Automatic Encoding From Natural Language to First-Order Horn Clauses

Emma Bjørkås

Dept. of Information Science and Media Studies
University in Bergen
Bergen, Norway

Marija Slavkovik

Supervisor

Abstract

A central topic within the field of machine ethics, and other fields where moral reasoning is needed, is to incorporate large quanta of (moral) rules represented in natural language using a formal specification that can be used for reasoning. The problem lies in the difficulty of automatically encoding natural language to logic. The task demands a need for knowledge engineers to manually encode moral rules, a solution that is expensive, with a low degree of efficiency and scalability. With the introduction of large language models comes the opportunity to explore the advantages they may provide in this task. We investigate how we can create a system able to automatically encode natural language norms to logic using GPT-4 as the main encoding element. We use norms from the Commonsense Norm Bank from the literature which contains a collection of both common sense and counter-intuitive norms and manually analyse how well GPT-4 is able to encode these norms to first-order logic (FOL). Additionally, we convert all FOL-encoded formulas to Horn clauses and analyse what types of norms that cannot be expressed in Horn. We find that our encoding system is able to encode 51% of the norms accurately to FOL in terms of syntactic properties and semantic faithfulness. We find that the mistakes GPT-4 makes for the remaining 49% of the encodings are numerous and of varying degrees of predictability and severity. We show how the implementation of a sentiment analysis model can serve as an mitigation tool to one of these distinct mistakes.

Contents

Acknowledgments

First and foremost, I would like to thank my supervisor Marija Slavkovik for always providing encouragement, expertise, and an endless supply of gaming anecdotes that I never understood. I would like to express my gratitute to Luca Fossen for his help in creating the NL norm dataset and faithfulness classifications. Also, thank you to my brother, Eivind Bjørkås, and Ana Ozaki for taking the time to suggest helpful insights and correct stupid formulations. :)

Thank you to Vanessa, Didrik, Anton, Kristian (and sometimes Snorre) in room 645 for being the silliest gooses and smartest rubber ducks. Also, thank you to the rest of the "pauserommet" people for creating fun and lighthearted breaks in a stressful time. Love you all. <3

I acknowledge the use of OpenAI's ChatGPT 4[1] and the academic writing tool Writeful[2] to explain concepts for my own understanding and refine the academic language of this thesis. The language refinement was performed by instructing the tools to rephrase certain blocks of text or accepting Writeful's automatic language suggestions. In the following paragraphs I provide an example that demonstrates the use of these tools.

I gave ChatGPT-4 the following instruction. "Rephrase this paragraph to fit into a related work chapter intro for a master thesis: In this chapter we explain the current research done in the relevant topics. Specifically when it comes to how LLMs are used in NLP task and the topics of logic encoding and prompt engineering. Additionally, we include a summary of relevant research within the field of moral conflict to underline the problem of conflicting norms in autonomous systems."

The model's response was: "This chapter provides an overview of existing research pertinent to the themes addressed in this thesis. It delves into the utilization of large language models (LLMs) in natural language processing (NLP) tasks, with particular attention on logic encoding and prompt engineering. Furthermore, the chapter includes a review of studies related to moral conflicts, highlighting the issue of conflicting norms in autonomous systems, which underscores the significance of this research within the broader field."

Elements of this response was incorporated into the paragraph seen in Chapter 3.

[1] https://chatgpt.com
[2] https://www.writefull.com/

Chapter 1

Introduction

1.1 Problem Description and Motivation

Artificial intelligence (AI) is embedded in our daily lives, whether it is clear to those who interact with it or not. Many devices are increasingly becoming more capable of unsupervised operation [Zhang and Lu, 2021]. Your watch knows when you should relax and your rice cooker can determine when the cooking is complete. As machines gain autonomy, they also play roles in ethical decision-making, essentially functioning as moral arbiters. Is it acceptable that your streaming service decides what is inappropriate to show to your children? Is it acceptable for your refrigerator to deny you access to late-night snacks based on health data from your bathroom scale? Who decides what is the right thing for a machine to do? These are some of the questions that are addressed within the field of machine ethics [Anderson and Anderson, 2007].

One of the objectives of machine ethics research is to automate ethical behavior by machines and other artificial agents. Machine ethics lies in the intersection between moral philosophy and computer science, as it deals with determining *what* ethical theories should be developed or implemented in moral machines, as well as *how* to implement them in a way that ensures ethical machine behavior [Tolmeijer et al., 2021, Slavkovik, 2022].

As more and more aspects of our daily lives are influenced by machine-made decisions and the scope and importance of the machine-made decisions increases, there is an increasing need to consider the moral foundation these decisions are based on. By introducing carefully considered and well-implemented moral foundations in autonomous machines, we can limit the negative consequences that follow from their actions that ultimately affect people. As we are able to make improvements in machine ethics, it will create room for incorporating machines in tasks that are today seen as too risky to be performed by machines. When making decisions, a machine is often faster, better able to calculate and compare consequences of actions, more consistent, and more efficient than people [Conitzer, 2023]. By improving machine ethics, qualities could be utilized in tasks that depend on ethical decision making, such as self-driving cars or automatic diagnosis of diseases [Zhang and Lu, 2021]. Research in machine ethics is, in other words, crucial to improving the moral groundwork of the machines already implemented into our daily life, as well as to further explore the numerous possibilities of machine involvement that lie ahead. The problem lies in how we should build these machines to perform these tasks.

There are two main architectural approaches for implementing ethical machines, these being top-down of ethical theories and bottom-up approaches. A top-down approach to machine ethics means to capture the essence of moral judgment through some principles, rules, or goals against which all moral situations can be considered [Wallach et al., 2008]. The problem with top-down theories, such as the classic consequentialist and deontological approaches, or even public laws, is that they are rarely able to cover all aspects that may require moral judgment. People make moral decisions every day, sometimes guided by public law and sometimes guided by social norms. Social norms exist in areas not covered by public law, such as the intimate and private sphere. Social norms are rules of morality that people expect others to follow. These rules come from the private sphere of society and are often based on a common sense that most people share. Take the example of what movies and TV shows a child should be allowed to watch. It is not stated in public law what movies children of certain ages are allowed to watch. Still, these are situations that a moral agent, such as one implemented into a streaming service, might have to consider. How do we determine moral rules covering these aspects of life? A bottom-up approach determines what is the right thing to do based on aggregating common opinions among people [Conitzer et al., 2024]. That means, according to social choice, the most ethically correct course of action in a situation is the action deemed most ethically correct by "most people" [Baum, 2020]. We believe that moral norms can be used to represent the opinions of most people in AMAs.

By creating a moral knowledge base (KB) of a non-contradicting collection of norms and rules, one can create autonomous moral machines (AMAs) that are better able to justify their moral decisions by referring to the rules on which they are based [Wallach and Allen, 2009]. When creating moral KBs it is important to ensure that the set of rules does not contain contradictions, as decisions based on explicitly inconsistent pairs of rules will be absurd. Imagine creating a KB that contains the two rules "selling energy drinks to teenagers is good" and "selling energy drinks to teenagers is bad". An AMA built on this KB would be unable to deduce a moral judgment about the situation.

Conitzer [2023] points to one major problem when implementing social choice as a moral foundation in AMAs: "In general, we want the type of input or feedback that we ask of humans to be (1) natural to give, (2) informative about their preferences and values, and (3) of a type that can be used to align AI systems. For example, with current methods, having humans comment on an AI output in an open-ended text box may satisfy

1 and 2, but not 3". People express themselves in terms of natural language (NL), such as English, Norwegian or Japanese, while machines are instructed in terms of programming languages, such as Python, Java or C++. This creates a gap between the language most people use to express feedback, and the appropriate language used to instruct machines. Ideally, one could instruct machines using natural language. The challenges related to this is called the "NL to machine encoding problem".

One solution to the NL-to-machine-encoding problem is to use experts who are knowledgeable both in NL and machine-readable language to bridge the gap between these two sides. Knowledge engineering is the process by which experts consider how to structure and represent elements to be used as the knowledge base in a system. Using knowledge engineers has the advantage of creating transparency in terms of reasoning, and the process is relatively easy to control, although it is not entirely without issues [Shore, 1996]. When systems grow larger, the disadvantages often outweigh the advantages. Expert knowledge engineering often requires time and resources that might be unavailable, and the process of adding elements to the KB or editing existing elements in the KB often proves too difficult to maintain [Brachman and Levesque, 2004].

Some claim that complex large language models (LLMs) such as OpenAI's GPT-4 and Google's Gemini have revolutionized modern AI and opened up a wide spectrum of possibilities when it comes to working with data and language processing [Min et al., 2023, Zhao et al., 2023, Bubeck et al., 2023, Wu and Hu, 2023, Wei et al., 2023]. LLMs are able to construct knowledge about numerous different fields, and can create relations between elements in text. The question is if we can use this tool to simulate a knowledge engineer that is able to scale to larger systems. LLMs come with their own disadvantages, the most prominent being their tendency to provide incorrect information in a convincing manner, their limited domain-specific knowledge, and their poor ability to reason [Hadi et al., 2023]. If we want to exploit this technology in knowledge engineering moral KBs in AMAs, we need to identify the LLM's capacity for the tasks and understand the model's shortcomings. How closely can an LLM simulate a knowledge engineer in the field of machine ethics?

