users table that also sets some constraints on the attributes is given in Listing 2.4.

Listing 2.4: SQL Constraint Examples

```
1 CREATE TABLE users (  
2     user_id INT PRIMARY KEY,  
3     username VARCHAR(50) NOT NULL,  
4     email VARCHAR(255) NOT NULL,  
5  
6     CONSTRAINT unique_key_username UNIQUE (username),  
7     CONSTRAINT check_username CHECK (LEN(username) BETWEEN 5 AND 10)  
8     CONSTRAINT check_email CHECK (email ~ '^[^@]+@[^@]+\.[^@]+$')  
9 );
```

ACID properties A core concept in many modern relational database management systems are the so-called ACID properties (Atomicity, Consistency, Isolation and Durability). Usually, these properties are implemented by using a concept called *transactions*. Transactions handle one or more database operations in an all-or-nothing principle, meaning it will either apply all of the operations included in a transaction or none of them. This ensures that more complex operations involving multiple steps like transferring money from bank account A to bank account B is processed completely (e.g. deducing money from account A, add money to account B) or rolled back in case of a failure.

Atomicity ensures that transactions are seen as single units of work. Many transactions include multiple operations, but for the database system these operations are seen as one encapsulated operation. The operations in a transaction are either fully committed or completely rolled back in case of a failure. **Consistency** ensures that transactions can never result in an inconsistent state, that is, it will never result in a state where data integrity is violated. **Isolation** prevents concurrent transactions from interfering with each other through various locking and concurrency control mechanisms. **Durability** guarantees that committed changes persist permanently, even if the database system crashes.

Integrity constraints A central aspect of relational databases is the definition of so-called integrity constraints. These constraints are logical conditions that restrict the state of a database to ensure consistency and correctness. Examples include unique keys, which ensure that certain tuples are unique in the dataset, functional dependencies (FDs), which describe that certain attribute values determine other values, inclusion dependencies (INDs), which express foreign-key relationships across different tables and check constraints, which restrict data types or attribute value ranges [4]. While these constraints mainly guide schema design and query optimization in traditional database management systems, they are also relevant for data quality management. Integrity constraints and the different forms and types they can be defined in are covered in Subsection 2.3.2.

Data Quality Management

The concept of data quality extends beyond just the syntactic correctness of data. It refers to the degree of which data is fit for use in a specific context or for a specific user query. Data quality can be considered as one of the dimensions of data governance and high-quality data [77]. Ladley describes data governance (and to that extent also data quality) as a "required business capability if you want to get value from your data". High quality data is sometimes also defined as **data fit for purpose**. As the negative impacts of bad data quality became inherent, data quality management has become an increasingly crucial part of information systems, including tools for data profiling, monitoring and cleansing.

Dimensions of data quality Fan et al. [41] describes five central issues connected to data quality, namely (1) data consistency, (2) data de-duplication, (3) data accuracy, (4) information completeness and (5) data currency.

- **Data consistency** refers to the requirement of data being free from contradictions and inconsistencies. This includes ensuring that the same entity is represented in a uniformed way throughout the database, that referential integrity is maintained and that business rules are consistently applied. Inconsistencies in the data can be represented as violations of data dependencies (e.g. integrity constraints) [41].
- **Data de-duplication** addresses the challenge of identifying and removing duplicated tuples in the database that refer to the same entity. Data de-duplication is required for effective data cleaning, where duplicates need to be removed, and data integration, where information from different sources about the same entity need to be resolved or merged [41]. Effective de-duplication is non-trivial as duplicates can also contain small variations in formatting, spelling or completeness.
- **Data accuracy** refers to the degree of which data correctly represents the real-world entities they are intended to describe. Examples of inaccurate data can come from errors during measurement, wrongly labeled data or outdated information.

Ensuring accuracy often requires validating the data against an external source and implementing quality measures throughout the data life cycle.

- **Information completeness** refers to the extent of which all relevant pieces of data for the described entity are covered. Missing data can occur at the attribute level (e.g. certain values are missing) or at relation level (missing tuples in a table). Traditional work on information completeness usually adopts either the *Closed World Assumption (CWA)* or the *Open World Assumption (OWA)* [4]. The CWA assumes that a database has collected all tuples but values of certain attributes might be missing, while the OWA assumes that in addition to missing values also tuples may be missing [41].
- **Data currency or timeliness** tackles the question whether or not the data properly reflects the current state of the entities being described. As data is changing continuously, data can become outdated and potentially misleading. Adding timestamps to the tuples can theoretically help in data currency, however, in practice timestamps are often unavailable or imprecise [41].

2.3.2 Constraint Types, Learning and Mining

Data quality management relies heavily on understanding and implementing various types of constraints that ensure data consistency, integrity and validity. This subsection explores the fundamental constraint types and formalisms used in modern data quality systems.

Integrity Constraints and Data Dependencies

Data dependencies form the theoretical foundation for understanding relationships between data elements. Integrity constraints are a fundamental concept that describes rules or constraints for ensuring data consistency in applications like database design, data integration and data cleaning.

Functional Dependencies (FDs) Functional dependencies represent the most fundamental type of data dependency. They express when one set of attributes uniquely determines another set of attributes.

A functional dependency is formally defined as $(X \rightarrow A)$, where X is a set of attributes (the determinant) and A is an attribute (or a set of attributes) that is functionally dependent on X [105]. This dependency holds in a relation R if for any two tuples t_1 and t_2 in R , when $t_1[X] = t_2[X]$, then $t_1[A] = t_2[A]$ also holds. Or in other words, if two tuples have identical values for all attributes in X , they must also have identical values for attributes in A .

An example of a functional dependency in an address relation would be the dependency between the *Zipcode* and the *City* attributes, as the *Zipcode* usually determines the *City*

attribute (and vice-versa) [105]. FDs are crucial for database normalization and detecting data inconsistencies.

Conditional Functional Dependencies (CFDs) Conditional FDs extend traditional functional dependencies by incorporating conditions under which the dependency must hold. These conditions allow the dependency to capture more complex and more specific data dependencies.

Fan et al. [42] defines a CFD as a pair $(X \rightarrow A, t_p)$, where $X \rightarrow A$ is a standard functional dependency and t_p is a pattern tuple that specifies conditions on attributes in X and A . CFDs are particularly valuable for data cleaning as they can capture semantic relationships that traditional FDs cannot express.

For example, using the data given in Table 2.2, the functional dependency ($Zipcode \rightarrow City$) would be violated as the *Zipcode* "10001" would match both "New York" and "Manhattan". This functional dependency can be extended with an additional condition on the *Country* attribute to catch that inconsistency. The functional dependency can be extended to the CFD $(Zipcode \rightarrow City, T)$, where $T = (Country, "USA")$. This new (conditional) functional dependency now states that the *Zipcode functionally determines City, given that the country is "USA"*.

Table 2.2: Example data for city and zipcode CFDs

Zipcode	City	Country
10001	New York	USA
90210	Beverly Hills	USA
M3C 0C1	Toronto	Canada
10001	Manhattan	USA

Compared to traditional FD mining, discovering conditional functional dependencies is much more complex as it requires finding both the underlying functional dependency and the associated semantic patterns required for the conditions.

Inclusion Dependencies Inclusion Dependencies are fundamental integrity constraints in relational database theory that specify containment relationships between attribute sets across different relations [4].

An inclusion dependency is formally defined as $R_1[X] \subseteq R_2[Y]$, where R_1 and R_2 are relations and X and Y are attribute sets of the same cardinality. The constraint ensures that every combination of values in attributes X of relation R_1 must also appear in attributes Y of relation R_2 .

An example of an inclusion dependency can be found in an e-commerce database with an *Order* relation containing attributes (*OrderID*, *CustomerID*, *ProductID*) and a *Customer* relation with attributes (*CustomerID*, *Email*, *Address*). The inclusion dependency $Order[CustomerID] \subseteq Customer[CustomerID]$ ensures that every customer ID refer-

enced in an order must exist in the Customer relation, which is essentially a simple foreign-key constraint in relational database terms. These constraints are essential for maintaining referential integrity and are more general than simple foreign key constraints as they can involve multiple attributes. This flexibility makes them particularly useful for modeling more complex real-world relationships where referential integrity involves composite attributes or non-key references.

Tuple-generating Dependencies (TGDs) Tuple-generating dependencies are a generalization of inclusion dependencies that specify when the existence of certain tuples in one relation implies the existence of related tuples in another relation.

A TGD is formally expressed as:

$$\forall \bar{x}(\phi(\bar{x}) \rightarrow \exists \bar{y}\psi(\bar{x}, \bar{y}))$$

where $\phi(\bar{x})$ is a conjunction of relational atoms (the premise) and $\psi(\bar{x}, \bar{y})$ is a conjunction of relational atoms (the conclusion). Thus, a TGD is a dependency in which no equality atoms occur.

An example TGD in our e-commerce database could state that if a customer has placed an order, then there must exist a corresponding shipping record:

$$\forall c, o(\text{Order}(o, c, p) \rightarrow \exists s \text{ShippingRecord}(s, o, c))$$

Equality-generating Dependencies (EGDs) Equality-generating dependencies generalize functional dependencies by allowing more complex conditions where equality must hold. An EGD is a dependency for which the right-hand formula of the implication is a single equality atom [4]. While functional dependencies usually operate withing a single relation, EGDs can span multiple relations and specify equality constraints based on join conditions or other patterns. EGDs are formally described as:

$$\forall \bar{x}(\phi(\bar{x}) \rightarrow x_i = x_j)$$

where $\phi(\bar{x})$ is a conjunction of relational atoms and x_i, x_j are variables from \bar{x} .

Given the the e-commerce database described earlier, an EGD that ensures that customers with the same email must also have the same CustomerID could be:

$$\forall c_1, c_2, e(\text{Customer}(c_1, e, a_1) \wedge \text{Customer}(c_2, e, a_2) \rightarrow c_1 = c_2)$$

Primary and Unique Key Constraints Primary key constraints ensure that each tuple in a relation can be uniquely identified, combining uniqueness and non-null requirements [4]. Unique key constraints provide uniqueness without the non-null restriction. These constraints are essential for entity integrity and duplicate prevention.

Check Constraints Check constraints define domain-specific validation rules that restrict the possible values for attributes based on business logic or semantic requirements. Some examples include range constraints (e.g. $\text{age} > 0$), format constraints (e.g. an email matching a specific pattern) or cross-attribute constraints (e.g. that $\text{StartDate} < \text{EndDate}$). Check constraints are crucial for preventing or finding invalid data entries.

Foreign Key Constraints Foreign key constraints maintain referential integrity by ensuring that values in referencing attributes correspond to existing values in referenced relations. These constraints are essential in preventing orphaned records and ensuring consistency across related entities.

Denial Constraints A special formalism for representing database constraints are so-called denial constraints, which specify conditions that must not hold on a valid database instance. Compared to traditional integrity constraints that usually define what should be true, denial constraints explicitly state forbidden patterns or combinations of invalid data. Denial constraints are described in a universally quantified first order logic formalism, which allows them to be more expressive than regular integrity constraints like functional dependencies [26].

A *denial constraint* is a logical formula of the form

$$\forall x_1, x_2, \dots, x_n : \neg\phi(x_1, x_2, \dots, x_n)$$

where ϕ is a conjunction of atoms consisting of relational predicates and comparison operations. The constraint is *violated* if and only if there exists a binding of the variables x_1, x_2, \dots, x_n to database tuples such that $\phi(x_1, x_2, \dots, x_n)$ evaluates to true.

Chu et al. [26] classify the different types of integrity constraints based on two criteria, namely (1) single tuple vs. table level and (2) with constants involved vs. with only column variables. The authors conclude that denial constraints are expressive enough to cover all of the classified integrity constraints. However, the authors note that the increased expressive power of denial constraints also makes it harder to automatically obtain them, as the search space is larger. Thus, generating useful denial constraints is often reliant on domain experts, which makes the process expensive and time consuming.

2.3.3 Association Rule Mining

Association Rule Mining (ARM) is one of the most important and well researched areas of data mining. It is a fundamental unsupervised learning technique that is used to discover meaningful patterns, correlations and relationships in large datasets. An association rule is typically expressed as an implication of the form $(X \Rightarrow Y)$, where $X, Y \subset I$ are sets of items called *itemsets* [145]. X is called the *antecedent* (or left-hand side) and Y *consequent* (or right-hand side) of the rule. The rule essentially expresses that the presence of X implies the (likely) presence of Y . These rules are evaluated using statistical measures such as support and confidence, which will be described later in Subsection 2.3.3.

A classical example of how association rules can be used in business applications is the "customers who bought this also bought" recommendation sections found in online stores. These recommendations are derived by identifying the underlying patterns in customer purchasing behavior. For example, collected data about recent orders shows that specific books are frequently purchased together. This insight can be used to personalize recommendations for customers browsing one of those books. Association rules can also capture domain-specific dependencies, ranging from functional dependencies in databases to integrity constraints or more complex dependencies like temporal dependencies.

While statistical methods can yield many rules, it is not guaranteed that all of the statistically valid rules provide actionable or useful insights. Finding out whether or not a given rule, pattern or constraint is of interest is not trivial. Han et al. [55] outline several aspects that help evaluate if a mined rule is **interesting**:

1. The rule should be easily interpretable and understandable by humans.
2. It should remain valid when applied to new or unseen data (with with some degree of certainty).
3. It should be potentially useful.
4. It should exhibit some degree of novelty rather than restating obvious domain knowledge.
5. It should support or validate a hypothesis set by the user.

Traditional Algorithms for Rule Mining

The field of association rule mining has been dominated by several foundational algorithms, each coming with distinct advantages and computational characteristics. The core challenge in rule mining lies in the exponential growth of possible itemsets as the number of items increases, making brute-force enumeration approaches computationally infeasible. Traditional rule mining techniques usually define a minimal threshold for the quality measures support and confidence, sometimes called *minsupp* and *minconf* [59].

Hipp et al. [59] categorize traditional algorithms based on two characteristics: (1) the strategy used to traverse the itemset search space (e.g. breadth-first vs. depth-first), and (2) the strategy used to compute the support values of the itemsets. Figure 2.7 provides an overview of this categorization.

Apriori The Apriori Algorithm, introduced by Agrawal et al. [6], is the foundational algorithm in this field. It uses a bottom-up approach, where frequent subsets are extended iteratively through candidate generation. Apriori employs a breadth-first search (BFS) with occurrence counting [59]. It introduced candidate pruning based on the downward closure property, which states that all non-empty subsets of a frequent itemsets must also be frequent, eliminating candidates with infrequent subsets before counting their support. Apriori requires multiple database scans and can be inefficient for large datasets.

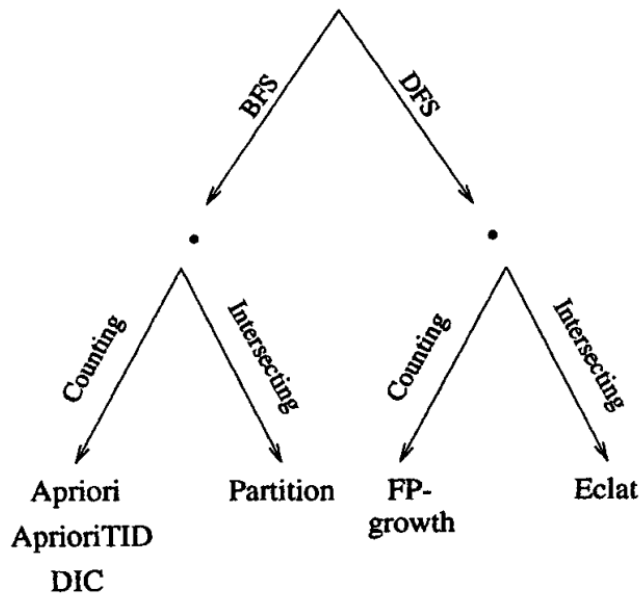


Figure 2.7: Categorization of traditional ASM algorithms by Hipp et al. [59]

Partition The Partition algorithm is an Apriori-like algorithm that addresses its scalability limitations by partitioning the database into chunks to fit easier into memory [59]. Locally frequent itemsets are identified in each partition, and a subsequent scan is required to verify global frequency. This approach reduces memory consumption but incurs the cost of an additional verification scan.

FP-Growth The FP-Growth algorithm [56] efficiently mines frequent itemsets from large datasets by compressing the data into a Frequent Pattern Tree (FP-Tree), which captures itemset frequency information in a hierarchical form. Mining is then performed using a recursive depth-first search (DFS). Unlike Apriori, FP-Growth avoids candidate generation entirely and requires only two database scans, making it more efficient for large datasets.

Eclat Eclat (Equivalence Class Transformation) is a DFS-based algorithm that computes frequent itemsets through intersections of transaction ID lists (TID-lists) [59]. Each item is represented by the list of transaction IDs in which it appears. Frequent itemsets can then be obtained by recursively intersecting these TID-lists. The algorithm maintains only the TID-lists along the current search path, eliminating the need for database partitions. The optimization lies in an early termination for intersections where the minimum support cannot be achieved.

Evaluation metrics

Traditional mining approaches rely heavily on statistical metrics to evaluate candidate rules during the mining process. The most common metrics are **support**, **confidence** and **lift**.

Support of an association rule $X \Rightarrow Y$ is defined as the percentage of records (or transactions) that contain $X \cup Y$ (and thus satisfy the given rule) to the total number of records in the dataset [145]. It is a good indicator on the statistical significance of a given rule, as rules with very low support may just be coincidental rather than meaningful patterns. As an example, imagine a shopping card dataset that contains information on products being purchased together. Given the association rule $Bread \Rightarrow Milk$, if 100 out of 1000 purchases contain both bread and milk, then $Support(Bread \Rightarrow Milk)$ would be $100/1000 = 0.1$ or 10%. The formula for calculating support is as follows:

$$Support(X \Rightarrow Y) = P(X \cup Y)$$

where $P(X \cup Y)$ is the probability that a record contains both itemsets X and Y .

Confidence of an association rule $X \Rightarrow Y$ is defined as the percentage of the number of records that contain $X \cup Y$ to the total number of records that contain X . Thus, confidence thus measures the reliability or strength of the association. Together with support, it is one of the most important metrics to assess rule reliability. Looking at the above example, if Bread would appear in 100 transactions and Bread + Milk appear together in 50 transactions, then $Confidence(Bread \Rightarrow Milk)$ would be $50/100 = 0.5$ or 50%. Confidence is calculated using the following formula:

$$Confidence(X \Rightarrow Y) = P(Y|X) = \frac{P(X \cup Y)}{P(X)}$$

where $P(Y|X)$ is the conditional probability that a record contains itemset Y given that it contains itemset X .

Lift of an association rule $X \Rightarrow Y$ measures how much more often X and Y occur together than it would be expected if they were statistically independent. Thus, lift evaluates the strength of the correlation between X and Y . In the milk and bread example, lift would measure how much more likely milk is to be purchased when bread is purchased compared to milk being purchased independently. It can be used as an indicator to see if the rule is better than a random chance (e.g. Lift > 1 means X and Y are positively correlated, Lift < 1 means they are negatively correlated, Lift = 1 means X and Y are independent). Lift is calculated using the following formula:

$$Lift(X \Rightarrow Y) = \frac{P(Y|X)}{P(Y)} = \frac{P(X \cup Y)}{P(X) \cdot P(Y)}$$

These metrics form the basis for evaluating association rules. In practice, support and confidence are used with thresholds during the mining process, while lift is mostly used to assess the quality and strength of the resulting rules. Both support and confidence are calculated and used in the framework implementation to assist in the evaluation process.

2.3.4 Rule-Based Systems and Logic Programming

Rule-based systems provide a declarative way to represent knowledge and facts that can be used to reason about data. Instead of specifying *how* to compute an outcome, rule-based systems specify *what* relationships or constraints hold in a given domain. An inference mechanism can then be used to derive new information or check a hypothesis against the knowledge base. This paradigm has proven valuable in data management, where rules are used to capture data dependencies, define constraints and enforce data quality.

Prolog and Datalog

Prolog is a declarative programming language based on first-order logic and commonly used for general logic programming [88]. Unlike procedural programming languages, Prolog programs describe what the problem is (using facts and rules) rather than how to solve the problem. A simplified version of Prolog is Datalog, which removes function symbols and restricts terms to constants and variables. Datalog has purely declarative semantics and is mostly used as a database query language [21]. Both Prolog and Datalog provide a formal framework for expressing and evaluating data quality rules over a given knowledge base. In this section the main concepts relevant for understanding both Prolog and Datalog rules are explained.

Predicates and Facts Predicates (or relation symbols) form the basic entities in Prolog and Datalog. They correspond to functions and procedures found in traditional programming languages and to relations in database systems [88]. Predicates are the general relationship or property being described, defined by name and arity (number of arguments). An example is the predicate **employee/3**, which indicates that the predicate takes three arguments.

Facts are ground predicates (e.g. instantiated with concrete values) that represent basic assertions about the world. An example of how facts in the context of database systems (e.g. tuples in a table) can be represented is the following:

```
employee(john, 'Engineering', 75000).
```

which states that John is an employee in the Engineering department with a salary of 75000. In database terms, that corresponds to a tuple in the *employee* relation.

Rules Rules allow the derivation of new facts from existing ones. A rule consists of a head and a body, separated by the symbol **:-**. The head is true if the body holds, thus representing an implication. The body may contain multiple predicates (separated by commas) as well as equality or comparison operators [39]. An example of a Prolog rule is the following:

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

2. THEORETICAL BACKGROUND

which expresses that "X is a grandparent of Z if X is a parent of Y and Y is a parent of Z". In the context of data quality, rules can be used to encode integrity constraint such as:

```
valid_salary(Name) :-  
    employee(Name, Dept, Salary),  
    Salary > 0,  
    Salary < 200000.
```

which checks if the employees salary is within a valid range.

Queries After facts and rules have been defined, a user can execute *queries*. A query has the same structure as the body of a rule. For demonstration purposes, the symbol `?-` can be included to specify that the following expression is a query. An example of a query asking about all employees with valid salaries could look like the following:

```
?- valid_salary(X) .
```

Or more specifically, to ask if John's salary is valid:

```
?- valid_salary(john) .
```

Queries thus allow users to retrieve information and test constraints against a knowledge base.

Closed World Assumption An important feature of logic programming is the *Closed World Assumption* (CWA), which assumes that any fact not explicitly stored in the knowledge base is false [4]. This is different to classical first-order logic, where unknown facts are simply unknown. Additionally, Prolog and Datalog employ *negation as failure*, which means that the negation of a predicate succeeds if the system fails to prove the predicate. These mechanisms are particularly useful for expressing data quality rules, where missing information is often treated as evidence of rule violations.

Related Work

The main research areas relevant for the thesis can be roughly split into (1) traditional statistical and rule-based mining techniques, (2) machine learning approaches, and (3) LLM-based approaches to data quality management. For evaluation and decision making, LLM-as-a-Judge systems have already been covered in academic research. This chapter reviews related work and recent research in these areas.

3.1 Traditional Rule Mining

Specifying data quality rules has traditionally relied on domain experts that carefully design business constraints or integrity rules. This process is both time-consuming and costly, which motivated research into automated discovery of rules from data. Early methods focused on identifying statistical regularities and structural patterns in relational data.

Chiang and Miller [23] explored discovering data quality rules from statistical correlations, showing that automated methods can significantly reduce the reliance on human experts. They developed algorithms to automatically infer Functional Dependencies and introduced the notation of Conditional Function Dependencies, both of which are discussed in Subsection 2.3.2. The proposed discovery algorithms were also able to find dirty values in a given dataset. To avoid the generation of too many rules, the authors evaluated interest metrics to rank and filter their results, so that only the most meaningful CFDs would be used for validation. The work demonstrated algorithms to automatically mine data quality rules that were not only relying on statistical metrics such as support and confidence.

Subsequent research expanded on the types and formalisms used for automated rule mining. Chu et al [26] introduced Denial Constraints as a unifying integrity rule formalism. A denial constraint is essentially a first-order logic formula that specifies

forbidden patterns in the data. The authors not only formalized DCs but also developed methods to automatically discover DCs from data, showing that mining more expressive rules is also feasible. This was a significant advance as DCs can capture more complex conditions that simpler rules could not express. The follow-up system **RuleMiner** [28] used this formalism and addressed two limitations of prior approaches: (1) it opened up the search beyond a single rule type like FDs by expressing all discovered rules as DCs, and (2) it introduced novel interesting-ness rankings to evaluate mined rules. By ranking discovered rules based on their brevity, coverage and lack of redundancy, the system is able to filter out invalid or overly specific rules. This metric dramatically reduces the manual effort needed to evaluate the output, as only the top-ranked constraints are presented to the user.

Despite these advances, traditional rule mining techniques have known limitations that motivate new approaches. First, they often generate an overwhelming number of candidates rules, which many of them are redundant, overly specific or hard to understand by domain experts. Even with interest metrics or rankings, domain experts may still be needed to evaluate or adopt mined rules. Second, purely statistical or pattern-based methods can miss rules that require semantic understanding of the data. For instance, a classic FD discovery algorithm cannot understand that values like "ENG" and "English" refer to the same concept and thus should be standardized.

3.2 LLMs and Data Quality Management

Driven by the limitations of traditional systems, recent work has started to incorporate LLMs to assist in data quality management, leveraging their ability to understand semantics and generate human-like rules or explanations. Unlike the purely data-driven methods discussed above, LLM-based approaches can incorporate external knowledge and context, enabling more meaningful rule suggestions and data cleaning actions. This section covers recent works that apply LLMs to data quality tasks, including rule generation, error and anomaly detection and data cleaning.

Hybrid Semantic Rule Generation Zhang et al. [144] propose *Cocoon*, a novel data cleaning system combining pattern-detection capabilities of traditional solutions with the semantic understanding capabilities of LLMs. Cocoon addresses complex data cleaning tasks by decomposing them into a workflow of smaller steps, mimicking how human experts would approach data cleaning. For each type of data quality issue (e.g. string inconsistencies, numeric outliers, functional dependency violations) the system first uses statistical or rule-based detectors to flag anomalies. Then an LLM is invoked to interpret and suggest corrections using their real-world knowledge. This hybrid pipeline incorporates the LLMs capability of detecting and proposing fixes for errors that do not violate obvious statistical rules but are still semantically wrong. Their evaluation showed that Cocoon outperforms state-of-the-art cleaning systems on standard benchmarks. The authors also note that naive approaches (e.g. prompting a model to clean an entire

dataset in one go) yielded near-zero accuracy on many tasks, highlighting the need for additional guidance in more complex tasks.

LLMs for Graph Data Consistency While most data cleaning research focuses on tabular data, Le et al. [78] explore using LLMs to mine consistency rules in graph-structured data. In their exploratory study, the LLM is prompted with representations of a property graph (with nodes, edges, labels and properties) and tasked to generate constraints that should hold in the data. Their findings show that LLMs can extract useful graph consistency rules automatically. The authors note that most of the rules generated were scheme-like constraints such as keys, unique property requirements or mandatory relationships that ensure graph integrity. For example, in a social network graph, the LLM could infer that no user can follow itself, or that every tweet node must be authored by a user node [78]. The LLM was also able to propose more complex rules such as temporal constraints. However, the results also revealed that LLMs tend to latch onto relatively obvious patterns for rule generation, and only occasionally capture deeper domain-specific rules.

Automating Data Standardization Data quality improvement often involves not just finding errors but also standardizing representations. Qi et al. [112] propose *CleanAgent*, an LLM-driven system focusing on automating these standardization tasks. The authors observe that while LLMs can generate code to transform data, it typically requires an expert user to prompt and refine that code. CleanAgent simplifies this by providing a high-level Python API that covers common data transformations with simple function calls. This makes the LLMs job easier, as instead of writing complex code from scratch, it just needs to choose and parameterize the right API calls required for the cleaning tasks. The system is built in an agentic way, that takes a user's description of how the data should be standardized and then uses the LLM to generate a sequence of API calls to achieve it [112]. The work demonstrates that LLMs can assist in data quality by serving as orchestrators of transformations.

These LLM-based approaches represent a growing trend in hybrid approaches, where traditional data profiling techniques are combined with LLM-driven insights. They demonstrate that LLMs can suggest data quality rules and cleaning steps when properly guided with prompts, templates or additional structured guidance. LLMs can incorporate real-world semantics and knowledge to catch inconsistencies in data, and propose constraints in novel output formats. However, these findings also highlight the importance of constraining or guiding the LLM in achieving the goal.

3.3 LLM-as-a-Judge

In recent years, LLMs have been widely adopted as automated judges to evaluate content or make decisions instead of human experts. LLMs have been used for a wide range of tasks, for example to grade student essays and coding assignments, assess

3. RELATED WORK

the correctness of solutions to math problems and even provide preliminary reviews of academic papers. While human evaluators are still considered to be the gold standard, they are expensive, slow and prone to inconsistencies (e.g. different experts may give diverging opinions). In contrast, LLM-based judges could offer scalability, low cost and uniformity. Unlike traditional programmatic metrics, LLMs can handle qualitative and contextual evaluations. For example, GPT-4 based evaluators have been shown to correlate better with human preferences than metrics like *BLEU* or *ROUGE* for natural language processing tasks [51].

A comprehensive survey by Gu et al. [51] provides a formal definition and taxonomy of LLM-as-a-Judge systems, while also identifying key challenges such as reliability, fairness and consistency. The authors emphasize that designing a trustworthy judge requires careful prompt engineering, calibration and bias mitigation strategies. Another issue is that LLMs may have implicit biases learned from training data or inserted during alignment.

Zheng et al [146] provide an empirical study in this area focusing on evaluating LLM-as-a-Judge systems by introducing two benchmarks. First, *MT-Bench* is introduced as a multi-turn benchmark for dialogue models, and *Chatbot Arena* as a large-scale crowdsource evaluation platform. Their experiments show that modern LLMs like GPT-4 achieve over 80% agreement with human preferences. However, the authors also observed systematic biases including position bias, verbosity bias and self-enhancement bias, which can influence judgments.

Framework and Implementation

This chapter covers the details of the framework architecture design and the implementations done in python. Section 4.1 gives a high-level introduction to the proposed framework, including architecture design, technologies and packages used and provides an overview of the whole generation and evaluation pipeline. Afterwards, Section 4.2 introduces the main contribution of this work, the *Template-Driven Rule Generation* process. Rule compilation and execution are shortly discussed in Section 4.3. The chapter ends with Section 4.4, which gives an introduction and overview of the evaluation components, namely the *Semantic Score Judge* and the *Business Domain Judge*.

4.1 Framework Architecture

The proposed framework was implemented as a python project consisting of multiple components. The main components proposed in this work are the **Rule Generation** component, the **Semantic Score Judge** and the **Business Domain Judge**. A high-level overview of the architecture and external components provided by the **Chat2Data** system can be seen in Figure 4.1.

4.1.1 Technologies and Packages

In order to implement the proposed framework, certain technologies and generic concepts have to be discussed. This section gives a quick overview of some important concepts design decisions made during implementation.

Database Sampling In order to get an LLM to a point where it can understand dataset semantics, one has to provide the LLM with a representative sample of the dataset in question. Simply providing the entire dataset is usually not possible, as large datasets would quickly exceed the LLMs inherent token limit (context window) and

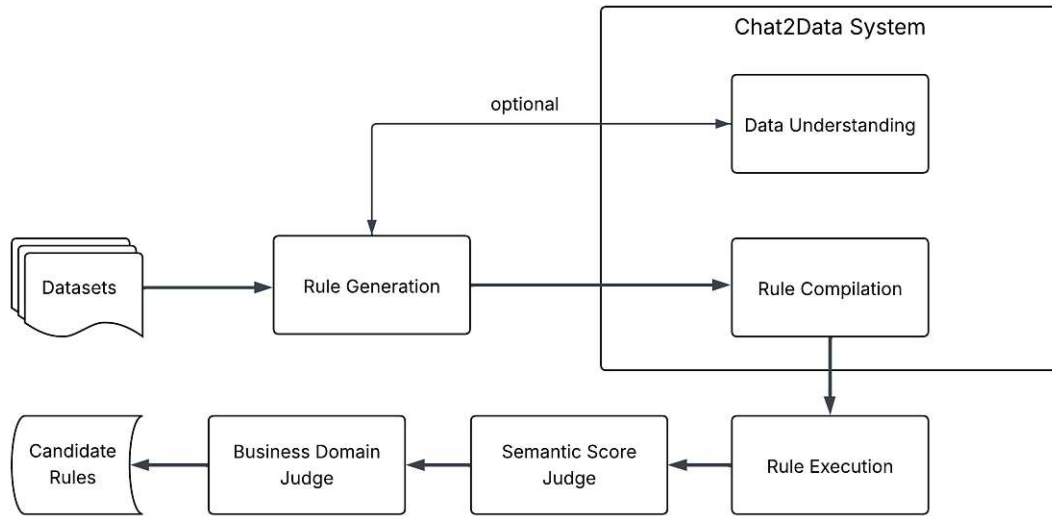


Figure 4.1: High level system architecture diagram of the proposed framework.

it would dramatically increase invocation costs, as the number of input tokens would increase.