In this thesis, we attempt to use GPT-4 to automatically encode moral norms, expressed in human natural language, as machine-readable instructions that can be used as a moral KB in an AMA. We choose to represent the NL norms as first-order Horn clauses, which are first-order logic (FOL) expressions in a specific format that can be used as machine instructions. First-order Horn clauses were chosen over any other logic structure because they can be handled by most logic programming languages available, such as Prolog[1] and ASP [Baral, 2003], and other automated theorem provers such as Isabelle[2]. We use the logical representations to analyze two aspects of logic encoding:

[1]see e.g. https://www.swi-prolog.org
[2]see e.g. https://isabelle.in.tum.de/Isar/

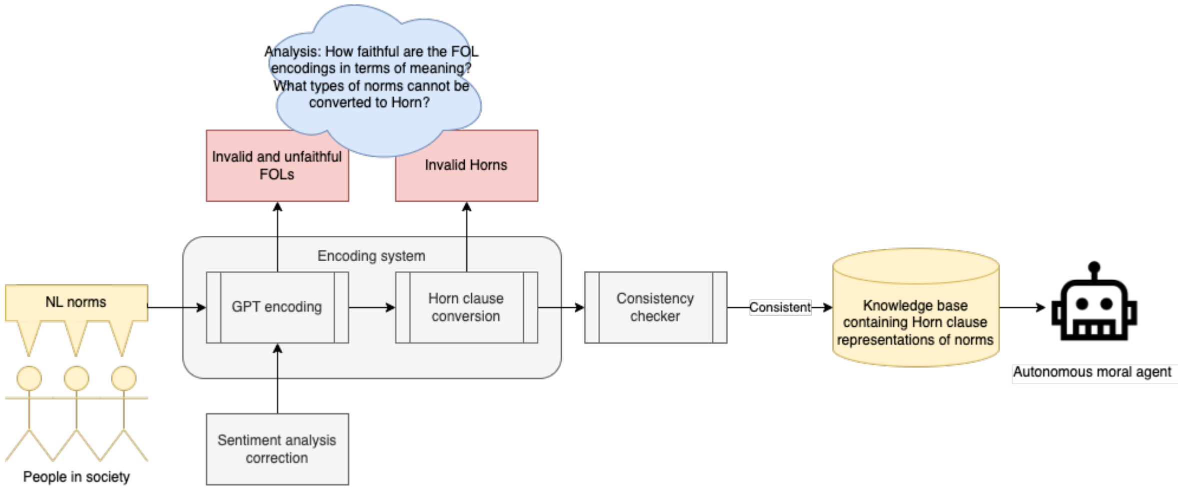


Figure 1.1: Thesis Outline Illustration

(1) what types of NL norms cannot be represented in first-order Horn, and (2) what mistakes does GPT-4 tend to make when asked to perform such a task? To ensure that the resulting set of machine-readable norms can be used as a KB for an AMA, we include an inconsistency checker that tells us whether or not the set of Horn clauses contains explicitly inconsistent clauses. We also show one approach to improving the system based on the mistakes identified by implementing a sentiment analysis model to correct these mistakes. A general outline of the solution approach presented in this thesis is illustrated in Figure 1.1.

In our implementation, we utilize techniques from the field of prompt engineering to construct a prompt that instructs the GPT-4 in encoding NL to FOL. Using techniques from natural language parsing, we are able to automatically verify whether or not the GPT-4 output follows the correct FOL syntax. We manually compare the output of the encoding system with the original norm sentence to determine the faithfulness of the automatic encodings in terms of intended meaning. We fine-tune the language model DistilBERT [Sanh et al., 2020] to create a sentiment analysis model used to improve the GPT-4 encodings[3]. In the end, we implement the resulting Horn representations of the norms to a logic program to find inconsistencies within the set of Horn clauses. We use an answer set programming solver, which is a simple and powerful declarative modeling language [Gebser et al., 2013].

We draw on knowledge from the fields of natural language processing (NLP), artificial intelligence, logic, sentiment analysis, and answer set programming (ASP). We use artificial intelligence to create logical representations of natural language sentences, and NLP in combination with FOL to verify the structure of these representations. We use the NLP tool sentiment analysis to show the possibility of enhancing one aspect of the GPT-encodings. Techniques within the field of logic are further applied to convert FOL

[3]DistilBERT is a distilled version of the popular LLM BERT [Devlin et al., 2019].

formulas to equivalent logic representations, such as Horn clauses. We use declarative logic programming to implement the inconsistency checker through the ASP language Clingo.

1.2 Research Questions and Success Criteria

The research question pursued in this thesis is:

- RQ: How can human language norms be converted into machine readable data represented by Horn Clauses, and which norms cannot be expressed in Horn Clauses?

To evaluate the success of the work in this thesis, specific criteria have been established in alignment with the overarching goals of the research. First, the thesis must demonstrate a comprehensive understanding of the theoretical framework on which it is based. Second, the research must introduce an encoding system that shows GPT-4's ability to encode natural language norms to FOL formulas. Third, the thesis should identify and describe the norm patterns that GPT-4 is unable to convert to FOL. Lastly, the thesis should contribute to the field of logic by describing the NL norm patterns that cannot be transformed to first-order Horn clauses.

When using deep learning models, one typically focuses on the accuracy of the model as a success criterion, perhaps specifically on counting false positive and false negative representations. In this thesis, we do not focus on such quantitative evaluations, but rather on the qualitative evaluations of the encoding system. The thesis aims to be a proof of concept, rather than a pilot implementation of a large volume of encodings to be used in a specific AMA. Thus, it is not necessary to encode a large number of examples and quantify the accuracy, but rather encode a smaller number and qualitatively analyze the mistakes in order to understand them.

1.3 Contribution

The thesis contributes to the fields of AI, machine ethics, and knowledge representation and reasoning by exploring the possibilities of using modern LLMs to solve the NL-to-machin- encoding problem.

The main contributions of the theses are the following:

(1) a proof of concept to demonstrate the ability to convert natural language norms to machine-readable representations of norms,

(2) a detailed analysis on the limitations of GPT-4 capabilities to encode NL norms to FOL,

(3) identification of what types of norm structures cannot be encoded to Horn clauses,

(4) a dataset of NL norms with their respective manually encoded baseline FOLs, and

(5) a dataset of NL norms and their respective GPT-4-encoded FOL and Horn representations along with an evaluation of the faithfulness of the GPT-4 encodings in terms of meaning.

1.4 Organisation of Thesis

This thesis is structured in nine main chapters, each designed to provide a comprehensive overview of the relevant theories, implementation methods, and contributions. Chapters 2 and 3 provide a conceptual background and a review of the relevant literature, specifically the topics of NLP, FOL, logic programming, and moral conflict. Chapter 4 outlines the methods and tools used in the implementation of the encoding system, the sentiment analysis model, and the inconsistency checker. Chapter 5 gives implementation details of the encoding system, explains how we performed the evaluation of the faithfulness of the encodings, and details the creation of the sentiment analysis model and the inconsistency checker. In addition, we describe how we performed the final round of automatic norm encodings to be used in further analysis and discussion. Chapter 6 presents the results of the experiments from the previous chapter. Chapter 7 provides an in-depth analysis on both the encoding mistakes made by GPT-4 and the norms that were unable to be represented in Horn clauses. Chapter 8 discusses some of the results of the analysis of encoding mistakes and reflects on how the results of this analysis are related to the current development of general use of LLMs. Chapter 9 concludes the thesis with a summary of the findings, limitations of the work, and suggestions for further work.

Appendices are included to provide additional context and reference material. The code and results produced for this thesis can be found on GitHub[4].

[4] `https://github.com/emmabjor/nl-horn-master/tree/main`

Chapter 2

Background

In this chapter, we introduce and define the key concepts and terminologies that are essential to understand the themes and methods presented in the subsequent chapters of this thesis. The background provided in this chapter serves as a foundation, offering clear and precise definitions and the necessary context to understand the research carried out. By establishing a common language and framework, we ensure that the ideas discussed in later chapters are approachable and comprehensible. In this chapter, we delve into the field of natural language processing (NLP), specifically natural language parsing, large language models (LLMs), and linguistic meaning. Additionally, we specify the syntax of first-order logic (FOL) and first-order Horn clauses. The chapter then provides a brief description of logic programming and answer set programming. Lastly, the dataset that provides the norms for this thesis, the Commonsense Norm Bank, is described.

2.1 Natural Language Processing

The way humans communicate through written and spoken words is crucial for efficient information exchange [Kapetanios et al., 2013]. Natural language processing (NLP) is a collection of computational techniques for automatic analysis and representation of human languages, where the goal is to convert natural language output into usable results [Maulud et al., 2021, Chowdhary, 2020].

The birth of NLP has often been attributed to the introduction of the Turing test proposed by Alan Turing in his paper "Computing Machinery and Intelligence" in 1950 [Turing, 1950]. This test is supposed to determine how well a machine is able to imitate human language and is often linked to the idea of determining intelligence in machines [Fanni et al., 2023]. Since then, NLP has been applied to tasks such as text classification, translation, summarization, knowledge retrieval, and text generation [Chowdhary, 2020]. In recent years, there has been major advancement in the field of NLP. Google introduced the Word2Vec algorithm in 2013, which uses neural networks to learn word associations from free text alone [1]. In 2018 this technology was further developed with the introduction of BERT (birdirectional encoder representations from transformers), a language model that is able to learn not only from the individual words themselves, but also from the context in which they are written [Fanni et al., 2023].

Traditional NLP sentence analysis is often divided into semantic analyses and syntax analyses. Semantic analyses serve to determine what a sentence means and are often performed by translating natural language into a language that uses simpler semantics, such as formal logic. Syntactic analyses aim to determine the structure of a sentence through a process called parsing [Chowdhary, 2020].

Before the introduction of large language models (LLMs), natural language parsing was the main technique used to identify structures in text [Chowdhary, 2020]. We describe the process of natural language parsing followed by the functionality of LLMs in the following sections.