Given these constraints, a sample of the dataset must be extracted and included in the prompt sent to the LLM. In the current implementation, this sample employs a naive approach, simply taking the first N rows of the dataset. While being simple and deterministic, this naive sampling strategy has some limitations. It assumes that the initial rows are representative of the entire dataset, which may not hold true if the data is ordered chronologically, sorted by specific attributes or when it contains regional clustering. Furthermore, edge-cases, outliers and rare data patterns might not be caught by the sample.

More sophisticated, informative sampling techniques could substantially increase an LLMs capability in generating useful rules and understanding the underlying data structure. In multi-table scenarios, the sampling challenge becomes even more complex. Sophisticated sampling strategies could be designed to represent more interconnected parts of the dataset, preserving foreign key relationships and ensuring that related records across tables are sampled together. This relational awareness is crucial for multi-table rule generation, as the LLM would first need to see cross-table dependencies to generate meaningful data quality rules.

The extracted sample is formatted in markdown before it gets included in the prompts, an example of how such a sample based on the stocks dataset (the datasets are discussed in more detail later in Section 5.1) looks like can be seen in Listing 4.1.

Listing 4.1: Markdown sample of the stocks dataset

Date	Open	High	Low	Close	Volume	Name
2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2013-02-12	14.45	14.51	14.1	14.27	8126000	AAL
2013-02-13	14.3	14.94	14.25	14.66	10259500	AAL
2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL
2013-02-15	13.93	14.61	13.93	14.5	15628000	AAL
2013-02-19	14.33	14.56	14.08	14.26	11354400	AAL

LLM API Calls As many components of the proposed framework require extensive API calls to external LLM providers, a centralized service was implemented to manage access and provide an easy mechanism to switch out models and providers. The python package **LiteLLM**¹ was used as a simple wrapper to access multiple LLM providers. A core feature of LiteLLM is the provided asynchronous completion functionality, as some parts of the proposed framework require parallelizing the API calls to reduce generation or evaluation times. The current implementation allows access to *Gemini*, *OpenAI* and *Claude* models.

Managing datasets The current implementation works mainly with .csv files, but other formats or wrappers for direct database access could be added easily. For this purpose, **pandas**² is used to load, manipulate, manage and sample datasets throughout the application.

Asynchronous capabilities Performance plays a crucial role in generating and evaluating data quality rules, as the process involves the generation of many candidate rules that also have to be evaluated. Due to the fact that API calls to external LLM providers already take quite some time, other areas of the application have been designed with a focus on performance and scalability. For this purpose, the **asyncio**³ python library is used to write concurrent, non-blocking code. Asyncio enables the application to issue multiple LLM API requests concurrently without blocking on each individual response, effectively parallelizing the operations. While one API call is waiting for a response from the LLM provider, the event loop can initiate additional requests or perform other tasks. Beyond LLM interactions, asyncio is also employed for concurrent rule execution against the dataset. Once candidate rules are generated, they must be validated by executing them against the actual data. Given the large number of candidate rules generated, parallelizing this step dramatically reduces execution time.

¹LiteLLM: LLM API wrapper package <https://docs.litellm.ai/>, Accessed: 2025-09-25

²Pandas: data analysis and manipulation tool <https://pandas.pydata.org/>, Accessed: 2025-09-25

³asyncio: Python concurrency library <https://docs.python.org/3/library/asyncio.html>, Accessed: 2025-09-25

4.1.2 Framework Pipeline

The whole workflow from data understanding to rule generation and evaluation is implemented as a pipeline. The main components of the pipeline are shortly discussed in this subsection. Each component implemented in the framework is discussed in detail later in this chapter.

Data Understanding The first step in the pipeline is the so-called data understanding process, which is optional but usually increases the overall quality of the rule generation and evaluation process. This step essentially uses LLMs to get insights on a given dataset. The data understanding step part is provided by the **Chat2Data** system and is thus only covered shortly.

The data understanding process gathers comprehensive information about the dataset, with particular emphasis on the following key aspects:

1. **Dataset Overview:** A comprehensive, human-readable description of the dataset's purpose, scope, structure and business context is generated.
2. **Domain Knowledge:** Contextual information including industry-specific terminology, business rules and domain-specific constraints that might be relevant are generated.
3. **Attribute Analysis:** A detailed semantic analysis of each column including data types, formats and categorical value meanings is generated.
4. **Attribute Relationships:** Logical dependencies between attributes, functional constraints, hierarchical relationships and mutually exclusive or conditional requirements are described.

Rule Generation A rule generation step takes one or more datasets and an optional data understanding result to generate a list of rule candidates. The main component is a template-driven generation process, which generates Prolog-like data quality checking rules. The template-driven generation process is discussed in detail in Section 4.2.

Rule Compilation The rule compilation step is performed on each generated rule candidate in order to get executable python-code that evaluates the rule against the dataset. This step is provided by the **Chat2Data** system. For the purpose of this thesis, it is assumed that the code provided by this step is valid and works for all generated rules.

Rule Execution The rule execution step takes the returned code from the rule compilation and executes it against the corresponding datasets. To speed things up, this process runs parallelized in multiple threads. The rule execution process then returns an object containing the statistical metrics *support* and *confidence* and a list of violations and satisfactions (e.g. as a list of indices) for the executed rules.

Semantic Score Judge The first evaluation step is the so-called Semantic Score Judge. This process calculates a novel *semantic-score*, which is a new type of metric that aims at catching the semantic meaning of a rule. This is achieved by using an LLM-as-a-Judge instance to evaluate whether or not the found violations for a given rule are genuine data errors or false positives. The results of these judgments are then used to calculate the semantic-score. This process is discussed in detail in Subsection 4.4.1.

Business Domain Judge The final evaluation of the pipeline is the Business Domain Judge. This LLM-as-a-Judge instance evaluates a given rule together with the calculated semantic-score. The goal of the Business Domain Judge is to remove useless, overly specific, and rules that lack semantic meaning. The judge then outputs some scores that reflect this judgment, which can be used to decide if the given rule should be kept or discarded.

4.2 Rule Generation

This section covers the main contribution of this thesis, the template-driven rule generation process. We begin by examining the limitations of naive prompting approaches through concrete examples that motivate the need for a more structured generation methodology. Subsection 4.2.1 presents the architectural design of the rule generation component. Following this, Subsection 4.2.2 provides an in-depth discussion of the template-driven implementation, including the underlying, structure defining bag-templates, the prompt generator and observations made during implementation.

Limitations of Naive Prompting To understand the necessity of a structured approach to rule generation, we first examine the performance of state-of-the-art LLMs when prompted in a naive way. When prompting recent LLMs like *GPT-4o* with a generic generation prompt, several limitations emerge that prevent reliable rule generation.

To illustrate the limitations, consider the following experiment. When prompting *gpt-4o* with the generic generation prompt shown in Listing 4.2 using a sample from the competitors dataset (see Table 4.1 for the schema), the model produces the results shown in Listing 4.3. The result shows both the potential and the fundamental limitations of naive prompting approaches. Note that several lower-quality rules have been removed from this listing to keep it precise.

Listing 4.2: Generic Rule Generation Prompt

```
Generate prolog-like data quality checking rules for the following
dataset:
{SAMPLE}
```

Listing 4.3: Example response of the Generic Rule Generation Prompt.

```
% Every competitor must exist in the companies table:
```

```

invalid_competitor(C1,C2) :-
    competitor(C1,C2),
    \+ company(C1).

invalid_competitor(C1,C2) :-
    competitor(C1,C2),
    \+ company(C2).

% Every similar link must point to valid products:
invalid_similar(P1,P2) :-
    similar(P1,P2),
    \+ product(P1).
invalid_similar(P1,P2) :-
    similar(P1,P2),
    \+ product(P2).

% Competitor links should be symmetric:
missing_competitor_symmetric(C1,C2) :-
    competitor(C1,C2),
    \+ competitor(C2,C1).

% Similar products should also be symmetric:
missing_similar_symmetric(P1,P2) :-
    similar(P1,P2),
    \+ similar(P2,P1).

% Founded year shouldnt be in the future:
invalid_founded_year(ID,Year) :-
    company(ID,Year),
    Year > 2025.

% Warranty should be non-negative:
invalid_warranty(PID,Months) :-
    product(PID,Months),
    Months < 0.

% Similar products should have same category:
invalid_similar_category(P1,P2,Cat1,Cat2) :-
    similar(P1,P2),
    product(P1,Cat1),
    product(P2,Cat2),
    Cat1 \= Cat2.

```

The result set contains some useful rules, especially ones that target referential integrity checks or symmetry characteristics of the competitor and similar relations. However, it also made some assumptions that might not hold (e.g. that similar products should have the same category). The syntactic check constraints are also quite limited, as the result set only included two checks, namely that the company's founded year shouldn't be in

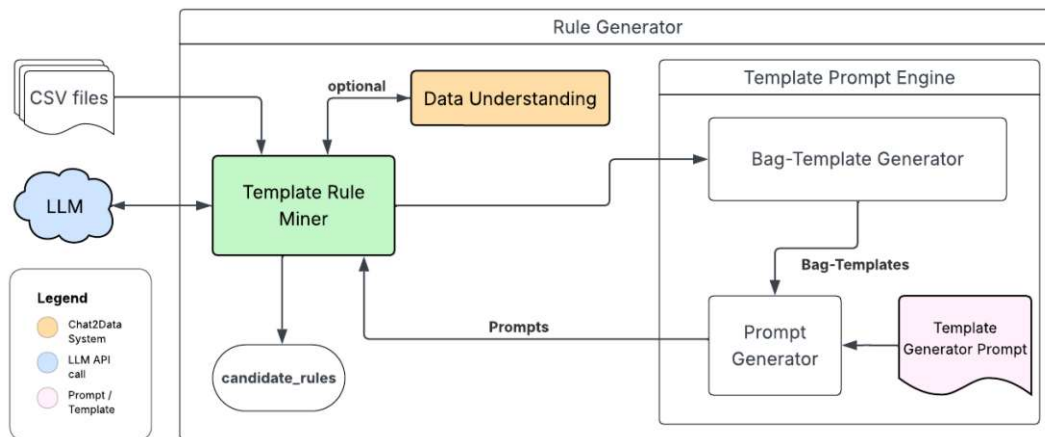


Figure 4.2: System architecture diagram of the proposed Rule Generation implementation.

the future and that warranty must be non-negative.

The primary issues of naive prompting for the rule generation observed during our experiments include: (1) **incompleteness**, as the generated rule sets fail to cover all relevant patterns present in the data and (2) **reliability**, as repeated prompting with identical inputs yields substantially different outputs, making the process unreliable.

Without additional guidance, the LLMs tend to generate rules based on the most obvious patterns in the sample, rather than systematically generating rules that cover a wider range of patterns and checks. This motivates the need for the template-driven, systematic approach that provide explicit structure to guide the generation process towards a more reliable and complete result set.

4.2.1 Rule Generation System Architecture

The proposed **Rule Generator** component consists of the **Template Rule Miner**, which is the main entry point responsible for processing and delegating the rule generation procedure. It can optionally call the **Data Understanding** component to get more information about the datasets in question. It then calls the **Bag-Template Generator** to get a list of bag-templates, which are discussed later in this Section. The **Prompt Generator** takes these bag-templates and generates fully fledged generation prompts by intelligently constructing them depending on the bag-template structures. The **Template Rule Miner** then takes these prompts, queries them against an LLM, and extracts the generated candidate rules in a structured format. An overview of the architecture of the rule generation component can be seen in Figure 4.2.

4.2.2 Template-driven Rule Generation

The main component of the proposed rule generation process is the so-called template-driven generation process. This generation process introduces **bag-templates**, which specify a blueprint for the rules to be generated. This method systematically guides the LLMs to generate more comprehensive and sophisticated rules by providing structured templates that specify the desired complexity and shape of the rules generated.

Bag-templates A bag-template is essentially a pre-structured pattern for rules, expressed as a set (bag) of relational symbols and operators, each defined with specific number of occurrences. Each bag-template defines which database predicates (i.e., relation names) and comparison operators may occur in the body of a generated Prolog-like rule, how often they may occur, and optionally which predicate or comparison atom must appear in the head of the rule. This controlled specification reduces the combinatorial explosion that would arise from enumerating all possible rule patterns directly, while still giving the LLM a certain degree of freedom to come up with meaningful rules that match the pre-defined pattern.

For demonstration purposes, consider the competitors database schema from Table 4.1. This database represents a structured model of products and their associated companies, a similarity relationship between products and a relation holding company competitors.

Table 4.1: Competitors database schema [47]

(a) Products

ID	Name	Manufacturer	Price
101	X100F	307	\$1399
102	GRIIIx	308	\$969
220	D850	309	\$2799

(b) Similar Products

PID1	PID2
101	102
102	220

(c) Companies

ID	Name	Industry	HQ
307	Fujifilm	electronics	Japan
308	Ricoh	electronics	Japan

(d) Competitors

CID1	CID2
307	308
307	421

Consider the following example bag-template: `[COMPETITORS(1), SIMILAR(2), PRODUCTS(3)]`. This bag-template specifies, that the generated Prolog rules should contain exactly one `COMPETITORS` atom, two `SIMILAR` atoms and three `PRODUCTS` atoms in the rule body, along with optional comparison operators. This bag-template then gets transformed into a complete prompt which guides the LLM into generating rules of exactly that form. We observed that this specification reduces the LLMs inherent *laziness*, as it forces the LLM into generating rules that fit this pattern.

Some more examples of bag-templates that are able to generate specific types of rules on the competitors dataset are as follows. The bag-template `[head:SIMILAR, SIMILAR(2)]` can lead to the generation of the transitivity rule `SIMILAR(X, Z) :- SIMILAR(X, Y), SIMILAR(Y, Z) ..` The bag-template `[head:SIMILAR, SIMILAR(1)]` can yield the symmetry rule `SIMILAR(X, Y) :- SIMILAR(Y, X)`. There are many more interesting bag-templates that lead to the generation of useful results, some of these bag-templates are shown in Section 5.1.

To increase the expressive power of the bag-templates, the syntax allows for the following extensions.

- **Head enforcement:** A template may specify that a particular database predicate atom or a comparison operator must occur in the rule head, e.g. `[head:SIMILAR]` or `[head:<]`. This can be used to construct prompts that generate TGDs or other dependencies such as asymmetry or transitivity rules. In cases where no head specification is given, denial-constraints might be generated that specify invalid states.
- **Multiplicity constraints:** Database predicates and comparison operators are annotated with occurrences, specifying how many times the annotated database predicate or comparison operator must occur in the generated rule, e.g. `[SIMILAR(2)]` (exactly two occurrences of the SIMILAR predicate) or `[COMPETITORS(1)]` (exactly one occurrence of the COMPETITORS predicate).
- **Comparison operators:** Inequality and ordering predicates (such as $\neq, <, >, \leq, \geq$) can be explicitly included or restricted, e.g. `[PRODUCTS(2), <(2), \neq (1)]` specifies that the PRODUCTS predicate shall appear exactly two times, the ' $<$ ' comparison operator three times and the ' \neq ' comparison operator one time.
- **Wildcards:** Bag-templates allow for the specification of wildcards in the multiplicity constraints, i.e. **optionally some** (*) and **at least one** (+). For example, the bag-template `[PRODUCTS(2), SIMILAR(+)]` instructs the LLM to generate rules with exactly two PRODUCTS predicates and at least one SIMILAR predicate. These wildcard specifications are especially helpful when a minimal set of bag-templates is required in scenarios where token cost and execution time are important.

Bag-Template Generation

The rule generator includes a component called the **Bag-Template Generator**, which is responsible for enumerating and generating a list of bag-templates that will be turned into prompts by the **Prompt Generator**. The bag-template generation takes in several options that allow for a fine-grained configuration of the generated bag-templates.

Enumeration characteristics To provide some flexibility over the generated bag-templates, the bag-template generator allows for the following parameters.

- **Maximum bag-size (default=7):** The maximum size of the bag. It sets the maximum number of database predicate atoms or comparison operators that are included in a bag-template. For example, setting this to three can generate sets of bag-templates that include combinations of up to three different database predicates or comparison operators (e.g. `[SIMILAR(1), PRODUCTS(2), =(1)]`).
- **Maximum database predicates (default=3):** Sets the maximum times a specific database predicate can appear in a single bag-template. This is usually set to three, because constraints where a relation occurs more than three times are unusual in practice. In single-table scenarios it is usually enough to set this to 2, as experiments showed that the resulting rules with higher settings are overly specific or of low quality.
- **Use comparison operators:** This flag either enables or disables the use of comparison operators appearing in the generated bag-templates. By default, this setting is set to false. If it is enabled, the bag-generator will generate bags that include specifications for occurrences of the comparison atoms `[≠, <, >, ≤, ≥, =]`.
- **Maximum comparison operators (default=2):** Sets the maximum times comparison operators can appear in a set of bag-templates, providing separate control over the amount of comparison operators.

The enumeration behavior of the bag-template generator exhibits both linear and exponential growth characteristics depending on the chosen parameters. As shown in Figure 4.3 and Table 4.2, the total number of enumerated bag-templates increases approximately linearly with the maximum bag-size, since each additional allowed atom position extends the combinatorial search space by a roughly constant factor. However, the enumeration grows exponentially with the number of available relations, as each new relation introduces additional possible combinations of predicate occurrences within the bag-templates. A similar exponential growth can be observed when increasing the maximum allowed occurrences per relation, since each added occurrence multiplies the number of potential configurations.

When comparison operators are enabled, the enumeration space expands even further, as these operators are treated the same way as database predicates. This effect can be clearly seen in the right plot of Figure 4.3, where the inclusion of six additional operator symbols leads to a significantly more generated bag-templates.

In practice, this means that the generator parameterization allows for a flexible trade-off between coverage and computational cost. Smaller bag sizes and restricted operators lead to compact enumerations suitable for fast generation, while higher limits enable more structured and expressive templates with increasing inference cost and time consumption. Generally speaking, the bag-template should be configured in such a way that it covers the most important bag-templates for a given set of input relations. Further discussions on this topic can be seen in Paragraph 5.4.1.

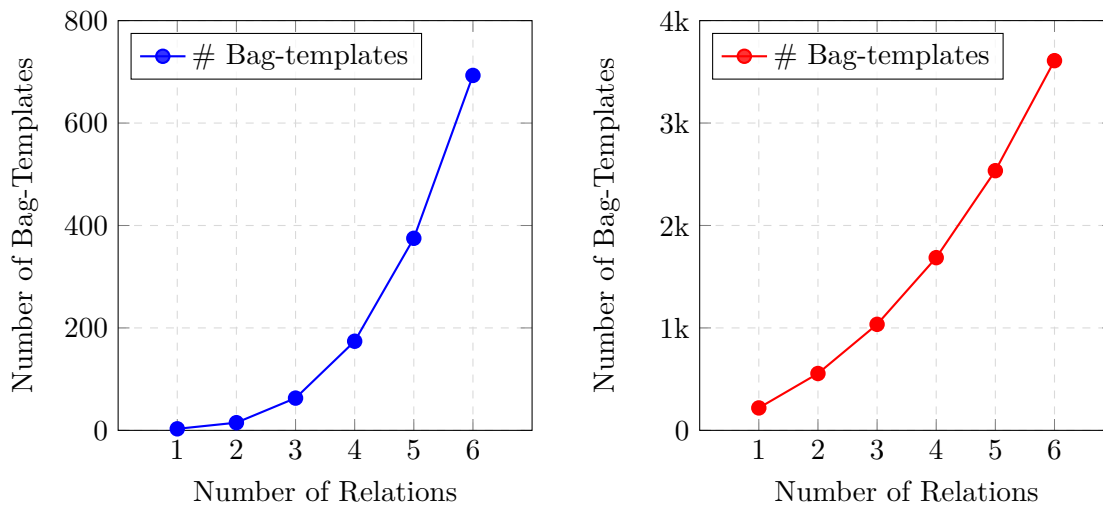


Figure 4.3: Growth of the number of generated bag-templates with an increasing number of relations. Left: without comparison operators, Right: with comparison operators. Configuration: maximum bag size = 3, maximum database predicates = 3.

Table 4.2: Growth of the number of generated bag-templates with an increasing number of relations and increasing max bag-size. Configuration: maximum database predicates = 3.

Comparisons	Max Bag Size	Number of Relations					
		1	2	3	4	5	6
False	1	3	6	9	12	15	18
	2	3	15	36	66	105	153
	3	3	15	63	174	375	693
	4	3	15	63	255	780	1,908
	5	3	15	63	255	1,023	3,366
	6	3	15	63	255	1,023	4,095
True	1	3	6	9	12	15	18
	2	39	87	144	210	285	369
	3	219	555	1,035	1,686	2,535	3,609
	4	699	2,055	4,419	8,223	13,980	22,284
	5	1,419	4,935	12,519	27,195	53,283	96,642
	6	1,995	8,247	25,047	64,599	148,179	310,023

Prompt Generator

The **Prompt Generator** takes in a bag-template and transforms it into a natural language generation prompt. It does so by inspecting the bag-template and constructing the prompt matching the structure defined in the bag-template.

For example, the bag-template [head:COMPETITORS, PRODUCTS(3), SIMILAR(2)] will result in the generator prompt seen in Listing 4.4.

Listing 4.4: Example Generator Prompt

```
{SAMPLE}
{DATA_UNDERSTANDING}

Assume relations COMPETITORS(CID1,CID2), PRODUCTS(ID,Name,
  Manufacturer,Price), SIMILAR(PID1,PID2) as explained before.
Generate prolog-like data quality checking rules that have a
COMPETITORS atom in their head and exactly 3 PRODUCTS atoms
and 2 SIMILAR atoms in their body, and no other database
predicate atoms, plus possibly some comparison atoms with
comparison operators !=, <, >, <=, and >=, and some negated
atoms.

If no head is specified also consider creating denial-constraints
with an empty head. Skip irrelevant columns by denoting them
with an underline.

You may use a predefined predicate match(<Value>,<Regex>) in the
rule body, that checks whether a value <Value> satisfies the
regular expression <Regex>.

By re-examining each rule carefully, please suppress the
repetition of a rule and also suppress any generated rule that
is recognized to be equivalent to an earlier generated rule.

## Output format
Format the output in clean json format using the following
structure:
{{
  "rule_name": "<rule name>",
  "rule": "<rule>"
  "rule_description": "<short description of the rule>",
}}
```

While the current implementation enumerates through all possible combinations of relations, it should be noted that choosing combinations of relations prior to the bag-generation process is advised. For example, if certain relations have nothing in common or are not relating to each other through foreign keys, it usually doesn't make sense to generate constraints that include those relations together. To solve this, the data understanding step could be extended to create pairs of relations that potentially have dependencies between each other. This groups of relations can then be set in the bag-

template generator to reduce the amount of bag-templates (and thus tokens needed) before creating the prompts that will be sent to the LLM.

Implications of overly specific bag-templates Experiments with this approach show that it leads to the generation of many rules, but the resulting rules are often too specific or make no sense at all. It seems as if the LLMs capability of generating useful rules sometimes collapses when the provided rule structure makes no sense on a given dataset or on a combination of relations. For example, when a bag-template includes two relations that are not related at all, the LLM will still generate some rules, but they won't have any useful meaning.

Specifying rule types Previous iterations of the generation prompts included type hints on the rules to be generated (e.g. functional dependencies, inclusion dependencies). We observed in early experiments that hinting the LLM with rule types in the generation prompt will almost exclusively lead to the generation of rules for these types. As one of the main goals for the template-driven generation was not to be reliant on rule type specifications, type hinting in the generation prompt was avoided. However, giving the LLM the freedom to generate rules in denial-constraint forms (e.g. with empty heads) has shown to be quite effective, as this format is very good at specifying data quality checking constraints.

4.3 Rule Compilation and Execution

To get statistical metrics, satisfying and violating rows for a given rule, a process called rule compilation and execution is employed. As the generated rules for such a framework can come in a variety of forms (e.g. in first-order logic formulas, as Prolog rules or in natural language), a dynamic execution process that works for all kind of rules is needed. The **Chat2Data** system employs a LLM-based rule compilation process, which generates executable python code for each given rule. This python code can then be executed against a dataset to obtain statistical metrics like the rules *support*, *confidence* and actual violating and satisfying data (e.g. row indices). A high-level overview of these components can be seen in Figure 4.4. The results of this process is necessary for the Semantic Score Judge, as that component needs to extract actual violating row data for the evaluated rules.

4.4 Rule Evaluation

As the template-driven rule generation process yields a lot of rules, rule evaluation and filtering is necessary to reduce the total amount of generated rules to a reasonable amount. For this purpose, two evaluation components are included in the proposed framework. The first evaluation is done via a so-called **Semantic Score Judge**, which evaluates a list of sampled violations for a given rule and calculates a semantic-score based on these

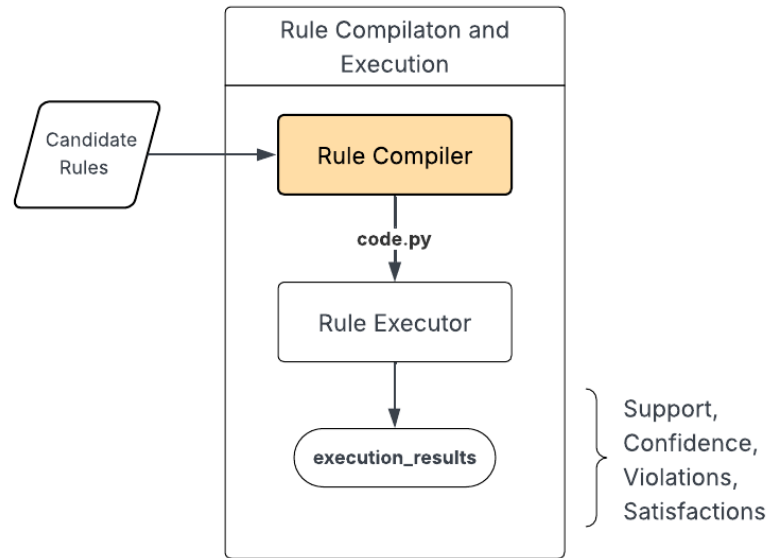


Figure 4.4: Rule Compilation and Execution system architecture. The rule compiler was provided by the Chat2Data system.

evaluations. The second evaluation is done using a so-called **Business Domain Judge**, which essentially acts as a final filtering mechanism to remove domain-inappropriate, low quality or overly specific rules.

4.4.1 Semantic Score Judge

Traditional evaluation metrics mostly rely on statistical characteristics of a given rule (e.g. by calculating support and confidence). While these metrics are useful for certain cases, they do not reflect a rule’s semantic meaning. To tackle this issue, the proposed framework introduces a novel *Semantic-Score*, which is calculated by evaluating rule violations and categorizing them into **genuine data errors** or **false positives**. This classification is done by a LLM-as-a-Judge system, which is the core component of this evaluation step. An architectural overview of the Semantic Score Judge and its components can be seen in Figure 4.5.

Classification of violations For each rule, a number of violations is randomly sampled from the evaluation results gathered from the rule execution step. These violations are then sent to an LLM with the semantic score classification prompt listed in Listing 4.5. To get reliable results, the number of violations should be large enough (i.e. at least 30-50). As this process would take a long time if each violation was evaluated within a single, isolated prompt, a batch-processing prompt was designed to speed things up.

This prompt sends multiple violations (currently in batches of 10) in a single prompt and formats the output in a structured format. This process significantly reduces token usage, as the initial inputs required (data understanding result, sample data) only have to be sent once per batch.

Listing 4.5: Semantic Score Classification Prompt

```

**Task: Evaluate Multiple Violations of Data Quality Rules**

You are an expert data quality analyst tasked with evaluating whether
detected rule violations represent genuine data errors or false
positives. Your judgments will be used to compute a semantic score
that measures how accurately a data quality rule identifies real
data errors.

**Inputs:**
- Data understanding results (IMPORTANT: contains assumptions that
  may be inaccurate)
  <data_understanding>{data_understanding}</data_understanding>
- Sample Data
  <sample_data>{sample_representation}</sample_data>

**Rule to evaluate**
<rule>{rule}</rule>
<rule_description>{rule_description}</rule_description>

**Violations to evaluate:**
<violations>{violations_text}</violations>

**Task Description**
For each provided violation, classify it into exactly one of these
categories:

(1) Likely a Real Data Error
The violation represents a genuine data quality issue that should be
addressed
Examples: obvious typos, impossible values, logical inconsistencies,
missing critical data
The data appears corrupted, incorrectly entered, or violates business
logic

(2) Unlikely to be a Real Data Error
The violation is likely a false positive where the rule is too strict
or misapplied
Examples: valid edge cases, acceptable variations, legitimate
exceptions
The data appears correct within its proper context or domain

(3) Unclear / Hard to Tell
Insufficient information to make a confident determination

```

```

The violation could reasonably be either a real error or false
positive
Ambiguous cases where domain expertise or additional context is
needed

**Output Format**
Provide your response as a JSON array with one object for each
violation in the same order they were presented:
[
  {{
    "violation_index": 1,
    "classification": 1 | 2 | 3,
    "confidence": 0.0-1.0,
    "reasoning": "Detailed explanation of your decision."
  }}
]
"""

```

Semantic-score The semantic-score is a metric that quantifies how accurately a data quality rule identifies genuine data errors vs. false positives, as determined by a LLM-based classification of rule violations. Unlike traditional metrics, the semantic-score aims at capturing semantic meaning and practical utility of a rule by evaluating the quality of its violation detection.

For a given rule with a set of violations V , a sample of violations (typically 30-50) is randomly selected and evaluated by the LLM judge. Each violation is classified into one of three categories:

- **(1) Likely a Real Data Error:** The violation represents a genuine data quality issue that requires attention
- **(2) Unlikely to be a Real Data Error:** The violation represents a false positive where the rule might be too strict
- **(3) Unclear / Hard to Tell:** Ambiguous cases where the LLM is not sure on the categorization

Due to the observed tendency of the LLM judge to classify violations as "*Unlikely to be a Real Data Error*", a weighted approach is employed to give higher emphasis to correctly identified genuine data errors. Violations classified as "*Unclear / Hard to Tell*" are excluded from the calculation. The formula for the (weighted) semantic-score is shown in Equation 4.1:

$$\text{Semantic Score} = \frac{w * R}{w * R + F} \quad (4.1)$$

Where:

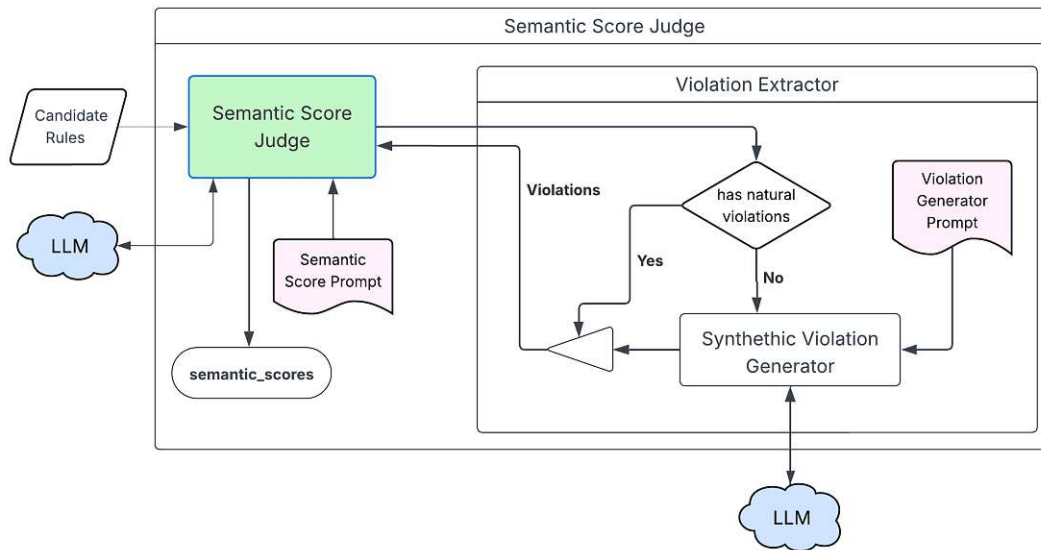


Figure 4.5: System architecture design of the Semantic Score Judge.