2.1.1 Natural Language Parsing

Natural language parsing is the process of computing the structural description of a sentence through "[...] mathematical characterization of derivations in the grammar using a specified algorithm" [Chowdhary, 2020]. The goal of natural language parsing is to identify the different components of a sentence and how they relate to each other by assigning a tree structure to the sentence. Tree parsing can be especially useful when trying to determine whether or not a sentence follows a specific grammatical structure, which plays an especially important role in language translation [Chowdhary, 2020]. The syntactic parsing process consists of searching through all possible parse trees until the correct tree is found. The parse tree output displays dominance and precedence relations between parts of a sentence, often including attribute-value annotations that capture aspects of the linguistic description [Sharma et al., 2013]. This is possible as natural language is structured into classified phrases, which can be broken down into smaller and smaller constituent phrases, until we end up with the ultimate constituents. Noun phrases (NP) and verb phrases (VP) are examples of such classification phrases in the English language.

Grammar defines rules for how phrases are allowed to be structured. In order to create a parse tree of a sentence that describes its phrase structure, it is necessary to define a grammar that specifies rules for how words and phrases can be combined. Context-free grammar (CFG) is used to specify the syntax of a language and is widely used for languages where hierarchy and nesting of structures are important. A CFG consists of a set of production rules that describe how symbols of the language can be combined to form valid strings [Chowdhary, 2020]. Example 2.1.1 shows a simple grammar definition

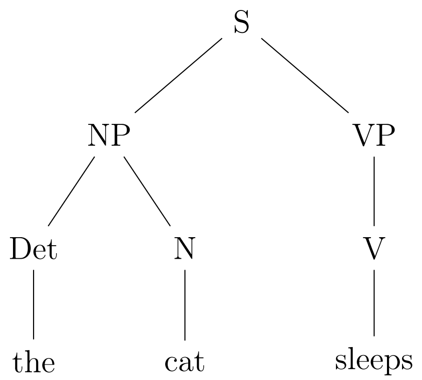
[1] Word2vec project information: `https://code.google.com/archive/p/word2vec/`

using CFG and how we can further use this structure to create a parse tree of a sentence.

We define a simple CFG:

$$
\begin{aligned}
S &\rightarrow \text{NP VP} \\
NP &\rightarrow \text{Det N} \\
VP &\rightarrow \text{V} \\
Det &\rightarrow \text{the} \\
N &\rightarrow \text{cat} \mid \text{dog} \\
V &\rightarrow \text{sleeps}
\end{aligned}
$$

In this CFG notation, a sentence (S) is made up of a noun phrase (NP) followed by a verb phrase (VB). A noun phrase (NP) is made up of a determiner (Det) followed by a noun (N). A verb phrase (VB) is just a verb (V). The determiner (Det) is the word "the". The noun (N) is the word "cat" or the word "dog". Lastly, the verb (V) is the word "sleeps". Using this grammar, we can generate the sentences "the cat sleeps" or the sentence "the dog sleeps". Parsing the sentence "the cat sleeps" would result in the following parse tree:



Example 2.1.1

Although tree parsing is a useful tool to employ to verify the structure of a machine-translated sentence, it is no longer the state-of-the-art technique for translating sentences. Parse tree translation finds the statistically best translation for each word based on its role in the sentence [Koehn, 2009]. Deep learning models, such as LLMs, on the other hand, represent words using vectors capable of incorporating the context of the word and find the best translation based on this [Topal et al., 2021]. We explain the use and architecture of LLMs in the next section.

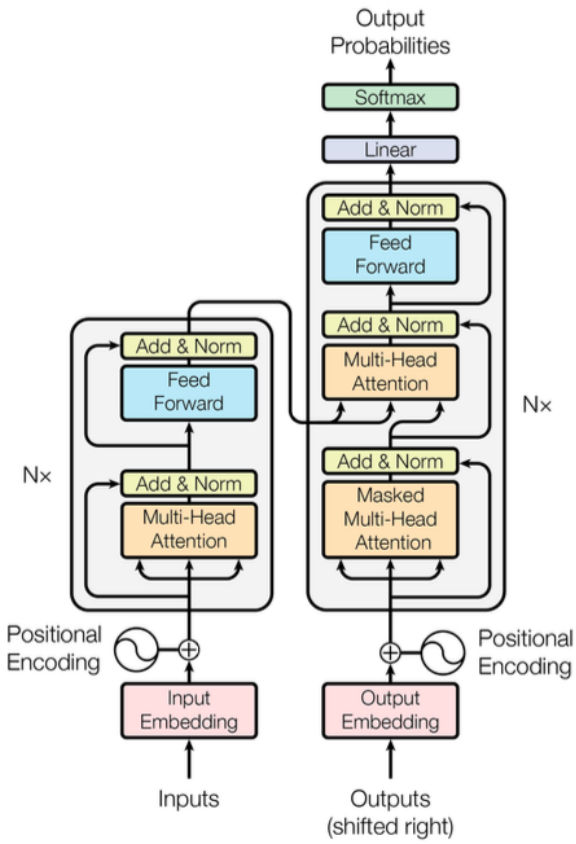


Figure 2.1: Transformer model architecture [Vaswani et al., 2017]

2.1.2 Large Language Models

Large language models are language models pre-trained on large amounts of text. The models learn how words and phrases commonly operate in relation to each other and use this to predict the most likely words to use in response to a request [OpenAI, 2024c].

OpenAI launched one of their newest LLM in March 2023, GPT-4, which is claimed to outperform previous models and other state-of-the-art systems when it comes to traditional NLP tasks [OpenAI et al., 2023].

Transformer Model

Most of the pre-trained LLMs that are widely used today utilize the transformer architecture introduced by Vaswani et al. [2017]. Unlike other popular deep learning models, such as sequence-based recurrent neural networks (RNNs), which process text sequentially, transformers utilize a mechanism called "self-attention". A model processing text sequentially is limited in its parallelization and long-range dependency capabilities, while self-attention enables the transformer model to process text data in parallel, allowing it to handle long-range dependencies throughout the text [Topal et al., 2021].

The transformer model is mainly composed of an encoder and a decoder, as illustrated in Figure 2.1. These parts can be used independently or in combination [Min et al., 2023].

An encoder-only model, such as BERT[2] [Devlin et al., 2019], strictly uses the encoder part of the transformer model, illustrated by the left component in Figure 2.1 [Min et al., 2023]. The encoder component includes a self-attention layer and a fully connected feed-forward network. It takes text as input and creates embeddings by encoding each word in the text input to a sequence of numbers, called a feature vector. The feature vector represents the meaning of the word. The self-attention layer of the encoder makes sure that this representation encompasses the context in which the word is used by also considering the words on either side of it. This means that the word "bow" will get different representations of the feature vectors in the two sentences from Example 2.1.2 as they have different meanings depending on the context in which they are used in [Vaswani et al., 2017, Hugging Face, 2023].

"Take a bow at the end of the play."
"Take a bow and an arrow to shoot."

Example 2.1.2

A decoder-only (or auto-regressive) model, such as the GPT models, strictly uses the decoder part of the transformer model, illustrated by the right component in Figure 2.1 [Min et al., 2023]. The decoder consists of the same self-attention and feed forward network layers as the encoder but has an additional masked self-attention layer. Similar to the encoder, the decoder creates a feature vector representation of the input text. However, in contrast to the encoder, this feature vector will not take the words on both sides of the target word into consideration, but rather only the words on one of the sides. Usually, only words on the left side will be considered and those on the right side will be masked. This means that only the context found on the left side of the target word will affect the feature vector representation of the target word. This is why the decoder component is said to be uni-directional as it only considers the context in one direction, in contrast to the bi-directional encoder. This quality can be advantageous in casual language modeling, which is the process of generating text sequences by continuously "guessing" (using statistical probability) the following word in a sequence. The model outputs the next word in the sequence and then reuses the sequence of outputs to generate the next following word, and so on in what is called an auto-regressive manner. This continues until the decoder outputs what is considered to be a stopping value, like a dot, that represents the end of the sequence. The context size, that is, how many output

[2]The BERT model and a range of other BERT-based models can be found and used through the HuggingFace community `https://huggingface.co/docs/transformers/en/model_doc/bert`

words the model includes in its input, differs from model to model. GPT-2, for example, has a maximum context size of 1024 words [Vaswani et al., 2017, Hugging Face, 2023].

An encoder-decoder (or sequence-to-sequence) model, such as BART, uses the encoder and decoder components in combination [Min et al., 2023]. Figure 2.1 shows how the encoder and decoder are connected in an encoder-decoder model. In this architecture, the numerical feature vector from the encoder, in addition to the initial start-sequence, is used as input to the decoder. The decoder creates the output in an auto-regressive manner where each word prediction is based on both the feature vector from the encoder and the previous output-sequence generated by the decoder [Vaswani et al., 2017, Hugging Face, 2023].

Prompt Engineering

Prompt engineering is defined as the process of designing effective input prompts that elicit desirable responses from language models [Ekin, 2023]. The prompt is the set of instructions that is provided to the LM and describes the task we want the LM to perform. Differences in the engineering quality of the prompt can lead to significant differences in the quality and relevance of the model output [White et al., 2023].

By refining the prompts based on established prompt engineering techniques and iteratively modifying the prompts based on the model's generated output, one can nurture more accurate and precise LM responses [Ekin, 2023, Velásquez-Henao et al., 2023].

2.1.3 Linguistic Meaning

Meaning is defined as a relation between symbols of a language and certain entities independent of that language [Gamut, 1991]. More specifically, it is the meaning of all types of constituent and expression in the language, as well as the relationship between them [Allan, 2001, p. 5]. Within the field of language and linguistics, semantics is the study of meaning in human languages. Humans construct meanings not just on a word-level, but dependent on the context of the phrases, sentences, and longer text in which the word is used. This is because the meaning of individual words can change within the context in which they are used, as illustrated in the two "bow"-sentences from Example 2.1.2 [Allan, 2001, p. 6].

The ability to distinguish words based on context is not unique for humans. As explained in Section 2.1.2, language models using the Transformer architecture incorporate the context surrounding a word in its representation. However, this is not the same as constructing true meaning from the language. As LMs are ultimately trained to learn form and patterns in text data, they cannot possess any meaning in terms of understanding the relation between natural language expressions and the communicative intent (the real-life object) they are evoking [Bender and Koller, 2020, Bender et al., 2021]. Bender and Koller [2020] illustrate this claim by introducing a thought experiment called "the Octopus test". The Octopus test presents an imaginary scenario in which an intelligent deep-sea octopus secretly taps into a conversation between two stranded individuals who communicate via an underwater cable. The octopus is completely unfamiliar with any kind of natural language and lacks experiential context to the concepts the two individuals are discussing, but it is able to learn patterns in the conversation and predict responses based on this. After a while of listening and learning, the octopus can convincingly imitate either individual in similar conversations, for example explaining in fine detail the difference between a papaya and a guava. However, if the octopus was physically presented with the two fruits, it would not be able to say which fruit is which. This is because, although the octopus can recreate natural language utterances, it does not know what those utterances mean or what they refer to [Bender and Koller, 2020]. These arguments can also be said to hold for a language model, where the LM acts as the octopus in a dialog. The LM knows how to create responses, but not what those responses mean.

People have a predisposition not only to construct meaning for themselves but also to project that construction onto others. That is, we implicitly assume that other people who are communicating with us share the same grounding of communicative intent to real-life objects as we do [Bender et al., 2021]. The illusion that LMs seem to have real understanding and meaning, acting like what Bender et al. [2021] call a *stochastic parrot*, is most likely due to this human tendency. Although LMs and computer programs do not have meaning in the sense that humans do, we are still able to represent meaningful entities in machine-readable language through logic.

2.2 First-Order Logic

2.2.1 Logic and Encoding Meaning

Logic is a form of language that can express knowledge in a computer-tractable form [Russell and Norvig, 2010]. We use logic as a tool for reasoning [Makinson, 2012, p. 189]. Traditional computer programming consists of creating programs using large structures of logic. By providing a computer with relatively simple but precise logical commands, we can achieve formidable results, such as solving complex mathematical equations or creating entire virtual worlds for gaming purposes [Bjørkeng, 2018, p. 23].

The reason why logic constitutes such a powerful tool is that it provides a way of expressing meaning in a more strict and defined way than natural language does. Although logic is unable to make machines understand concepts in the way humans do, it can provide a way to represent these concepts in a way that machines can utilize.

2.2.2 A Precis of First-Order Logic

First-order logic (FOL) is a collection of formal logic systems that generally consist of objects with certain properties and relationships between these objects [Russell and Norvig, 2010]. The syntax of FOL is built up of the following elements:

- *Terms*: A logical expression that refer to an object. A term can be either a constant symbol or a variable symbol.
  - Constant symbols (*A, B, OLA, NORWAY*): Symbols that refers to exactly one object.
  - Variable symbols (x, y, v_1, v_2): Symbols that can be substituted with multiple objects.
- *Predicates* (*P, R, Person, FromCountry*): Symbols that refer to a particular relation between objects.
- *Logical Connectives* or *Junctions* ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$): Logical symbols that combine atomic sentences to create complex sentences. $\neg$ represents negation, $\wedge$ represents conjunction, $\vee$ represents disjunction, $\rightarrow$ represents implication, and $\leftrightarrow$ represents biconditional.
- *Quantifiers* ($\forall, \exists$): Logical symbols that express properties of a collection of objects. $\forall$ represents "all" and $\exists$ represents "at least one".

An *atomic formula*, $R(t_1, ..., t_n)$, is represented by a predicate R followed by a left parenthesis, followed by n constant or variable symbols t_i, where n is the number of places of the predicate, with commas separating the terms, followed by a right parenthesis. Each predicate symbol has a fixed number of places to be filled by constants or variables.

We use logical connectives to combine atomic formulas into complex sentences such as $P(Q) \wedge FromCountry(OLA, NORWAY)$, which means that the two atomic formulas *P(Q)* and *FromCountry(OLA, NORWAY)* are both true simultaneously. The negation symbol, $\neg$, is used with single atomic formulas to express negation of said formula, for example $\neg FromCountry(OLA, SWEDEN)$

The sentence "All Norwegians are Europeans" can be expressed in FOL using the universal quantifier as $\forall x(Norwegian(x) \rightarrow European(x))$. The FOL formula can be read as "for all x's, if the x is Norwegian then the x is European".

Example 2.2.1

Quantifiers are used in combination with variables. Universal quantification ($\forall$) is used to express properties over all objects of a certain kind. An example of the use of universal quantifier can be seen in Example 2.2.1. In contrast, the existential quantifier ($\exists$) expresses properties over a specific set of objects without having to name the objects [Makinson, 2012, p. 189-239][Boolos et al., 2007, p. 106-112]. An example of the use of existential quantifier can be seen in Example 2.2.2

The sentence "Someone is from Norway and Sweden" can be expressed using the existential quantifier as $\exists x(FromCountry(x, NORWAY) \land FromCountry(x, SWEDEN))$. This FOL formula can be read as "there exists at least one x, where x is from the country Norway and x is from the country Sweden".

Example 2.2.2

Boolos et al. [2007] define a well formed formula (WFF) in FOL in the following: "[...]anything that is a (well formed first-order) formula can be built up from atomic formulas in a sequence of finitely many steps - called a *formation sequence* - by applying negation, junctions, and quantifications to simpler formulas" [Boolos et al., 2007, p. 107]. That means, if and only if F is a WFF, so is its negation, $\neg F$. And, if and only if F and G are WFF, then so are their conjunction, $(F \land G)$, disjunction, $(F \lor G)$, conditional, $(F \rightarrow G)$ or biconditional, $(F \leftrightarrow G)$. Example 2.2.3 shows a well-formed FOL, referred to as a syntactically valid FOL. Example 2.2.4 shows an FOL that is not well formed, referred to as an syntactically invalid FOL.

FOL: $\forall x((FromCountry(x, NORWAY) \lor FromCountry(x, SWEDEN) \lor FromCountry(x, DENMARK) \rightarrow Scandinavian(x))$

We can interpret this WFF to mean "for all x, if x is from Norway or x is from Sweden or x is from Denmark, then x is Scandinavian".

Example 2.2.3

2.2.3 Clause Normal Form

An FOL formula written in clause normal form is equivalent to the original formula in terms of satisfiability, but formulated with a special syntactic structure [Makinson, 2012, p. 204]. If two formulas are satisfiably equivalent, then one formula is satisfiable if and only if the other is [Boolos et al., 2007, p. 243]. Formulas written in clause normal form

FOL: $\forall x(FromCountry(Norwegian(x), NORWAY) \quad \vee$ $FromCountry(x, SWEDEN), FromCountry(x, DENMARK)$

The FOL is not WFF as the atomic formula *FromCountry(Norwegian(x), NORWAY)* is not well formed and the two atomic formulas *FromCountry(x,SWEDEN)* and *FromCountry(x,DENMARK)* are combined using an invalid connective.

Example 2.2.4

have the advantage of providing a consistent and predictable structure which is necessary when used in automatic processing. There are several types of normal forms. The ones considered in this thesis are described below.

Disjunctive Normal Form

A single disjunction is defined as any disjunction of atomic formulas and is written on the form $p_1 \wedge ... \wedge p_n$, where p_i is a valid first-order atomic formula. A formula is in disjuntive normal form (DNF) if and only if it consists of a disjunction ($\vee$) of conjunctions ($\wedge$), or in other words, a distribution of OR's over AND's, on the form seen in Equation 2.1, where p_i and q_i are valid first-order atomic formulas [Makinson, 2012, p. 202-203]. Note that a single conjunction is also considered a valid DNF structure.

$$(p_1 \wedge ... \wedge p_n) \vee ... \vee (q_1 \wedge ... \wedge q_m) \tag{2.1}$$

Conjunctive Normal Form

A single conjunction is defined as any conjunction of atomic formulas and is written on the form form $p_1 \vee ... \vee p_n$, where p_i is a valid first-order atomic formula. A formula is in conjunctive normal form (CNF) if and only if it consists of a conjunction ($\wedge$) of one or more disjunctions ($\vee$), or in other words, a distribution of AND's over OR's, on the form seen in Equation 2.2, where p_i and q_i are valid first-order atomic formulas [Makinson, 2012, p. 204-205]. Note that a single disjunction is also considered a valid CNF structure.

$$(p_1 \vee ... \vee p_n) \wedge ... \wedge (q_1 \vee ... \vee q_m) \tag{2.2}$$

Skolem Normal Form

Skolem normal form is a specific type of clause normal form where all existential quantifiers are removed from the formula [Boolos et al., 2007]. Removing existential quantifiers from an FOL formula is done through the process of Skolemization. Each variable that is quantified by an existential quantifier is replaced with a Skolem constant or a Skolem function. An existentially quantified variable that exists independently of any other quantifier can simply be replaced with a unique constant, which can be seen as replacing the variable with a unique name for the individuals represented by the existentially quantified variable. For example, the Skolemized version of the formula $\exists x(Fx)$ is simply $F(A)$, where A is a unique constant not used elsewhere. When the existentially quantified variable exists within the scope of an universally quantified variable, we need to represent the variable using a Skolem function [Boolos et al., 2007, p. 237-243]. A Skolem function is a unique function, f_i, for each existentially quantified variable that contains the the variables for each universally quantified variable the existentially quantified variable exists within the scope of. We show how an FOL formula is converted to the Skolem form in Example 2.2.5.