- R = Number of violations classified as "Likely a Real Data Error"
- F = Number of violations classified as "Unlikely to be a Real Data Error"
- w = Weight factor for real errors (default: 2.0)

The weighting factor w ensures that rules that are effective in identifying genuine data errors receive higher semantic scores, compensating for the judge's conservative bias. Experiments with other models or iterations of the judge prompt that include few-shot examples could also help in mitigating that bias. A semantic score approaching 1 indicates that a rule mostly identifies genuine data errors, while scores closer to 0 suggest that a rule detects mostly false positives. While the current implementation does not filter based on the semantic-score alone, it should be noted that thresholds could be applied to already filter out scores with low semantic scores in this step.

Synthetic violation generation

As the semantic-score calculation is dependent on analyzing the violations of a given rule, evaluating rules without natural violations is not directly possible. To tackle this issue, the proposed framework includes a **Synthetic Violation Generator**, which can be used to create a variety of synthetic violations for a given rule. These generated violations can then be used to calculate the semantic-score. It should be noted that this technique has not been evaluated, for further discussions on limitations see Paragraph 5.4.2.

The prompt template for the synthetic violation generation can be seen in Listing 4.6.

Listing 4.6: Synthetic Violation Generation Prompt

```

**Task: Generate Synthetic Data Quality Rule Violations**

You are tasked with generating realistic synthetic violations for a data quality rule
that currently has no violations in the dataset. These synthetic violations will be
used to evaluate the rule's ability to detect real data errors.

**Inputs:**
- Data understanding results
<data_understanding>{data_understanding_results}</data_understanding>

- Dataset Sample
<dataset_sample>{sample_representation}</dataset_sample>

**Rule to Generate Violations For:**
<rule>{rule}</rule>
<rule_description>{rule_description}</rule_description>

**Requirements:**
1. Generate {settings.num_violations} diverse synthetic violations that would trigger
this rule
2. Each violation should represent a plausible data error that could occur in real-
world scenarios
3. Violations should vary in severity and type (some obvious errors, some subtle)
4. Consider different violation categories:
- **Type 1: Obvious Data Errors** - Clear mistakes (typos, impossible values)
- **Type 2: Edge Case Violations** - Valid-looking but rule-breaking data
- **Type 3: Contextual Violations** - Values that break business logic or domain
rules

**Output Format:**
Provide your response in this exact JSON structure:
{{
  "violations": [
    {{
      "violation_id": 1,
      "violation_type": "obvious_error" | "edge_case" | "contextual_violation",
      "severity": "high" | "medium" | "low",
      "synthetic_row": {{
        "column1": "value1",
        ...
      }}
      "violation_description": "Clear explanation of what makes this a violation",
      "expected_classification": 1 | 2 | 3,
    }},
    ...
  ]
}}

**Important Guidelines:**
- Make violations realistic and contextually appropriate for the dataset domain
- Ensure synthetic rows follow the general structure and data types of the original
dataset
- Vary the violation types to test different aspects of the rule
- Include both clear violations (should be classified as real errors) and ambiguous
cases
- Consider the rule's intent and create violations that test its boundaries

```

4.4.2 Business Domain Judge

While the Semantic Score Judge evaluates rules based on their ability to detect genuine data errors, not all semantically valid rules are appropriate for a given business domain or dataset context. The **Business Domain Judge** serves as a final filtering mechanism to ensure that only high-quality, domain-appropriate, and practically valuable rules are kept in the final rule set. This component acts as a gatekeeper, removing rules that may

be statistically sound or semantically valid but lack practical utility or domain relevance. This is especially important in the context of the template-driven rule generation process, as it also generates a lot lower quality rules.

Evaluation approach The Business Domain Judge evaluates each candidate rule using a LLM-as-a-Judge system that considers multiple quality dimensions. Unlike the Semantic Score Judge, which focuses on violation classifications, the Business Domain Judge evaluates the rule by considering its domain appropriateness, practical value and overall rule quality. The evaluation process is designed to handle rules with and without semantic-scores, and could be extended to also consider statistical metrics like support and confidence.

For each candidate rule, the judge receives the following inputs:

- Data understanding result and sample data for context
- The rule definition and its generated description
- Optional semantic score from the Semantic Score Judge

The evaluation prompt can be seen in Listing 4.7. It instructs the LLM to assess rules across three primary dimensions, each scored on a scale between 1-5:

1. **Domain Appropriateness (1-5):** This dimension measures whether or not the rule makes sense for the specific dataset and business context.
2. **Rule Quality (1-5):** This dimension evaluates whether the rule is well-formed and if it is overly restrictive or permissive. It was introduced during experiments on more complex datasets, where the rule generation process generated many rules that were too complex or overly specific due to the underlying bag-templates.
3. **Practical Value (1-5):** This dimension assesses whether or not the rule detects meaningful data quality issues that would improve overall data quality if addressed.

Listing 4.7: Business Domain Judge Prompt (abbreviated)

```

**Task: Final Decision on Data Quality Rule Retention**

You are a senior data quality architect responsible for curating a high-
quality set of data quality rules. Your task is to make the final
decision on whether a candidate rule should be kept or discarded from
the rule set.

**Inputs:**
- Data understanding results: <data_understanding>{data_understanding}</
  data_understanding>
- Sample data: <sample_data>{sample_representation}</sample_data>

```

```

- Rule Under Evaluation:
  <rule>{rule_candidate.rule}</rule>
  <rule_description>{rule_candidate.rule_description}</rule_description>
- Semantic Score (optional): {semantic_score} (0.0 = poor at finding real
  data errors, 1.0 = excellent; may be 'N/A')

**Evaluation Approach:**
- If the semantic score is provided, use it as a qualitative signal only.
- If the semantic score is missing ('N/A'), base your decision primarily on
  domain appropriateness, practical value, and rule quality.

**1. Domain Appropriateness**
- Does this rule make logical sense for this specific dataset and domain?
- Are the constraints reasonable given the business context?

**2. Optional Semantic Signal**
- Treat the semantic score as a supportive indicator
- Do not overrule strong domain logic solely because of a mediocre score

**3. Practical Value**
- Does the rule detect meaningful data quality issues?
- Is the rule specific enough to be actionable?

**4. Rule Quality Assessment**
- Is the rule logically sound and well-formed?

**Output Format:**
{{
  "decision": "KEEP" | "DISCARD" | "UNCERTAIN",
  "confidence": 0.0-1.0,
  "primary_reason": "Main factor driving the decision",
  "domain_appropriateness_score": 1-5,
  "rule_quality_score": 1-5,
  "practical_value_score": 1-5,
}}

```

Decision framework Based on the multi-dimensional assessment, the Business Domain Judge makes one of three decisions:

- **KEEP:** The rule is domain-appropriate, practically valuable and well-formed and thus should be kept in the result set.
- **DISCARD:** The rule is inappropriate for the given domain, has low practical value or is poorly formed and thus should be discarded from the result set.
- **UNCERTAIN:** If the judge is not sure about the rule in question, the rule is classified as uncertain.

Each decision comes with detailed reasoning that includes the primary factor for the decision. When semantic scores are available, they are treated as supportive indicators

rather than decisive factors. This is important, as the whole generation pipeline sometimes fails to compile, execute or calculate the semantic scores properly (or there are no natural violations). In these cases, the Business Domain Judge should still be able to evaluate the given rule based on domain logic or practical value. As the Business Domain Judge outputs the scores for each dimension, a custom decision logic can easily be implemented for more fine grained filtering control.

Parallel processing To efficiently handle large rule sets, the Business Domain Judge implements parallel processing through asynchronous execution. Multiple rules are evaluated concurrently, significantly reducing total evaluation time. To evaluate larger rule sets, a batch evaluation is possible that evaluates a batch of rules per prompt, significantly reducing token cost.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Results and Discussion

This chapter covers the results and evaluations of the main components of the proposed framework, namely the template-driven **Rule Generator**, the novel semantic-score metric calculated based on evaluations done by the **Semantic Score Judge** and the results of a consistency evaluation conducted for the **Business Domain Judge**.

5.1 Rule Generation Evaluation

In order to evaluate the rule generation process, the generated rules are evaluated against a pre-defined set of golden-rules, which were carefully selected by domain-experts. These golden-rules represent a minimal set of important constraints and rules for the corresponding dataset. The evaluation is done by running the template-driven generation process on each dataset. The generated rules are then manually checked against the list of golden-rules. Furthermore, the generated list of rules is inspected and labeled by a group of collaborating researchers working on the Chat2Data system. Some of the generated rules that were labeled as valuable (and were not already included in the golden-rules) are shown for each dataset.

5.1.1 Evaluation Datasets

Evaluation of the proposed rule generation framework is done on three datasets that were kindly provided by collaborating researchers working on the Chat2Data system. The evaluation focuses on single-table datasets to reduce complexity and the number of rules that have to be manually evaluated. However, early experiments show promising capabilities in multi-table rule generation, which is an interesting topic for future research 6.2.

Hospital dataset The hospital dataset [27] contains hospital performance metrics, tracking clinical quality measures for acute care facilities. Each row represents a specific's hospitals performance on standardized measures (e.g. heart attack care, infection prevention) and references to state averages. The dataset also contains hospital identification details like location information, facility characteristics and clinical quality measure codes. A shortened sample of the dataset can be seen in Table 5.1. Please note that some columns have been removed in the sample for demonstration purposes.

Provider	Hospital Name	City	State	ZIP	Hospital Type	Condition	Measure Code
10018	Callahan Eye..	Birmingham	AL	35233	Acute Care..	Surgical Inf..	SCIP-CARD-2
10019	Helen Keller..	Sheffield	AL	35660	Acute Care..	Heart Attack	AMI-1
10001	Southeast Ala..	Dothan	AL	36302	Acute Care..	Heart Attack	AMI-1
39316	SURGICAL IN..	WYOMISSING	PA	19610	Acute Care..	Heart Attack	AMI-1
35316	PRESENTATION..	ROLLA	ND	58367	Critical Access..	Heart Attack	AMI-1

Table 5.1: Hospital Dataset Sample (shortened)

Tax dataset This dataset was synthetically generated and is based on the schema provided by Bohannon et al. [15]. The dataset contains individual taxpayer records with personal identification, location details and financial information relevant for tax calculations. The dataset can be used for tax liability computations and demographic analysis across geographic US locations. A shortened sample of the dataset can be seen in Table 5.2.

FName	LName	Gender	State	Zip	HasChild	Salary	Rate	SingleEx	MarriedEx	ChildEx
Lama	Varghese	M	CT	6495	Y	21500	5.00	0	24500	0
Antonio	Delgrange	M	TN	38187	Y	92000	0.00	0	0	0
Shuetsu	Abrantes	F	AZ	86045	Y	85500	4.040	2100	0	2300
Ives	Rachui	F	VA	24348	N	47500	5.75	0	1800	0
Navid	Bos	M	MD	21722	N	32000	4.75	0	4800	0

Table 5.2: Tax Dataset Sample (shortened)

Stocks dataset This dataset contains historical daily stock trading data, tracking key price metrics, trading volume and ticker symbols across multiple trading sessions. The dataset is designed for financial analysis, including price trend analysis, volatility assessment and trading strategies. A sample of the stocks dataset can be seen in Table 5.3

Table 5.3: Stock Dataset

Date	Open	High	Low	Close	Volume	Name
2013-02-08	15.07	15.12	14.63	14.75	8,407,500	AAL
2013-02-11	14.89	15.01	14.26	14.46	8,882,000	AAL
2013-02-12	14.45	14.51	14.10	14.27	8,126,000	AAL
2013-02-13	14.30	14.94	14.25	14.66	10,259,500	AAL
2013-02-14	14.94	14.96	13.16	13.99	31,879,900	AAL

5.1.2 Rule Generation Settings

As all of the used datasets for evaluations consist only of a single-table, the bag-template generation was run with the following settings:

- **Standard bag-templates:** The standard bag-template generation was run with the **maximum bag-size** and **maximum comparison occurrences** parameters set to 2. This means that the generated bags will at most have predicates with arity 2 (e.g. `hospital(2)` or comparison atoms like `>(2)`). Our experiments show, that higher occurrences are very unlikely to yield good rules, especially for simpler and single-table datasets.
- **Extended head bag-templates:** Extended head bag-templates, which specify the head predicate or comparison atom have also been included in the bag-generation. While this does introduce some redundancy in the generated rules, our experiments show that this can actually benefit rule coverage, as the LLM then sometimes re-visits certain types of rules (e.g. functional dependencies that were already covered by standard bag-templates).
- **Sample and Model:** Evaluations were conducted using **gpt-5-mini-2025-08-07** and a fixed sample size of 30.
- **Conversation resets:** The template rule miner allows for two settings when generating rules from a list of generation prompts. It can either reset the conversation (i.e. the context) for each generation prompt, or work through all generation prompts in the same conversation thread. The first setting usually yields more rules but the output is not as consistent. The second setting reduces redundancy of the generated rules (as the LLM knows all rules generated prior to the currently processed generation prompt) but it can run out of context if many generation prompts are processed. For the evaluation runs, this parameter was set to false, thus all generation prompts were processed in the same conversation.
- **Data Understanding:** All evaluation runs include the results from the data understanding step provided by the Chat2Data system.

For the hospital dataset, the Bag-Template Generator created a list of 40 bag-templates shown in Figure 5.1.

Rule Checker Implementation To assist in the evaluation of candidate rules against golden rules during early experiments, a naive **Rule Checker** utility was implemented. This tool uses an LLM to determine semantic equivalence between rules, checking whether each golden rule is covered by at least one candidate rule. While this is a quite naive approach, experiments revealed several important considerations for reliable rule checking. First, format consistency is critical - when golden and candidate rules were expressed in the same formalism (e.g. both in Prolog notation), the checkers accuracy

Figure 5.1: List of 40 bag-templates generated for the hospital dataset.

```

[hospital(1)], [hospital(2)],
[hospital(1), ≠ (1)], [hospital(1), ≠ (2)], [hospital(2), ≠ (1)], [hospital(2), ≠ (2)],
[hospital(1), < (1)], [hospital(1), < (2)], [hospital(2), < (1)], [hospital(2), < (2)],
[hospital(1), > (1)], [hospital(1), > (2)], [hospital(2), > (1)], [hospital(2), > (2)],
[hospital(1), ≤ (1)], [hospital(1), ≤ (2)], [hospital(2), ≤ (1)], [hospital(2), ≤ (2)],
[hospital(1), ≥ (1)], [hospital(1), ≥ (2)], [hospital(2), ≥ (1)], [hospital(2), ≥ (2)],
[hospital(1), = (1)], [hospital(1), = (2)], [hospital(2), = (1)], [hospital(2), = (2)],
[head:hospital, hospital(1)], [head:hospital, hospital(2)],
[head: ≠, hospital(1)], [head: ≠, hospital(2)],
[head: <, hospital(1)], [head: <, hospital(2)],
[head: >, hospital(1)], [head: >, hospital(2)],
[head: ≤, hospital(1)], [head: ≤, hospital(2)],
[head: ≥, hospital(1)], [head: ≥, hospital(2)],
[head: =, hospital(1)], [head: =, hospital(2)]

```

improved significantly. Second, including natural language descriptions with the formal rule definitions was essential, as it provided the LLM with additional semantic context to assess equivalence even when syntactic representations differed. Third, earlier models such as *gpt-4o* struggled with this task, while *gpt-5-mini* demonstrated better accuracy. Note that this approach is not always reliable, and further systematic evaluation of its accuracy would be necessary before using it in production settings. Listing 5.1 shows the core prompt structure used for rule coverage checking.

Listing 5.1: Rule Checker Prompt Structure

```

You are an expert in checking data quality rules and denial
constraints. Your task is to check which golden rules are covered
by a list of candidate rules. A candidate rule covers a golden
rule if they are semantically equivalent or if they enforce the
same constraint, even if variable names or syntax differs.

**Task**
For each golden rule, check if it is covered by any candidate rule.
Return matches in structured JSON format.

Here are the golden rules:
<golden_rules>
{golden_rules}
</golden_rules>

```

```

Here are the candidate rules:
<candidate_rules>
Candidate 1:
Rule: {rule_formal_definition}
Description: {rule_natural_language_description}
...
</candidate_rules>

```

5.1.3 Evaluation Results

The evaluation results for the three datasets can be seen in Table 5.4. Note that the list of golden rules provided is quite small (5-10 rules per dataset). Due to that, the rule generation process with 40 bag-templates is quite overkill, and many rules generated were either redundant, overly specific or not valuable at all. We also observed that for the small set of given golden-rules, a corresponding **minimal set of bag-templates** can be constructed, which in most cases is enough to cover all the golden-rules that can be generated by the template-driven generation process. However, it is not guaranteed that all the rules that can be constructed by a given bag-template are indeed always generated. This is a major reliability limitation that has to be investigated in further research, as discussed in Subsection 5.4.1.