The FOL formula:

$$\forall x \forall y \exists z(F(x) \wedge (G(y) \vee H(z))$$

is converted into Skolem normal form through the process of Skolemization to the formula:

$$F(x) \wedge (G(y) \vee H(f(x, y)))$$

where the existentially quantified variable, z, is replaced by a Skolem function, f, that exist in the context of the higher-order universally quantified variables, x and y.

Example 2.2.5

2.2.4 Horn Clauses

Horn clauses are a particular type of first-order normal form clauses. A Horn clause a is a formula on the form seen in Formula 2.3 or Formula 2.4, where p_i and q are valid first-order atomic formulas with universally quantified variables.

$$\neg p_1 \vee ... \vee \neg p_n \vee q \tag{2.3}$$

$$\neg p_1 \vee ... \vee \neg p_n \tag{2.4}$$

Horn sentences are conjunctions of Horn clauses. A positive, or definite, Horn clause has exactly one positive atomic formula, as in Formula 2.3. A negative Horn clause has zero positive atomic formulas, as in Formula 2.4. A definite Horn clause like Formula 2.3 can also be written as: $p_1 \wedge ... \wedge p_n \rightarrow q$, where the parte before the implication symbol $\rightarrow$ is called the antecedent and the part after the implication symbol $\rightarrow$ is called the consequent [Brachman and Levesque, 2004, p. 85-86].

All other formulations are considered invalid Horn. That is, clauses that, when written as a disjunction of atomic formulas, contain two or more positive atomic formulas. Formula 2.5 shows the format of an invalid Horn clause, as at least two atomic formulas (q_1 and q_2) are positive. We can extend the number of conjuncted atomic formulas in Formula 2.5 indefinitely with positive or negative atomic formulas, but as long as there are at least two positive atomic formulas, it will always be considered invalid Horn.

$$\neg p_1 \vee ... \vee \neg p_n \vee q_1 \vee q_2 \vee ... \vee q_m \tag{2.5}$$

We describe the Horn clause structure as Horn clauses are commonly used in logic programming.

2.3 Logic Programming

Logic programming is defined as the use of logic to perform computation, often built upon Horn clause logic [Kowalski, 2013]. Logic programming was first introduced by Green [1969] as a way to use clausal form logic to represent computer programs [Lloyd, 2012]. In 1972, Kowalski and Colmerauer implemented the first Prolog system using SL-resolution (linear resulution with selection function) using the full clausal form of FOL [Kowalski, 2013]. Since then, the Prolog language has extended to many different branches or dialects and remains one of the most popular logic programming languages to this day.

With the development of Datalog and answer set programming (ASP), logic programming has become more declarative, which means that the user describes what is counted as a solution to a problem without specifying an algorithm to solve it [Kowalski, 2013]. Logic programming played an important role in the development of expert systems, as expert systems are built on declarative representations of expert knowledge [Merritt, 2012]. Logic-based approaches to AI suggest that autonomous agents should use knowledge bases to create models of their environments and goals. These knowledge bases should be a collection of statements in a declarative language that includes common sense knowledge as well as expert knowledge that the agent needs to perform specific tasks [Erdem et al., 2016].

2.3.1 Answer Set Programming

Answer set programming is a form of logic programming that uses a declarative programming methodology. The answer set solvers perform a search to find stable models consisting of all facts that can be derived from the rules defined in the program [Lifschitz, 2019, p. 1-4]. This can be thought of as a set of beliefs that are rational to be held given the collection of rules and facts stated in the program [Erdem et al., 2016].

ASP-based languages and Prolog appear similar, but use different computational mechanisms. Prolog allows the user to describe how a solution should be reached, while ASP approaches only allows the declaration of relationships and constraints [Lifschitz, 2019]. Prolog computes a result to a query using SLD resolution (Selective Linear Definite clause resulution), depth-first search, and backtracking by representing the possible solution to the query as branches in a solution tree and exploring branch by branch until a solution is found or no further progress can be made. ASP approaches, on the other hand, computes stable models that contain a collection of truths that together satisfy the rules of the problem. If there are multiple such collections, ASP approaches will find all the stable models. Lastly, Prolog and ASP languages differ in the way they represent negation. ASP languages support both strong (or classical) negation as well as negation as failure, whereas Prolog only supports negation as failure. Negation as failure, `not p`, means `p` is assumed to be false if it cannot be proven true. Strong negation, `-p`, means that `p` is explicitly false [Lifschitz, 2008].

Clingo is one of the more widely used ASP languages and is a part of Potassco (Potsdam Answer Set Solving Collection), a collection of ASP-related tools created at the University of Potsdam [Lifschitz, 2019].

A Clingo program consists of rules, facts and constraints. These structures are formed from atoms. An atom consists of a predicate symbol and an optional list of arguments enclosed in parentheses. An atom, a rule, or another syntactic expression is ground if it does not contain variables. A typical rule in the Clingo programming language consists of a head and a body separated by the "if" symbol `:-` and with a period at the end, such as:

```
q(X) :- p(X).
```

which can be read as $P(x) \rightarrow Q(x)$.

Rules that do not contain `:-` (a rule head without its body) is called a fact, such as:

```
p(a).
```

A rule with no head is called a constraint and is used to define scenarios that should not be present in any stable model, such as:

```
:- p(X).
```

By declaring rules and constraints that show what is allowed (or not allowed) and facts that show what exists within the program, an ASP solver can find stable models that find all possible facts to be derived from the program. If no stable models are found, it means that the rules and facts are conflicting and that the program will terminate as unsatisfiable [Lifschitz, 2008]. A simple satisfiable Clingo program is shown in Example 2.3.1. An unsatisfiable Clingo program is shown in in Example 2.3.2.

```
% Define the parent relationships using facts
parent(john, mary).
parent(john, tom).
parent(mary, sue).
parent(mary, tim).
parent(tom, kate).

% Define the granparent relationships using a rule
grandparent(X, Y) :- parent(X,Z), parent(Z,Y).

% Output the grandparents
#show grandparent/2
```

This program will yield one stable model. By asking the program to `#show grandparent/2`, the output will show the grounded grandparent relations that are possible in this model. The output will be:

```
Stable Model 1:
grandparent(john, sue).
grandparent(john, tim).
grandparent(john, kate).
```

Example 2.3.1: Satisfiable Clingo script

```
% Define the parent relationships using facts
parent(john, mary).
parent(john, tom).
parent(mary, sue).
parent(mary, john).
parent(tom, kate).

% Define the parent constraint
:- parent(X,Y), parent(Y,X).
```

This program will terminate as `unsatisfiable` as the two facts `parent(john,mary)` and `parent(mary,john)` directly conflict the parent constraint, which says that no pair of people can be defined as the parents of each other.

Example 2.3.2: Unsatisfiable Clingo script

2.4 Commonsense Norm Bank

The Commonsense Norm Bank dataset, created by Jiang et al. [2021] for the Delphi experiment[3], contains 1.7 million examples of descriptive judgments on everyday situations. The judgments consist of different aspects of social norms and ethics drawn from five different data sources: SOCIAL CHEMISTRY [Forbes et al., 2021], ETHICS [Hendrycks et al., 2021], Moral Stories [Emelin et al., 2021], Social Bias Inference Corpus (SBIC) [Sap et al., 2020] and SCRUPLES [Lourie et al., 2021].

The different sources are combined and categorized into two different data types: depictions/questions of everyday situations (referred to as "free-form") and statements with moral implications (referred to as "yes/no") [Jiang et al., 2021]. Examples of free-form sentences can be seen in Example 2.4.1. As we want our system to be able to convert statements with moral implications, we are only interested in the yes/no types.

"expensive items usually looking better than cheap items"
"Am I allowed to punish someone?"
"letting dogs be neglected and alone"
"Sue plops down onto her friends couch"

Example 2.4.1: Commonsense Norm Bank free-form sentences [Jiang et al., 2021]

The yes/no category consists of social norms/rules of thumb that include an action and the judgment of the said action, such as "it is kind to express concern over your neighbor's friend", where "express concern over your neighbor's friend" is the action, which we call the norm part of the sentence, and "it is kind" is the judgment of the action, which we call the evaluation part of the sentence.

Table 2.4.2 shows ten examples of sentences from this category. These sentences are taken mainly from the Social Chemistry corpus created by Forbes et al. [2021]. The Social Chemistry corpus was constructed using 104,000 real-life situations gathered from four domains: titles of posts in the subreddits "r/confessions" and "r/amitheasshole", sentences from the ROCStories corpus and titles from the "Dear Abby" advice column archives [Forbes et al., 2021]. For each of these situations, crowd workers wrote 1 to 5 rules of thumb inspired by the situation, resulting in 292,000 rules [Forbes et al., 2021]. In addition to these 292,000 rules of thumb from the Social Chemistry corpus, Jiang et al. [2021] introduced noisy syntactic forms of the rules to increase robustness, resulting in a total of 478,000 statements of ethical judgments[4].

[3] A demo of the Delphi prototype can be found at `https://delphi.allenai.org`
[4] We accessed this data set by reaching out to the author of Jiang et al. [2021].

People should not get a divorce.
It's rude to try to force a person to do something they don't want to do.
It's not okay to want others to know what is right.
You should not always learn how to do your job right.
It's irrational to not want to be inconvenienced by new habits.
You shouldn't put up with a parent's rude and cruel behavior.
It is bad to go out with people who you like.
It's fine to spend time watching a movie.
You should accept someone's excuse for being racist.
It's good to spend time with your partner.