Dataset	(min) bag-templates	Token cost	# Rules	Time	Accuracy
hospital	[hospital(2), ≠(1)]	52k IN	297	2354s	71%
		46k OUT			
stocks	[stocks(1), <(1)]	48k IN	289	2638s	100%
	[stocks(1), >(1)]	44k OUT			
	[stocks(2), =(1)]				
tax	[tax(2)]	51k IN	321	3438s	75%
	[tax(2), ≠(1)]	61k OUT			
	[tax(2), >(1)]				
Overall	–	–	–	–	82%

Table 5.4: Evaluation results against pre-defined golden rules.

Hospital For the hospital dataset, the template-driven generation process was able to generate 71% of the provided golden rules. Additionally, Table 5.5 shows some of the generated rules that were not covered in the golden-rules, but labeled as valuable by collaborating researchers. Please note that columns unrelated to the rule (usually marked with an underline) have been removed for better visibility.

Stocks For the stocks dataset, the template-driven generation process was able to generate 100% of the provided golden rules. Additionally, Table 5.6 shows some of the generated rules that were not covered in the golden-rules, but labeled as valuable by collaborating researchers.

Generated Rule	Description
<code>:- hospital(PN), PN ≤ 0.</code>	Denies any row where the provider number is not positive.
<code>:- hospital(Phone), Phone ≤ 0.</code>	Denies any row where the phone number is not positive.
<code>:- hospital(ZIP_Code), ZIP_Code < 10000.</code>	Denies rows where the ZIP code is not a valid 5-digit number.
<code>:- hospital(ES), ES ≠ 'Yes', ES ≠ 'No'.</code>	Denies rows where emergency service is neither 'Yes' nor 'No'.
<code>:- hospital(HN), not HN ≠ "".</code>	Denies rows where the hospital name is the empty string.
<code>:- hospital(MC), not MC ≠ "".</code>	Denies rows where the measure code is the empty string.
<code>:- hospital(PN,HN1), hospital(PN,HN2), HN1 ≠ HN2.</code>	Denies inconsistent hospital names for the same provider number.
<code>:- hospital(City1,ZIP), hospital(City2,ZIP), City1 ≠ City2.</code>	Denies different city values for the same ZIP code.
<code>Sample ≠ 'nan' :- hospital(Sample).</code>	Denies rows where sample contains the literal string 'nan'.

Table 5.5: Additional rules for the hospital dataset, generated by the template-driven generation process and labeled as valuable by collaborating researchers.

Tax For the tax dataset, the template-driven generation process was able to mine 75% of the provided golden rules. Additionally, Table 5.7 shows some of the generated rules that were not covered in the golden-rules, but labeled as valuable by collaborating researchers.

Token cost

The token costs for the rule generation runs are estimated as follows:

- **Sample:** As the generation process requires providing the LLM with a sample of the dataset in question, the token cost for the sample representation depends highly on the contents of the dataset (e.g. dimensions and attribute values). For example, for the sample used for the hospital dataset, the sample representation is about **2500 tokens** (calculated with *tokencounter*¹).
- **Data Understanding:** As for the sample, the data understanding result is also highly dependent on the used dataset. For the hospital dataset, the data understanding result is about **2000 tokens**.

¹<https://tokencounter.org/>, Accessed: 2025-09-25

Generated Rule	Description
<code>:- stocks(_, Open, _, _, _, _, _), Open ≤ 0.</code>	Denies any row where the opening price is not positive.
<code>:- stocks(_, _, High, _, _, _, _), High ≤ 0.</code>	Denies any row where the day's high price is not positive.
<code>:- stocks(_, _, _, Low, _, _, _), Low ≤ 0.</code>	Denies any row where the day's low price is not positive.
<code>:- stocks(_, _, _, Low, _, _, _), Close ≤ 0.</code>	Denies any row where the day's close price is not positive.
<code>:- stocks(_, _, _, _, _, Volume, _), Volume < 0.</code>	Denies rows where trading volume is negative.
<code>:- stocks(_, _, _, _, _, _, Name), Name = "".</code>	Denies rows where the ticker name is the empty string.
<code>:- stocks(Date, _, _, _, _, _, _), not match(Date, ^[0-9]{4}-[0-9]{2}-[0-9]{2}\$).</code>	Denies rows where the date does not follow the YYYY-MM-DD format.

Table 5.6: Additional rules for the stocks dataset, generated by the template-driven generation process and labeled as valuable by collaborating researchers.

- **Generation prompts:** For each bag-template, a generation prompt is generated that is sent to the LLM after the conversation was initialized with sample data and data understanding results. The generation prompt length is only slightly influenced by the used datasets, primarily through its dimensions (e.g. how many columns the relation has). Depending on the used bag-template, the generation prompt is about **1200 tokens** for the hospital dataset. However, as the generation prompts are sent for each bag-template, this token cost has to be multiplied by the number of generation prompts. Each evaluation's generation process used **40 bag-templates**, so the total token cost of the generation prompts is **1200*40 = 48000 tokens** per dataset.
- **Output tokens:** As each generation prompt will result in an LLM response (e.g. the generated rules for that generation prompt), the output tokens also have to be considered. This depends mostly on the amount of rules generated and the dimension of the relations included in the rules. For reference, all LLM responses of the generation run for the hospital dataset sum up to about **46k output tokens**.
- **Overall token cost:** Overall, the whole generation process for the evaluation datasets takes between **50k to 60k input tokens** and about **50k output tokens** per dataset, depending on the used dataset and generated rules. With the current context window of gpt-5-mini of 400k tokens, the generation process easily fits into one context window.

Generated Rule	Description
<code>:- tax(Rate), tax(Rate), Rate < 0.</code>	Denies any record with a negative Rate value.
<code>:- tax(Rate), tax(Rate), Rate > 100.</code>	Denies any record with a Rate greater than 100.
<code>:- tax(Has, CEx), tax(Has, CEx), CEx > 0, Has \neq 'Y'.</code>	Denies a record that has a positive ChildExemp but HasChild is not 'Y'.
<code>:- tax(FName, LName, SingleExemp), tax(FName, LName, MarriedExemp), SingleExemp > 0, MarriedExemp > 0.</code>	Denies records where the same name has positive SingleExemp and positive MarriedExemp.
<code>:- tax(Zip), tax(Zip), not match(Zip, '[0-9]5\$').</code>	ZIP codes that do not match the 5-digit pattern should be flagged.
<code>:- tax(MS), MS \neq 'S', MS \neq 'M'.</code>	Denies records where Marital-Status is not 'S' (Single) or 'M' (Married).

Table 5.7: Additional rules for the tax dataset, generated by the template-driven generation process and labeled as valuable.

Overall, the evaluation shows that the template-driven generation process can generate valuable rules that match those defined by domain-experts. However, for the small set of golden-rules provided, the full generation process utilizing all 40 bag-templates is not efficient, as a minimal set of bag-templates can already cover these rules. This highlights the need for a smart pre-selection of bag-templates, if token-cost and time consumption is a limitation.

5.2 Semantic Score Evaluation

In this section, the semantic-score metric is evaluated using synthetic data and comparisons between a baseline dataset and a modified dataset that includes additional violating rows.

The semantic-score is calculated for each golden rule based on a reduced version of the hospital dataset, for our experiment we name this the *hospital-baseline* dataset. In a next step, the **Synthetic Violation Generator** is used to introduce new, genuine data errors for a randomly selected subset of golden rules. These violations are appended to the *hospital-baseline* dataset, resulting in the modified dataset *hospital-modified*. If the hypothesis stating that the semantic-score reflects a rule's capability in detecting real data errors holds, the semantic-scores should increase for the rules that were randomly sampled for synthetic violation generation, as they now should fire against more violations,

which then should be reflected in the evaluation done by the Semantic Score Judge.

The evaluation procedure is as follows:

1. Select a list of candidate rules X on baseline dataset Y
2. Calculate semantic-scores using the Semantic Score Judge on baseline dataset Y
3. Manipulate dataset $Y \rightarrow Y'$ to contain more "real" violations for a randomly selected subset of rules X'
4. Calculate semantic scores again
5. If the semantic-score can reflect a rule's ability in detecting real data errors, the semantic-scores for the selected rules X' should increase

5.2.1 Evaluation Settings

To get reasonable results from the Semantic Score Judge, the rules need a certain amount of violations that can be classified. On larger datasets, rules might have thousands of violations each. To find a middle ground between reliability and token cost, the Semantic Score Judge provides several configuration options.

- **Max violation samples:** A maximum number of violation samples that will be used for classification and thus semantic-score calculation can be set. Setting this to 50-100 is usually sufficient to get a good representation of a rule's violations. The violations will be randomly sampled from the execution result provided by the rule execution process. For this experiment, the max violation samples was set to 100, which results in every violation being evaluated for the given dataset.
- **Use batch evaluation:** This setting enables batch processing of the sampled violations. This significantly speeds up the classification process that is required to calculate the semantic-scores. The evaluation was run with this setting set to 10.
- **Sample and Model:** Evaluations were conducted using **gpt-5-mini-2025-08-07** and a fixed sample size of 30.
- **Data Understanding:** All evaluation runs include the results from the data understanding step provided by the Chat2Data system.

5.2.2 Evaluation Results

The results from Figure 5.2 demonstrate that rules with added synthetic violations (Rules 1, 3, 4, and 5) show an increase in semantic scores, with Rule 5 showing the most substantial improvement from 0.36 to 0.76. The control group rules (Rules 2, 6, and 7) that were not modified show little variety, confirming the semantic score's sensitivity to actual data quality issues.

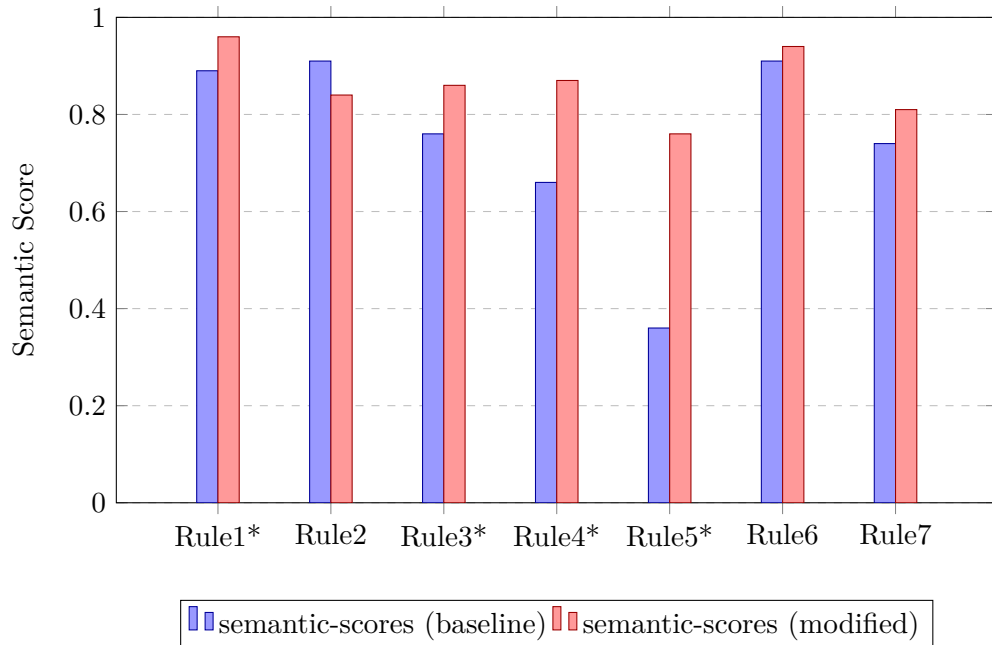


Figure 5.2: Semantic score comparison before and after adding synthetic violations. For rules 1, 3, 4, and 5 synthetic violations were added (marked with *) to the modified dataset, while for rules 2, 6, and 7 no synthetic violations were added to the modified dataset.

Quantitative Analysis Table 5.8 presents a detailed breakdown of the semantic score changes across all evaluated rules.

Rule	Baseline	Modified	Absolute Change	Relative Change	Status
Rule 1	0.89	0.96	+0.07	+7.9%	Modified*
Rule 2	0.91	0.84	-0.07	-7.7%	Control
Rule 3	0.76	0.86	+0.10	+13.2%	Modified*
Rule 4	0.66	0.87	+0.21	+31.8%	Modified*
Rule 5	0.36	0.76	+0.40	+111.1%	Modified*
Rule 6	0.91	0.94	+0.03	+3.3%	Control
Rule 7	0.74	0.81	+0.07	+9.5%	Control
Modified Rules Average: +0.195 (+41.0%)					
Control Rules Average: +0.010 (+1.7%)					

Table 5.8: Detailed semantic score changes for all rules. Modified rules show a mean improvement of 41.0%, while control rules show only 1.7% average change.

The results demonstrate that rules with added synthetic violations (Rules 1, 3, 4, and 5) show major increases in semantic scores, with an average improvement of 41.0%. Rule 5 shows the most dramatic improvement (+111.1%), increasing from 0.36 to 0.76. In

contrast, the control group rules (Rules 2, 6, and 7) show minimal average change of 1.7%, confirming that the semantic score reflects a rule’s capability in detecting real data errors. The difference between the modified rules average improvement (+41.0%) and the control group’s minimal change (+1.7%) validates the hypothesis that semantic scores are sensitive indicators of rule effectiveness in detecting genuine data errors.

Token cost

The token costs for the semantic score evaluation runs are estimated as follows:

- **Sample and Data Understanding:** As the Semantic Score Judge prompt also uses a sample representation and a (optional) data understanding result, these take up about **1900 + 2200 input tokens**.
- **Semantic Score Judge prompts:** For each batch of violations, the instructions for the Semantic Score Judge prompt that evaluates and classifies the violations is about **500 tokens**. Adding to this are the representations of the violations, which are about **400 tokens per violation** (depending on the dataset and the type of violation).
- **Output tokens:** For each batch of violations, the LLM will respond with about 1000 tokens (depending on the dataset).
- **Overall token cost:** In the above described evaluation scenario, the rules had between 30-60 violations that were judged by the LLM. Assuming an average of 45 violations, the output token cost sums up to about $45 \cdot 1000 = 45\text{k tokens}$. Input token cost sums up to $1900 + 2200 + 500 + 45 \cdot 400 = 22.6\text{k tokens}$.

5.3 Business Domain Judge Evaluation

We evaluate the Business Domain Judge with a focus on consistency and reliability. For this purpose, the Business Domain Judge evaluates the set of golden rules for the three datasets hospital, stocks and tax. During our experiments, we observed that one of the main limitations of the Business Domain Judge is its consistency (this was especially prominent on earlier models like *gpt-4o*. Running the judge multiple times with the same inputs sometimes has varying results. The proposed evaluation highlights some of the observed variations, which rules caused them and how the judge explained these decisions.

Evaluation Settings The evaluation runs were conducted using the following settings:

- **No Semantic-Score:** Due to the fact that many of the golden rules for the used datasets had no natural violations, the evaluation was done without the calculated semantic-scores. This is a limitation of this evaluation, as it would certainly be interesting to see the results combined with the proposed semantic-score.

Table 5.9: Evaluated Golden Rules on the hospital dataset

Rule ID	Description
1	Records with the same ZIP code must have the same city
2	Records with the same ZIP code must have the same state
3	Records with the same phone number must have the same ZIP code
4	Records with the same measure code must have the same measure name
5	Records with the same measure name must have the same measure code
6	Records with the same state and measure code must have the same state average reference
7	Records with the same provider number must have the same phone number

Table 5.10: Rule Evaluation Consistency of the Business Domain Judge on the hospital dataset

Rule ID	Decision	Consistency (%)	Domain Score	Quality Score	Practical Value
1	KEEP	100.0	4.0 ± 0.0	4.0 ± 0.0	4.0 ± 0.0
2	KEEP	100.0	5.0 ± 0.0	5.0 ± 0.0	4.8 ± 0.42
3	UNCERTAIN	60.0	2.7 ± 0.48	3.1 ± 0.32	2.5 ± 0.53
4	KEEP	100.0	5.0 ± 0.0	5.0 ± 0.0	5.0 ± 0.0
5	KEEP	100.0	4.2 ± 0.42	3.9 ± 0.32	4.0 ± 0.0
6	KEEP	100.0	5.0 ± 0.0	4.8 ± 0.42	4.1 ± 0.32
7	KEEP	100.0	5.0 ± 0.0	4.7 ± 0.48	4.8 ± 0.42

- **Data Understanding:** All evaluation runs are done with a prior data understanding result.
- **Batch-processing:** To speed up the evaluation process, the Business Domain Judge has the option to evaluate rules in batches. All runs are done in batches of 10 rules each, thus for most datasets only a single prompt is sent to the LLM, and all evaluations are returned in one prompt.
- **Sample and Model:** Evaluations were conducted using **gpt-5-mini-2025-08-07** and a fixed sample size of 50.

Evaluation of the Hospital Golden Rules To evaluate the consistency of the Business Domain Judge for the hospital dataset, ten independent evaluation runs were conducted on the hospital dataset using the fixed set of seven golden rules defined by domain experts. Table 5.9 lists the evaluated rules, while Table 5.10 summarizes the results of the evaluation runs.

Overall, the domain judge shows reasonable consistency on the hospital dataset, maintaining agreement on six of the seven rules across all conducted runs. The only rule showing varying results was *Rule 3* ("Records with the same phone number must have the same ZIP code"), where the judge alternated between *DISCARD* and *UNCERTAIN* decisions. This is quite interesting, as this variation comes from uncertainties in the given

Table 5.11: Evaluated Golden Rules on the stocks dataset

Rule ID	Description
1	The opening price must be greater than or equal to the day's lowest price.
2	The day's lowest price must be less than or equal to the closing price.
3	The day's highest price must be greater than or equal to the day's lowest price.
4	The day's highest price must be greater than or equal to the closing price.
5	The opening price must be less than or equal to the day's highest price.
6	No duplicate records can exist for the same date and ticker combination.

Table 5.12: Rule Evaluation Consistency of the Business Domain Judge on the stocks dataset

Rule ID	Decision	Consistency (%)	Domain Score	Quality Score	Practical Value
1	KEEP	100.0	5.0 ± 0.00	4.9 ± 0.32	4.8 ± 0.42
2	KEEP	100.0	5.0 ± 0.00	4.8 ± 0.42	4.8 ± 0.42
3	KEEP	100.0	5.0 ± 0.00	5.0 ± 0.00	5.0 ± 0.00
4	KEEP	100.0	5.0 ± 0.00	4.8 ± 0.42	4.6 ± 0.70
5	KEEP	100.0	5.0 ± 0.00	4.8 ± 0.42	4.6 ± 0.52
6	KEEP	100.0	5.0 ± 0.00	3.8 ± 0.92	5.0 ± 0.00

domain. The judge consistently reflected this uncertainty in its scoring and reasoning results, pointing out that there are reasonable business scenarios where shared numbers span across multiple ZIP codes (e.g. centralized call centers, health system-wide contact numbers and so on).

Evaluation of the Stocks Golden Rules To evaluate the consistency of the Business Domain Judge for the stocks dataset, ten independent runs were conducted using the fixed set of six golden rules. Table 5.11 lists the evaluated rules, while Table 5.12 summarizes the results of the evaluation runs.

Overall, the domain judge shows perfect consistency on the evaluation of the golden rules for the stocks dataset, maintaining agreement on all six rules across all conducted runs. Each rule consistently received the KEEP decision with 100% consistency, reflecting the clear price-relationship logic checked by these rules. Slight variations in the reported quality and practical value scores reflect small differences in the provided justifications across the conducted runs rather than a conceptual disagreement.

Evaluation of the Tax Golden Rules To evaluate the reliability of the domain-specific rule judge on the tax dataset, ten independent runs were conducted using the fixed set of eight golden rules provided by domain experts. Table 5.13 lists the evaluated rules, while Table 5.14 summarizes the results of the evaluation runs.

Table 5.13: Evaluated Golden Rules on the tax dataset

Rule ID	Description
1	No two records can have the same area code and phone number but belong to different cities.
2	Records with the same ZIP code must have the same city.
3	Records with the same area code must have the same state.
4	Records with the same ZIP code must have the same state.
5	Taxpayers in the same state with the same child exemption status must have consistent child tax credits.
6	Taxpayers with identical IDs must have the same reported income.
7	Taxpayers with identical names and states must have the same ZIP code.
8	In the same state, taxpayers with higher salaries should not have lower tax payments.

Table 5.14: Rule Evaluation Consistency of the Business Domain Judge on the tax dataset

Rule ID	Decision	Consistency (%)	Domain Score	Quality Score	Practical Value
1	DISCARD	80.0	2.5 ± 0.53	3.4 ± 0.52	2.2 ± 0.42
2	KEEP	60.0	3.1 ± 0.99	3.6 ± 0.70	3.0 ± 0.94
3	DISCARD	80.0	2.5 ± 0.53	3.4 ± 0.52	2.2 ± 0.42
4	KEEP	100.0	4.8 ± 0.42	4.7 ± 0.48	4.8 ± 0.42
5	DISCARD	100.0	1.8 ± 0.42	2.4 ± 0.52	1.7 ± 0.48
8	UNCERTAIN	80.0	2.9 ± 0.57	3.2 ± 0.63	3.0 ± 0.67

Overall, the Business Domain Judge showed moderate consistency on the tax dataset, maintaining consistent decisions on several structurally sound rules such as *Rule 4* (“Records with the same ZIP code must have the same state”). Rules grounded in standard address normalization (e.g. ZIP code, city and state relationships) show 100% agreement across all runs. However, rules tied to behavioral or numerical relationships, such as *Rule 8* (“In the same state, taxpayers with higher salaries should not have lower tax payments”), displayed uncertainty or only partial agreement. Interestingly, the rule generation runs on the tax dataset had similar varying results, especially on the rules that were likely to be discarded by the Business Domain Judge. It seems like the LLMs are missing some important domain knowledge for this dataset, which could be addressed by adding a more complete data understanding result or some additional metadata for the dataset.

Evaluation Summary The Business Domain Judge evaluations across three datasets show that judge consistency varies with rule complexity and domain clarity. The judge achieved perfect consistency (100%) on structurally sound rules that align with clear domain logic (e.g. as defined by the data understanding result or assumed by the LLM based on the sample), such as address normalization constraints and price relationship rules in the stocks dataset. However, reliability decreased for rules involving behavioral patterns or domain-specific business logic. For example, evaluation runs on the tax dataset showed varying results where several rules only had 60-80% consistency in their

decisions. Interestingly, inconsistencies often reflected genuine domain ambiguities rather than judge failures. For example, *Rule 3* ("*Records with the same phone number must have the same ZIP code*") in the hospital dataset alternated between *UNCERTAIN* and *DISCARD* because shared phone numbers across ZIP codes represent a legitimate edge case in healthcare systems.

These findings suggest that the Business Domain Judge serves as an effective filtering mechanism for clearly inappropriate rules while appropriately flagging rules that require human domain expertise for final validation. The observed variations in judge decisions, particularly on the tax dataset, also highlight potential gaps in the LLMs ability to understand domain knowledge, which could be addressed through enhanced data understanding results or additional domain-specific metadata.

5.4 Limitations and Challenges

In this section limitations and challenges of the rule generation and evaluation process are discussed. As the template-driven rule generation process is the main contribution of this work, the focus is set on the limitations of the generation process in Subsection 5.4.1. Some limitations of the rule evaluation components are discussed in 5.4.2.

5.4.1 Limitations of the Rule Generation

This subsection covers some of the observed limitations of the rule generation process and discusses potential solutions.

Reliability of the Generation process A major limitation of the template-driven generation process is completeness and reliability. Experiments show that the same set of bag-templates does not always generate the same rules. This is a known issue with LLMs and their inherent non-determinism. Introducing bag-templates that are syntactically different, but usually match to the same rules in slightly different formats (e.g. DCs created by the bag-template `hospital(2), ≠(1)` can match rules created by the bag-template `head:=, hospital(2)` can help in increasing reliability. Another option to tackle completeness is re-running the generation process multiple times, however this would require a strategy to find and filter out redundant rules.

Minimal set of bag-templates Enumerating many bags and prompting the LLMs for each generation prompt is both token and time consuming. The conducted experiments show that a smart selection of the used bag-templates can already cover a wide range of rules and dependencies. For example, using bag-templates with comparison atoms might not be needed at all for datasets without numerical values. For multi-table datasets, the used bag-templates might only enumerate combinations of relations that likely have dependencies between each other. The evaluations of the three datasets show, that the minimal set of bag-templates is already capable of covering many of the golden rules (and later runs on more complex bag-templates in many cases resulted in no additional

valuable rules). However, it should be noted that the used datasets in the generation evaluation were not that complex, so in cases where the datasets are more sophisticated or have higher dimensions, more complex bag-templates might still be needed to generate valuable rules.

Parallelizing the generation process In order to speed up the generation process, a parallelization could be implemented. However, as this requires multiple conversation threads in which the generations occur, one has to make sure that redundant rules are not included in the result set. For this purpose, a reasoning LLM could be used to check if a given rule X is in a given set of rules. For the evaluation of the generation process, a similar, quite naive system was implemented that showed promising results. However, reliability of such a redundancy check system would need to be systematically evaluated.

Context Limitations The rule generation process currently generates rule in Prolog-like syntax. While this format has some benefits (e.g. expressiveness), it also has some downsides. High-dimensional data can be problematic as fitting it into context for sample data might introduce needle-in-the-haystack limitations inherent in LLMs (e.g. in gene expression datasets). High-dimensional data is also more complex to represent in the generation prompt and the resulting Prolog rules, however this could be addressed by suppressing unnecessary columns.

5.4.2 Limitations of the LLM Judges

Rules without natural violations One downside of the semantic-score implementation is its dependence on natural rule violations occurring in the dataset. Even though a synthetic violation generator can be used as a fallback in these cases, it remains unclear whether or not the calculated semantic-score (based on synthetic violations) actually reflects a rule's capability in detecting real data errors. Further evaluations are necessary to find out if this approach brings additional insights on rules without natural violations.

Judge Reliability Furthermore, reliability is still an area that needs further investigation. Both the Semantic Score Judge and the Business Domain Judge had varying results in our conducted experiments. This is to some degree stemming from the inherent randomness that modern LLMs have. A potential solution could be to add few-shot examples to the corresponding prompts, so that the LLMs know in more detail what the application expects from the respective judge. Experimenting with few-shot variations of the prompts could be a potential future research direction.

Evaluation of the Business Domain Judge Filtering As mentioned in Subsection 4.4.2, the Business Domain Judge was evaluated with a focus on reliability and consistency. A systematic evaluation of the Business Domain Judge ability in processing and filtering many candidate rules could yield additional insights on its limitations.

Conclusion and Future Work

6.1 Summary

This thesis proposed and evaluated a novel framework for automated data quality rule generation and evaluation using LLMs. The work addressed fundamental limitations of both traditional rule mining approaches and naive LLM-based methods through two primary contributions, a template-driven rule generation process and a semantic evaluation procedure.

The template-driven generation approach systematically guides LLMs through structured bag-templates that specify rule complexity and shape, addressing the inherent unreliability and incompleteness of naive prompting strategies. Evaluation on three datasets demonstrated that this approach successfully generated between 75-100% of expert-defined golden rules while also discovering additional valuable constraints not captured in the original golden rules. For the stocks dataset, the framework achieved complete coverage of golden rules, while maintaining strong performance (71-75%) on the hospital and tax datasets.

The second major contribution introduced a novel semantic-score metric that measures rule effectiveness through LLM-based classifications of violations as either genuine data errors or false positives. Validation experiments using synthetic violations demonstrated that the metric reliably reflects a rule's capability in detecting real data errors, showing an average 41% improvement for rules with added violations compared to only 1.7% variation in control rules. This semantic evaluation, combined with the Business Domain Judge for filtering domain-inappropriate rules, provides a comprehensive evaluation framework beyond traditional statistical metrics.

However, the research also revealed important limitations. The template-driven generation approach still has reliability and completeness limitations, as identical bag-templates can produce varying rules across multiple runs due to inherent LLM non-determinism.

The enumeration of bag-templates also comes with scalability challenges, requiring an intelligent pre-selection strategy to balance coverage and computational cost. The semantic-score's dependence on natural violations presents challenges for rules without violations. The synthetic violation generation offers a potential workaround for this problem, but this approach requires further systematic evaluation.

6.1.1 Research Questions

These findings directly address the research questions that motivated this work.

1. **RQ1 (LLM effectiveness in generating meaningful rules):** The evaluation demonstrated that modern LLMs can successfully generate semantically meaningful and syntactically valid data quality rules when provided with structured guidance, achieving 75-100% coverage of expert-defined rules across three datasets.
2. **RQ2 (Template-driven vs. Knowledge-driven approaches):** The template-driven method proved capable of discovering rules outside predefined categories while maintaining systematic coverage. However, as this process generates many rule candidates, subsequent rule evaluation and filtering is required to create a concise rule set. The comparison suggests these approaches are complementary rather than mutually exclusive. Revisiting this research question and conducting an empirical evaluation and comparison of both approaches would be an interesting future research area.
3. **RQ3 (LLM-as-a-Judge reliability):** The semantic-score validation experiments confirmed that LLM judges can provide reliable complementary metrics beyond traditional statistical measures, with the semantic-score metric successfully signaling on rules that detect genuine errors. The Business Domain Judge also showed promising capabilities in evaluating rule quality on defined criteria.
4. **RQ4 (Reducing expert reliance):** The framework demonstrates that automated generation and evaluation can significantly reduce the need for domain expertise. However, experts remain valuable for validating generated rules and providing the initial golden rules used for framework evaluation, although the time investment required is substantially reduced compared to manual rule generation from scratch.

6.2 Future Research

Multi-table rule generation and evaluation While this thesis focused the evaluation on single-table datasets, experiments during development showed promising capabilities of the template-driven rule generation on multi-table datasets. Future work should systematically evaluate the generation approach on complex multi-relational schemas. Key challenges include intelligent pre-selection of relation combinations to avoid combinatorial explosion, preservation of foreign key relationships and cross-table dependencies during

sampling and a systematic evaluation of the generated rules. The data understanding component could be extended to identify likely relationships between tables, guiding the bag-template generation into generating a more reasonable combination of bag-templates.

Hybrid generation approaches Early experiments where we compared results from a knowledge-driven generation with the results from the template-driven generation process show some promising capabilities to tackle reliability and completeness. If implemented correctly, these two processes could complement each other for a more complete set of generated rules. Knowledge-driven approaches excel at generating specific constraint types (e.g. functional dependencies) with high reliability, while the template-driven approach could focus on multi-table rule generation and on discovering more complex or novel patterns outside the predefined categories. Challenges when combining the two approaches include the orchestration strategy, whether or not the template-driven results could generate signals to expand the established knowledge-bases and how de-duplication strategies can be employed to effectively merge the outputs from both approaches. As for the evaluation methods employed in this work, both the semantic-score and the Business Domain Judge can also be employed for rules generated by a knowledge-driven approach.

Reliability and completeness guarantees A critical limitation is the non-deterministic nature of LLM-based generations. Future research could explore multiple generation runs, formal redundancy detection using reasoning models or symbolic approaches and adaptive generation strategies that check for missing constraint types. Understanding the theoretical bounds of what bag-templates can express and whether certain rule families inherently require different generation strategies would strengthen the frameworks foundation.

Fine-tuned specialized models Rather than relying only on general-purpose LLMs, domain-specific fine-tuning could improve both generation and evaluation. A training corpus of dataset and golden-rule pairs could be constructed to fine-tune generation models. Research should assess whether smaller fine-tuned models can match or exceed larger general models for these specialized tasks, potentially reducing costs and latency.