Example 2.4.2: Commonsense Norm Bank norms [Yang et al., 2023]

Chapter 3

Related work

In this chapter, we present the existing research on relevant topics that provides the basis for the work in this thesis. This chapter specifically explores how LLMs are employed in NLP tasks, with particular attention to logic encoding and prompt engineering. We include a summary of research within the field of moral conflict to highlight the problem of having conflicting norms within autonomous systems. The methodology used in the literature search can be found in Appendix A.

3.1 Natural Language Processing and Large Language Models

3.1.1 Natural Language Processing Tasks

The idea of pre-training a language model was first suggested by Collobert and Weston [2008] as a way for computers to automatically learn features rather than handcraft them. With the introduction of the neural network architecture called the Transformer model, it became possible to scale up neural model training, which was the predominant approach at the time, in a way not possible with earlier approaches [Vaswani et al., 2017, Min et al., 2023]. The current most popular large pre-trained language models, such as GPT, BERT, and BART are all based on the Transformer architecture [OpenAI et al., 2023, Devlin et al., 2019, Lewis et al., 2019].

Min et al. [2023] summarize recent developments when it comes to using LLMs in NLP tasks in their survey overview and finds that there are three trending paradigms in the use of pre-trained language models (PLMs) for NLP. These are fine-tuning PLMs, using prompt-based learning and text generation tasks [Min et al., 2023].

Bubeck et al. [2023] argue how the current development of LLMs, especially the GPT-4 model, exhibit remarkable capabilities in several domains, especially in that of language processing tasks.

Basu et al. [2021] present a semantic-driven approach to natural language understanding based on semantic knowledge mapping rather than machine learning approaches that learn patterns based on training data. They propose a method of using sentence parse trees in combination with VerbNet[1] to map the parse tree to the knowledge it represents [Basu et al., 2021]. They present two applications of this approach, SQuARE (Semantic-based Question Answering and Reasoning Engine) and StaCK (Stateful Conversational Agent using Commonsense Knowledge), that are able to represent knowledge in ASP by using semantic driven ASP code generation that maps a sentence's parse tree to the skeletal parse tree found in the VerbNet lexicon based on the verbs in the sentence. They found both applications to perform better than machine learning based systems in regard to accuracy and explainability [Basu et al., 2021].

Hu et al. [2024] present a survey of Knowledge Enhanced Pre-trained Language Models (KR-PLMs) in natural language understanding (NLU) and natural language generation (NLG). Within the fields of NLU and NLG there are several types of knowledge that can be explicitly incorporated into a PLM, and Hu et al. [2024] propose taxonomies for both fields and the relevant work within the taxonomies.

The goal of NLU is to enable machines to understand and interpret data expressed in NL. Hu et al. [2024] argue that within the field of NLU, knowledge can be incorporated to a PLM as text, linguistic knowledge, knowledge graph (KG) and rule knowledge. Text knowledge is retrieved from general-domain text collections. Examples of linguistic knowledge is Part-of-Speech (POS) tagging and sentiment tags of words. Knowledge graphs are expressions meant to represent real world knowledge in the structured form of a graph which can help improve semantic and relational understanding in PLMs. Rule knowledge is incorporated by formalizing rules from external sources to logic representations, and thus facilitate better reasoning in the PLM [Hu et al., 2024]. An example of a system incorporating rule knowledge is RuleBERT, which trains the LLM BERT on a set of soft Horn rules to create probability predictions for a given natural language (NL) hypothesis [Saeed et al., 2021].

The goal of NLG is to enable machines to generate data expressed in NL in a way similar to how humans do it. Within the field of NLG, Hu et al. [2024] argue that the performance of a PLM can be enhanced by incorporating retrieval-based or KG-based knowledge. In retrieval-based methods, additional relevant knowledge is retrieved from external sources and used to guide the generation process. KG-based methods use knowledge graphs representations to input the relevant knowledge used in the generation process [Hu et al., 2024].

The fields of NLU and NLG are important when it comes to logic encoding, the task of converting NL statements to logic representations.

[1] A lexicon providing skeletal parse trees, thematic roles and semantic representations of each verb class.

3.1.2 Logic Encoding

Prior to the rise of LLM, encoding tasks were developed to convert natural language to logic using approaches such as reinforcement learning [Lu et al., 2022] and neural networks [Singh et al., 2020, Petrucci et al., 2018].

Han et al. [2022] demonstrate different LM's abilities to perform natural language reasoning with FOL by fine-tuning and prompting the medium and large language models GPT-3.5, BERT, RoBERTa, GPT-NeoX, OPT, and Codex on a gold-standard data set containing pairs of natural language sentences and their manually encoded FOL representation. They specifically tested GPT-3 and Codex's ability to translate from NL to FOL and found the models to perform only slightly better than random [Han et al., 2022].

Yang et al. [2023] display a system called LogicLlama that enhances the performance of the GPT-3.5 model by first prompting GPT-3.5 to encode natural language sentences to FOL, and then using a fine-tuned language model calledLlama-7B model to correct these results. Yang et al. [2023] compared the results to those produced by prompting GPT-4 alone and found that using GPT-4 is as good in encoding from NL to FOL as the LogicLlama system.

Rajasekharan et al. [2023] present a system called STAR (Semantic parsing Transformer and ASP Reasoner) which uses LLMs to generate predicates from a sentence representing the semantic properties of the sentence, and use ASP to reason with these predicates. In addition, the ASP reasoning system is equipped with commonsense knowledge related to the predicates. The STAR system is evaluated by comparing its accuracy to that of the accuracy of GPT-3 on three different reasoning tasks. These tasks are, solving qualitative reasoning problems, solving math word problems and acting as a hotel concierge holding a conversations with a user looking for restaurant recommendations. Their experiments show that GPT-3 is able to extract knowledge predicates from sentences with high accuracy using few examples [Rajasekharan et al., 2023]. The STAR system outperforms the GPT-3 model in the qualitative reasoning task and conversational task, and performs equally well as GPT-3 on the math word task [Rajasekharan et al., 2023].

3.1.3 Prompt Engineering

Min et al. [2023] argue that the use of prompt-based learning is one of the state-of-the-art approaches to using LLMs for NLP tasks. Research in the field of prompt engineering has focused on formulating a set of concrete prompt engineering techniques or methods that can be used quantitatively and yield consistent results [White et al., 2023, Ekin, 2023, Velásquez-Henao et al., 2023, Arora et al., 2022, Wei et al., 2023, Zhou et al., 2023].

Liu et al. [2021] give a systematic review of prompting methods in NLP by organizing the existing work within the field along several dimensions. They argue that the NLP paradigm has shifted from mainly using a “pre-train, fine-tune” approach to a “pre-train, prompt, and predict” approach. Instead of adapting pre-trained LMs via objective engineering, prompts are formulated to manipulate the model to predict the desired output [Liu et al., 2021]. Liu et al. [2021] identify basic design choices to be considered in prompting methods, these being pre-trained model choice, prompt engineering, expanding the paradigm and prompt-based training strategies. They give an extensive overview of the different works in which these prompting methods have been applied to, describing the tasks and PLMs used [Liu et al., 2021].

Zhu et al. [2023] suggest a benchmark designed to measure the resilience of LLMs to adversarial prompts, in addition to a comprehensive analysis of prompt robustness and transferability. The benchmark, PromptBench, considers how well the LLMs are able to respond to perturbations in prompts and thus identifies key attributes of robust prompts [Zhu et al., 2023].

3.2 Moral Conflict

Moral conflict, or moral dilemmas, is defined in the field of moral philosophy as situations in which an agent has moral reasons to perform two actions, but doing both actions causes a conflict [McConnell, 2022]. When AMAs are guided by norm inputs from multiple different stakeholders, dilemmas in the form of normative conflicts may arise.

Santos et al. [2017] give an overview of the current state-of-the-art solutions to detecting and resolving normative conflicts in multi-agent systems. These techniques are analyzed based on whether they address conflicts at the design stage or during runtime, and whether they handle direct or indirect conflicts. The techniques are compared in terms of strengths and weaknesses, and how they are applied in practical scenarios. The paper organizes the existing literature according to which technique they use and describe how they compare against each other.

Liao et al. [2023] propose a solution to the moral conflict problem by introducing an ethical recommendation component using techniques from normative systems and formal argumentation theory. This component solves the moral conflict between different stakeholders by considering the arguments in each stakeholder’s normative system, combining the normative systems, or deciding which of the stakeholder takes preference over the others. This component is meant to represent a Jiminy Cricket-like figure from the story of Pinoccio, acting as an ethical advisor to the AMA by producing moral recommendations through resolving normative dilemmas.

The works outlined in this chapter provide a useful baseline for the methods and experiments that will be described in the following chapters of this thesis.

Chapter 4

Methodology

In this chapter, we describe the methods and tools used to develop a system that can encode natural language sentences that express norms into first-order Horn logic representations. The chapter includes a detailed examination of the methods used in the implementation of the system and the analytical methods applied to assess its outcomes.

Section 4.1 of this chapter outlines the research design choices applied to two distinct parts of the research and also specifies the boundaries for the FOL encodings. Section 4.2 describes the tools and methods that were used collectively in both parts of the research.

4.1 Research Design

The work in this thesis can be divided into two distinct parts. The first is a developmental part, which consists of the implementation of an encoding system aimed at encoding natural language norms to Horn clause representations, as well as the implementation of a sentiment analysis model and an inconsistency checker. The second is an analytical part, which consists of the analysis of the results of the encoding system performed to draw conclusions about the effectiveness and limitations of the encoding system. The research designs for each of these parts are described individually.

The main goal of the developmental research part is to create a system able to convert natural language norms into first-order Horn clauses. The implementation of the encoding system involves breaking down the encoding task into several distinct steps, as illustrated in Figure 4.1.