Another interesting topic could be the use of RAG techniques for the generation process and calculating the semantic-score. An agentic LLM system could be provided with tools to automatically access the dataset in question. This could allow the LLM to autonomously lookup violating data or expand beyond the provided sample in both generation and evaluation prompts.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Overview of Generative AI Tools Used

Generative AI tools have been used to grammar check certain parts of this work. They have also been used to brain-storm and structure outlines of the theoretical background chapter. Some paragraphs have been rephrased using generative AI tools to increase the academic tone while preserving the original content and argumentation. Furthermore, generative AI tools have been used to format and create certain LaTeX objects such as tables, graphs and listings. For example, Table 5.8, Table 4.2 and Figure 5.2 have been created using Claude with data I gathered during my experiments.

Additionally, generative AI tools have been used to support in debugging and implementation throughout the development of the proposed framework. For instance, Claude assisted in refactoring code to asynchronous implementations that enable efficient concurrent operations.

The main tools used in this work are as follows:

- **Claude**[7]: Coding and formatting (Accessed between 1st February 2025 to 1st October 2025)
- **ChatGPT**[102]: Coding and formatting (Accessed between 1st February 2025 to 1st October 2025)
- **Perplexity AI**[107]: Literature Research (Accessed between 1st February 2025 to 1st October 2025)



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1	Overview of machine learning categories [110]	8
2.2	Figure of a Neural network consisting of an input, hidden and output layer by Prince [110]	10
2.3	Visualization of word clustering in embedding space [97]	14
2.4	The Transformer Model (Encoder-Decoder) as proposed by Vaswani et al. [131]	18
2.5	GPT-3 performance scaling on TriviaQA by Brown et al. [18]	19
2.6	Overview of reasoning strategies based on Kumar et al. [76]	25
2.7	Categorization of traditional ASM algorithms by Hipp et al. [59]	41
4.1	High level system architecture diagram of the proposed framework.	50
4.2	System architecture diagram of the proposed Rule Generation implementation.	55
4.3	Growth of the number of generated bag-templates with an increasing number of relations. Left: without comparison operators, Right: with comparison operators. Configuration: maximum bag size = 3, maximum database predicates = 3.	59
4.4	Rule Compilation and Execution system architecture. The rule compiler was provided by the Chat2Data system.	62
4.5	System architecture design of the Semantic Score Judge.	65
5.1	List of 40 bag-templates generated for the hospital dataset.	74
5.2	Semantic score comparison before and after adding synthetic violations. For rules 1, 3, 4, and 5 synthetic violations were added (marked with *) to the modified dataset, while for rules 2, 6, and 7 no synthetic violations were added to the modified dataset.	80

List of Tables

2.1	Comparison of Common Activation Functions	9
2.2	Example data for city and zipcode CFDs	37
4.1	Competitors database schema [47]	56
4.2	Growth of the number of generated bag-templates with an increasing number of relations and increasing max bag-size. Configuration: maximum database predicates = 3.	59
5.1	Hospital Dataset Sample (shortened)	72
5.2	Tax Dataset Sample (shortened)	72
5.3	Stock Dataset	72
5.4	Evaluation results against pre-defined golden rules.	75
5.5	Additional rules for the hospital dataset, generated by the template-driven generation process and labeled as valuable by collaborating researchers.	76
5.6	Additional rules for the stocks dataset, generated by the template-driven generation process and labeled as valuable by collaborating researchers.	77
5.7	Additional rules for the tax dataset, generated by the template-driven generation process and labeled as valuable.	78
5.8	Detailed semantic score changes for all rules. Modified rules show a mean improvement of 41.0%, while control rules show only 1.7% average change.	80
5.9	Evaluated Golden Rules on the hospital dataset	82
5.10	Rule Evaluation Consistency of the Business Domain Judge on the hospital dataset	82
5.11	Evaluated Golden Rules on the stocks dataset	83
5.12	Rule Evaluation Consistency of the Business Domain Judge on the stocks dataset	83
5.13	Evaluated Golden Rules on the tax dataset	84
5.14	Rule Evaluation Consistency of the Business Domain Judge on the tax dataset	84



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [1] Github code repositories. Open Source Software Platform, 2023. Accessed: 2025-02-25.
- [2] Reddit. Social Media Platform, 2023. Accessed: 2025-02-25.
- [3] Wikipedia. Online Encyclopedia, 2023. Accessed: 2025-02-25.
- [4] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
- [5] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [6] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A Inkeri Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328, 1996.
- [7] Anthropic. Claude. <https://claude.ai>, 2024. Accessed between 2025-02-01 and 2025-10-01.
- [8] Anthropic, PBC. Anthropic. <https://www.anthropic.com>, 2023. Accessed: 2025-02-25.
- [9] Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [10] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- [11] Yushi Bai, Jiahao Ying, Yixin Cao, Xin Lv, Yuze He, Xiaozhi Wang, Jifan Yu, Kaisheng Zeng, Yijia Xiao, Haozhe Lyu, et al. Benchmarking foundation models with language-model-as-an-examiner. *Advances in Neural Information Processing Systems*, 36:78142–78167, 2023.

- [12] Caroline Bassett. The computational therapeutic: exploring weizenbaum’s eliza as a history of the present. *AI & SOCIETY*, 34(4):803–812, 2019.
- [13] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.
- [14] Reuben Binns. Fairness in machine learning: Lessons from political philosophy. In *Conference on fairness, accountability and transparency*, pages 149–159. PMLR, 2018.
- [15] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietidis. Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 746–755, 2007.
- [16] Noémi Bontridder and Yves Poulet. The role of artificial intelligence in disinformation. *Data & Policy*, 3:e32, 2021.
- [17] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010: 19th International Conference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.
- [18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [19] Joe Burton and Simona R Soare. Understanding the strategic implications of the weaponization of artificial intelligence. In *2019 11th international conference on Cyber Conflict (CyCon)*, volume 900, pages 1–17. IEEE, 2019.
- [20] Joseph Carlsmith. Is power-seeking ai an existential risk? *arXiv preprint arXiv:2206.13353*, 2022.
- [21] Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. What you always wanted to know about datalog (and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.
- [22] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. Legal-bert: The muppets straight out of law school. *arXiv preprint arXiv:2010.02559*, 2020.
- [23] Fei Chiang and Renée J Miller. Discovering data quality rules. *Proceedings of the VLDB Endowment*, 1(1):1166–1177, 2008.

- [24] KR Chowdhary. *Fundamentals of artificial intelligence*. Springer, 2020.
- [25] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.
- [26] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.
- [27] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 458–469. IEEE, 2013.
- [28] Xu Chu, Ihab F Ilyas, Paolo Papotti, and Yin Ye. Ruleminer: Data quality rules discovery. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1222–1225. IEEE, 2014.
- [29] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [30] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.
- [31] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [32] Common Crawl Foundation. Common Crawl. <https://commoncrawl.org/>, 2023. Accessed: [Insert access date here].
- [33] Roberto Confalonieri, Ludovik Coba, Benedikt Wagner, and Tarek R Besold. A historical perspective of explainable artificial intelligence. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(1):e1391, 2021.
- [34] Andrew Critch and David Krueger. Ai research considerations for human existential safety (arches). *arXiv preprint arXiv:2006.04948*, 2020.
- [35] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.
- [36] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [37] Jacob Eisenstein. Natural language processing. *Jacob Eisenstein*, 507, 2018.
- [38] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

- [39] Ulle Endriss. Lecture notes an introduction to prolog programming. *Lectured on September, 24, 2000*.
- [40] Ernst & Young. Venture capital investment in generative ai almost doubles globally in 2024 as momentum accelerates in transformative sector, 2024. Accessed: 2025-02-19.
- [41] Wenfei Fan and Floris Geerts. *Foundations of data quality management*. Morgan & Claypool Publishers, 2012.
- [42] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 23(5):683–698, 2010.
- [43] Wenqi Fan, Yujian Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501, 2024.
- [44] William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.
- [45] Iason Gabriel and Vafa Ghazavi. The challenge of value alignment. In *The Oxford Handbook of Digital Ethics*. Oxford University Press Oxford, 2022.
- [46] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2, 2023.
- [47] Georg Gottlob, Jinsong Guo. Automatic database enrichment and curation using large language models. <https://patents.google.com/patent/US20250045256A1/en>, 2024. Accessed: 2025-09-25.
- [48] Jose Manuel Gomez-Perez, Ronald Denaux, Andres Garcia-Silva, Jose Manuel Gomez-Perez, Ronald Denaux, and Andres Garcia-Silva. Understanding word embeddings and language models. *A Practical Guide to Hybrid Natural Language Processing: Combining Neural Models and Knowledge Graphs for NLP*, pages 17–31, 2020.
- [49] Ian Goodfellow. Deep learning, 2016.
- [50] Google LLC. Google. <https://www.google.com>, 2023. Accessed: 2025-02-25.
- [51] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

- [52] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. Xai—explainable artificial intelligence. *Science robotics*, 4(37):eaay7120, 2019.
- [53] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [54] Thilo Hagendorff. The ethics of ai ethics: An evaluation of guidelines. *Minds and machines*, 30(1):99–120, 2020.
- [55] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.
- [56] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12, 2000.
- [57] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.
- [58] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ml safety. *arXiv preprint arXiv:2109.13916*, 2021.
- [59] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining—a general survey and comparison. *ACM sigkdd explorations newsletter*, 2(1):58–64, 2000.
- [60] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.
- [61] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [62] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [63] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

- [64] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [65] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.
- [66] Lance Y Hunter, Craig D Albert, Josh Rutland, Kristen Topping, and Christopher Hennigan. Artificial intelligence and information warfare in major power states: how the us, china, and russia are using artificial intelligence in their information warfare and influence operations. *Defense & Security Analysis*, pages 1–35, 2024.
- [67] Jiaming Ji, Tianyi Qiu, Boyuan Chen, Borong Zhang, Hantao Lou, Kaile Wang, Yawen Duan, Zhonghao He, Jiayi Zhou, Zhaowei Zhang, et al. Ai alignment: A comprehensive survey. *arXiv preprint arXiv:2310.19852*, 2023.
- [68] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [69] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [70] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.
- [71] Sujit Khanna and Shishir Subedi. Tabular embedding model (tem): Finetuning embedding models for tabular rag applications. *arXiv preprint arXiv:2405.01585*, 2024.
- [72] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [73] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.
- [74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [75] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

- [76] Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Salman Khan, and Fahad Shahbaz Khan. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*, 2025.
- [77] John Ladley. *Data governance: How to design, deploy, and sustain an effective data governance program*. Academic Press, 2019.
- [78] Hoa Thi Le, Angela Bonifati, and Andrea Mauri. Graph consistency rule mining with llms: an exploratory study. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 748–754. OpenProceedings.org, 2025.
- [79] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [80] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [81] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.
- [82] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
- [83] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- [84] Rumeng Li, Xun Wang, and Hong Yu. Llamacare: An instruction fine-tuned large language model for clinical nlp. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 10632–10641, 2024.
- [85] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.
- [86] Yiheng Liu, Hao He, Tianle Han, Xu Zhang, Mengyuan Liu, Jiaming Tian, Yutong Zhang, Jiaqi Wang, Xiaohui Gao, Tianyang Zhong, et al. Understanding llms: A comprehensive overview from training to inference. *arXiv preprint arXiv:2401.02038*, 2024.

- [87] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- [88] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.
- [89] Shayne Longpre, Robert Mahari, Anthony Chen, Naana Obeng-Marnu, Damien Sileo, William Brannon, Niklas Muennighoff, Nathan Khazam, Jad Kabbara, Kartik Perisetla, et al. The data provenance initiative: A large scale audit of dataset licensing & attribution in ai. 2023.
- [90] Ludovica Marinucci, Claudia Mazzuca, and Aldo Gangemi. Exposing implicit biases and stereotypes in human and artificial intelligence: state of the art and challenges with a focus on gender. *AI & SOCIETY*, 38(2):747–761, 2023.
- [91] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.
- [92] Menlo Ventures. The state of generative ai in the enterprise (2024), 2024. Accessed: 2025-02-19.
- [93] Meta Platforms, Inc. Meta. <https://meta.com>, 2023. Accessed: 2025-02-25.
- [94] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*, 2021.
- [95] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [96] Marvin Minsky and Seymour A Papert. *Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry*. MIT press, 2017.
- [97] Piero Molino, Yang Wang, and Jiawei Zhang. Parallax: Visualizing and understanding the semantics of embedding spaces via algebraic formulae. *arXiv preprint arXiv:1905.12099*, 2019.
- [98] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- [99] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

- [100] Zoëy Nguyen, Anthony Annunziata, Vinh Luong, Sang Dinh, Quynh Le, Anh Hai Ha, Chanh Le, Hong An Phan, Shruti Raghavan, and Christopher Nguyen. Enhancing q&a with domain-specific fine-tuning and iterative reasoning: A comparative study. *arXiv preprint arXiv:2404.11792*, 2024.
- [101] Franz Josef Och and Hermann Ney. Statistical machine translation. In *5th EAMT Workshop: Harvesting Existing Resources*, 2000.
- [102] OpenAI. Chatgpt. <https://chat.openai.com>, 2022. Accessed between 2025-02-01 and 2025-10-01.
- [103] OpenAI. Openai. <https://openai.com>, 2023. Accessed: 2025-02-25.
- [104] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- [105] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.
- [106] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [107] Perplexity AI. Perplexity ai. <https://www.perplexity.ai>, 2022. Accessed between 2025-02-01 and 2025-10-01.
- [108] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.
- [109] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.
- [110] Simon JD Prince. *Understanding deep learning*. MIT press, 2023.
- [111] Project Gutenberg. Project Gutenberg. <https://www.gutenberg.org/>. Accessed: [Insert access date here].

- [112] Danrui Qi, Zhengjie Miao, and Jiannan Wang. Cleanagent: Automating data standardization with llm-based agents. *arXiv preprint arXiv:2403.08291*, 2024.
- [113] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *iee assp magazine*, 3(1):4–16, 1986.
- [114] Alec Radford. Improving language understanding by generative pre-training. 2018.
- [115] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.
- [116] Kylie Robison. Openai cofounder ilya sutskever says the way ai is built is about to change. *The Verge*, 12 2024. Accessed: March 24, 2025.
- [117] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [118] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [119] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [120] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.
- [121] Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. The prompt report: a systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608*, 2024.
- [122] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [123] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
- [124] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.
- [125] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35:19523–19536, 2022.

- [126] I Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.
- [127] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.
- [128] Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q Tran, Dani Yogatama, and Donald Metzler. Scaling laws vs model architectures: How does inductive bias influence scaling? *arXiv preprint arXiv:2207.10551*, 2022.
- [129] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [130] Trieu H Trinh and Quoc V Le. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*, 2018.
- [131] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [132] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [133] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.
- [134] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [135] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.
- [136] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [137] Muhammad Mudassar Yamin, Mohib Ullah, Habib Ullah, and Basel Katt. Weaponized ai for cyber attacks. *Journal of Information Security and Applications*, 57:102722, 2021.

- [138] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.
- [139] Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.
- [140] Wenhao Yu, Hongming Zhang, Xiaoman Pan, Kaixin Ma, Hongwei Wang, and Dong Yu. Chain-of-note: Enhancing robustness in retrieval-augmented language models. *arXiv preprint arXiv:2311.09210*, 2023.
- [141] Shengbin Yue, Wei Chen, Siyuan Wang, Bingxuan Li, Chenchen Shen, Shujun Liu, Yuxuan Zhou, Yao Xiao, Song Yun, Xuanjing Huang, et al. Disc-lawllm: Fine-tuning large language models for intelligent legal services. *arXiv preprint arXiv:2309.11325*, 2023.
- [142] Jizhi Zhang, Keqin Bao, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. Is chatgpt fair for recommendation? evaluating fairness in large language model recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 993–999, 2023.
- [143] Mingtian Zhang, Shawn Lan, Peter Hayes, and David Barber. Mafin: Enhancing black-box embeddings with model augmented fine-tuning. *arXiv preprint arXiv:2402.12177*, 2024.
- [144] Shuo Zhang, Zezhou Huang, and Eugene Wu. Data cleaning using large language models. In *2025 IEEE 41st International Conference on Data Engineering Workshops (ICDEW)*, pages 28–32. IEEE, 2025.
- [145] Qiankun Zhao and Sourav S Bhowmick. Association rule mining: A survey. *Nanyang Technological University, Singapore*, 135:18, 2003.
- [146] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
- [147] Mingqian Zheng, Jiaxin Pei, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. When” a helpful assistant” is not really helpful: Personas in system prompts do not improve performances of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15126–15154, 2024.
- [148] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, 2023.

- [149] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sonia Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 19–27, 2015.
- [150] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.