The first step is to encode norms from the Commonsense Norm Bank into FOL representations. In the second step, the syntax of the FOL representations is verified. In the third step, verified FOLs are converted to conjunctive normal form (CNF). In the final step, CNFs are checked to determine if they can be categorized as valid Horn. Splitting the task into these particular steps gives us the opportunity to use existing tools found in the literature described in Chapter 3.

In hindsight, we found that by splitting up the encoding task, we gained more insight into where the system might need improvement. The system was evaluated iteratively and the implementation strategy improved. This iterative evaluation provided insight and ideas for further development of the system. The implementation of a sentiment analysis model and an inconsistency checker was motivated by the insight found through continuously evaluating the encoding system.

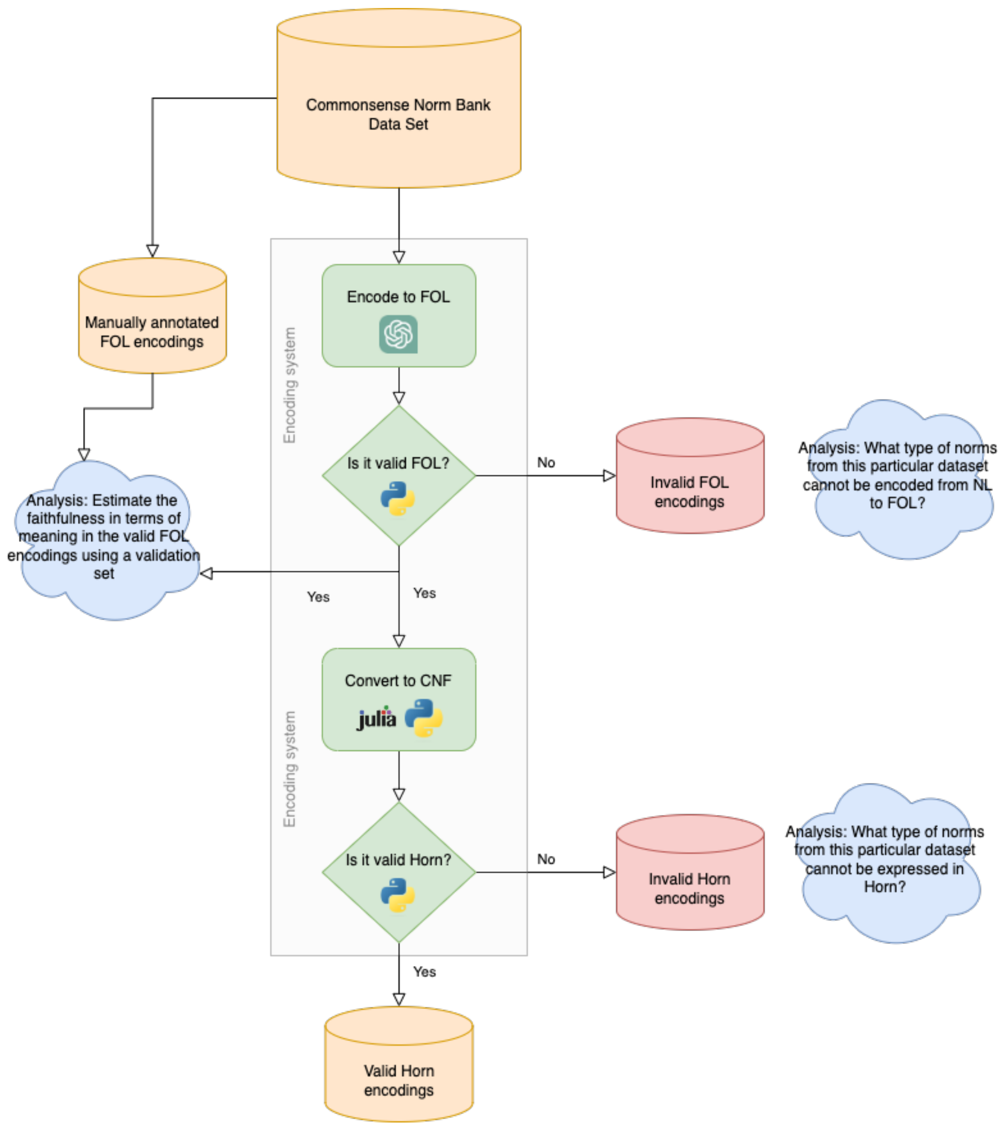


Figure 4.1: Illustration of encoding system

The analytical part of the research has three main goals. The first goal is to estimate the faithfulness of the encoding system in terms of meaning. That is, to assess whether the automatically encoded FOLs represent the intended meaning of the original norms in a sufficient manner. The second goal is to identify what types of norms GPT-4 encodes to FOL inaccurately. The third goal is to identify what types of norms that cannot be encoded to Horn clause representations. These three analyses are performed separately on a specific set of norms processed by the encoding system. In the faithfulness evaluation, predefined categories of faithfulness are used, in which each encoding will be classified. In the other two identification analyses, specific observations about each individual norm are gradually merged into more general patterns.

4.1.1 Delimitation of FOL Encodings

In order to obtain consistent encoding from the GPT-4 model that will generate definite Horn clauses, we choose to delimit the syntactic properties of the FOL encodings. As the NL sentences are moral norms we expect the sentences to be semi-structured, consisting of a situation and a moral judgment over said situation. The FOL encoding format is defined as follows:

To adhere to the definite Horn clause format, each encoding should be an implication where the antecedent represents the situation as a conjunction of predicates. We call this part of the norm “the norm part”. The consequent should represent the moral judgment of the situation using a predicate *evaluation*, with the moral judgment as a constant, such as *evaluation(GOOD)* or *evaluation(BAD)*. We call this part of the norm “the evaluation part”. An example of a FOL and its respective norm and evaluation part is shown in Example 4.1.1.

FOL: $\forall x(F(x) \land G(x) \rightarrow evaluation(GOOD))$
Norm part: $\forall x(F(x) \land G(x))$
Evaluation part: *evaluation(GOOD)*

Example 4.1.1: FOL norm and evaluation parts

After an initial look at the norms in the Commonsense Norm Bank, we found that a wide range of moral judgment terms were being used, such as “good”, “bad”, “nice”, “okay”, “not expected”, “terrible”, etc. We counted 84 different moral judgment terms used in 900 examples. Some examples of norms with different moral judgements can be seen in Example 2.4.2. Our initial implementation of the encoding system set out to encode each norm with their explicit judgment term as the moral judgment in the FOL encoding, such as *evaluation(TERRIBLE)*. However, we realized that doing this will create complications when trying to find inconsistencies at a later stage. That is, all negative judgment terms, such as “bad”, “not expected” or “terrible”, would need to be defined as conflicting with all positive judgment terms, such as “good”, “nice” or “okay”. To avoid this problem, we label all norms into one of two categories; norms that contain a positive moral judgment or norms that contain a negative moral judgment.

Positive moral judgments should be encoded with the consequent *evaluation(GOOD)*, while negative moral judgments should strictly be encoded with the consequent *evaluation(BAD)*. Such that "it's okay to help children" and "it's good to help children" should both be encoded with the evaluation part *evaluation(GOOD)*, while "it's terrible to hurt children" and "it's bad to hurt children" should both be encoded with the evaluation part *evaluation(BAD)*. We recognize that this may limit the expressiveness of the FOLs as there are norms that are difficult to categorize as having a positive or negative moral evaluation, such as the sentences "it's not hard to move away from a good group of friends", or "having sex with your friends isn't a terrible idea". However, we decided that the simplification that it provides outweighs the limitation of expressiveness. Example 4.1.2 shows how we want the NL norms to be encoded.

Norm: "it's okay to help children"
FOL: $\forall x \forall y (Child(x) \land Help(y, x) \rightarrow evaluation(GOOD))$

Norm: "it's terrible to hurt children"
FOL: $\forall x \forall y (Child(x) \land Hurt(y, x) \rightarrow evaluation(BAD))$

Example 4.1.2: Norms and their FOL representations

Each NL norm that expresses the evaluation part using either the phrase "you should" or "you should not" should be interpreted respectively as "it's good to" or "it's bad to". We show examples of norms containing these phrases and their ideal encodings in Example 4.1.3.

Norm: "you should always pay back your loans"
FOL: $\forall x \forall y (Loan(x) \land BelongsTo(x, y) \land PayBack(y, x) \rightarrow evaluation(GOOD))$

Norm: "you shouldn't give in to cravings"
FOL: $\forall x \forall y (Craving(x) \land GiveInTo(y, x) \rightarrow evaluation(BAD))$

Example 4.1.3: Norms and their FOL representation

The evaluation predicate should not be negated. If the moral implication assumes negation, this should be represented in the evaluation constant. For example, the norm "it's not good to develop bad habits", in which "not good" is considered to be the evaluation part of the norm, the evaluation part of the encoding should be expressed as *evaluation(BAD)* rather than $\neg evaluation(GOOD)$. We show the ideal encoding of the norm in Example 4.1.4.

Norm: "it's not good to develop bad habits"
FOL: $\forall x \forall y(Habit(x) \land Bad(x) \land Develop(y,x) \rightarrow evaluation(BAD))$

Example 4.1.4: Norm and its FOL representation

The logical connective symbolizing equivalence $=$ should not be used because parts of the FOL syntax verifier and parts of the CNF converter of the encoding system are not equipped to handle this structure (see Sections 5.2.2 and 5.2.3). Instead, constants or existential quantifiers should be used in the FOL encodings where equality is implied. The encoding of this structure is shown in Example 4.1.5.

Norm "it's bad to have different pets"
FOL: $\forall x \exists y \exists z(Pet(y) \land Pet(z) \land Have(x,y) \land Have(x,z) \rightarrow evaluation(BAD))$

Example 4.1.5: Norm and its FOL representation

Lastly, norms that are not grammatically correct or have nonsensical formulations should not be encoded. The sentence "it's not okay to of not examined who you are until now" is an example of such a nonsensical instance found in the Commonsense Norm Bank.

4.2 Tools and Methods

4.2.1 NL Norm Dataset

We created the NL norm dataset to use as a validation set for the automatic encoding. The NL norm dataset is a set consisting of 1001 NORM-FOL pairs. The norms are natural language sentences extracted from the yes/no category from the Commonsense Norm Bank describing a situation and a judgment over said situation. Each norm is paired with a manually constructed FOL encoding following the FOL delimitation defined in Section 4.1.1. We show example norms and their respective FOL encodings from the NL norm dataset in Example 4.2.1.

The FOL encodings in this dataset can be used as a correct baseline, to which we can compare the automatic encodings created by the encoding system. We choose to limit this set to 1001 instances, as the time and resources demanded to expand it beyond this proved too costly, and as the goal of this thesis is to obtain qualitative understanding

Norm: "you shouldn't get romantically involved with your friend's parent"
FOL: $\forall x \forall y \forall z(FriendOf(x, y) \wedge ParentOf(z, x) \wedge GetRomanticallyInvolved(y, z) \rightarrow evaluation(BAD))$

Norm: "it's ok to be addicted to harmful drugs"
FOL: $\forall x \forall y(Drug(x) \wedge Harmful(x) \wedge AddictedTo(y, x) \rightarrow evaluation(GOOD))$

Norm: "it's good to treat service workers well".
FOL: $\forall x \forall y(ServiceWorker(x) \wedge TreatWell(y, x) \rightarrow evaluation(GOOD))$

Example 4.2.1: Norms and their manual encodings from the NL norm dataset

rather than quantitative results, we do not need a large dataset to base our findings on.

4.2.2 Programming Tools

We attempted to use the encoding solution provided by Yang et al. [2023], the LogicLlama model, for the task of encoding norms to FOL. However, at the time of writing, we were not able to get the model to work. We reached out to the creators of the model, but their advice did not solve the problems we encountered. Since Yang et al. [2023] show that LogicLlama's overall ability to encode natural language into FOL does not surpass that of OpenAI's GPT-4 model, we decided to use the GPT-4 model instead [Yang et al., 2023].

To implement the encoding system, we mainly use the programming language `Python` (version 3.8.18) [Van Rossum and Drake, 2009] and the code editor Visual Studio Code. The programming languages `Julia` [Bezanson et al., 2017] and `Clingo` [Gebser et al., 2019] are integrated for specific purposes. The main packages required in our implementation are `numpy` (version 1.24.4) [Harris et al., 2020], `pandas` (version 2.0.3) [McKinney, 2010], `nltk` (version 3.8.1) [Bird et al., 2009], `openai` (version 1.11.1) and `juliacall` (version 0.9.15) [Li, 2019]. We access the GPT-4 model using OpenAI's Python API OpenAI [2024d].

A syntax verifier program created by Yang et al. [2023] is included in the encoding system as a way to verify the syntax of the FOL encodings created by GPT-4. The syntax verifier program is accessible through the public LogicLlama GitHub repository[1] [Yang et al., 2023]. Yang et al. [2023] created this program by defining a context-free grammar that coincides with the requirements of a well-formed FOL formula. The program takes an FOL formula as input and generates a string that represents the syntactic structure of the FOL formula. Using this string, the program creates a `CFG` object that can be processed by a CFG parser using Python's `NLTK` module [Bird et al., 2009]. The parser

[1] `https://github.com/gblackout/LogicLLaMA`

returns a valid `NLTK` parse tree according to the specified grammar. If no valid parse tree can be found, the FOL is considered syntactically invalid, and a `None` object is returned.

In order to convert FOLs to CNF structures, we use code from a public repository found on GitHub[2] that offers a solution to this problem [Alharthi, 2019]. The program is written in the programming language `Julia` [Bezanson et al., 2017], and we use `JuliaCall` [Li, 2019] to integrate it with our Python-based encoding system. The CNF conversion program performs a number of actions step-by-step on an FOL formula. The actions consist of eliminating implications, moving negations, standardizing variables, skolemization, and distributing conjunctions and disjunctions. The program returns the FOL formula structured in CNF format.

4.2.3 Prompt Engineering

In order to get the GPT-4 model to encode NL norms to FOL we provide it with a prompt, which is a message that instructs the model to perform a certain task. OpenAI and Microsoft provide information on the different techniques used when constructing such a prompt to get better results from LLMs, a method often referred to as prompt engineering [OpenAI, 2024b, Microsoft, 2024a]. The techniques we choose to consider are general system message adjustments, few-shot learning, self-correction, fine-tuning, chain-of-thought prompting, and temperature adjustment. Each of these are described in detail below.

The prompt system message provides the large language model with precise instructions on what we want it to do [Microsoft, 2024a]. A prompt system message can be either simple, giving the language model a clear and concise instruction, or more comprehensive, giving the model a more detailed instruction that may yield less precise answers.

Few-shot learning is a prompt strategy in which a set of examples is provided in the prompt input messages that show user input along with the preferred system response. This is provided to show, rather than tell, how the LLM should respond to a certain request, both in terms of content and format [OpenAI, 2024b].

Self-correction is a form of multi-turn dialogue in which the interaction between the user and the LLM unfolds over multiple turns, with each turn building on the previous one [Microsoft, 2024b]. Wu and Hu [2023] found in their research on the exploration of prompt engineering for language translation that this method tends to improve the quality of translations when using GPT-3.5.

OpenAI offers the ability to fine-tune its models, which means that you can provide domain-specific training data to the model that could improve its performance [OpenAI, 2024a]. In practice, this would work as an expansion of the few-shot learning described above, but instead of including three examples in the prompt, the model would be trained

[2] `https://github.com/wjdanalharthi/First-order-Logic-resolution`

on a much larger number of examples.

Chain-of-thought (CoT) prompting is, similar to self-correction, a multi-turn dialogue approach that divides a task into multiple smaller steps by creating several prompts each instructing one step toward the ultimate goal [Microsoft, 2024a]. According to Wei et al. [2023], CoT prompting of LLMs significantly improves the model's ability to perform reasoning.

The temperature parameter in the GPT models determines the randomness of the output. The temperature can be set to values between 0.0 and 2.0, where values closer to 0.0 create strict, concrete, and focused responses, while values closer to 2.0 introduce a larger degree of randomness in the model and create more divergent responses [Microsoft, 2024a].

4.2.4 Sentiment Analysis

Sentiment analysis (SA) is a text classification tool that aims to study people's opinions, attitudes, and emotions. It is most commonly used to identify opinions expressed in written customer reviews. Sentiment analysis can be performed on three different levels: document-level, sentence-level, and aspect-level. In document-level and sentence-level SA, the sentiment analysis model tries to classify whether a text expresses positive or negative sentiment without separating opinions over different entities or aspects within the same text. The only difference between the two comes down to the length of the text. Document-level SA considers a whole document, whereas sentence-level SA only considers a sentence. Aspect-level SA differs from document- and sentence-level SA in that it aims to classify the sentiment with respect to the specific entities over which they express the sentiment [Medhat et al., 2014].

The DistilBERT Base Model[3] is an example of a pre-trained language model. DistilBERT is a distilled version of BERT, which is pre-trained for language understanding [Devlin et al., 2019]. DistilBERT has 97% of BERT's language understanding capabilities, with a parameter size reduction of 40% compared to the original model [Sanh et al., 2020]. This ensures that the DistilBERT model runs 60% faster than the original BERT model, while still maintaining over 95% of BERT's performance capabilities [Hugging Face, 2024].

We use the DistilBERT base model to create a sentiment analysis model to enhance the encoding system. We chose this model in particular as it was recommended by HuggingFace for most use cases of sentiment analysis [Pascual, 2022].

[3]The model can be found on HuggingFace: `https://huggingface.co/distilbert/distilbert-base-uncased`

4.2.5 Answer Set Programming

We use ASP to create an inconsistency checker that can find conflict within a set of Horn-clause-represented norms. We choose to use ASP over any other logic programming language as it provides the opportunity to use strong negation, which is convenient when expressing Horn clauses containing negated atomic formulas. This task could likely be solved using other logic programming languages, but since this task is considered more of a proof of concept rather than a fully optimized part of the encoding system, we do not explore more than one solution.

4.2.6 Manual Assessment

To evaluate and analyze the encoding system implemented in the developmental part, we choose to manually assess the FOL and Horn clause outputs of the system in the analytical part. Although manual assessment is time-consuming and prone to human errors, we choose this over any other automatic semantic evaluation technique, as current automatic alternatives in this field have been shown to be lacking compared to human evaluation [Roy et al., 2021].

In the next chapter, we describe how we use the tools and methods outlined in this chapter to implement our solutions.

Chapter 5

Experiments

In this chapter, we outline the experimental part of the thesis and describe the implementation of practical components. The methods and tools used in the implementation are presented in Chapter 4.

We begin with Section 5.1, where we discuss the creation of the NL norm dataset. Section 5.2 includes a detailed description of how each part of the encoding system's overall encoding process was implemented. Sections 5.3 and 5.4 describe the implementation of methods meant to improve and expand the encoding system and its applications. These are the sentiment analysis model and the inconsistency checker. Section 5.5 describes how the enconding system's ability to capture sentiment in its FOL encodings was assessed[1].

5.1 NL Norm Dataset

We extracted a subset of the Commonsense Norm Bank to create the NL norm dataset described in Section 4.2.1. The NL norm dataset consists of 1001 norms selected from the yes/no category of the Commonsense Norm Bank in a semi-structured manner. For every 100 norms in the yes/no category, we copied the first 10 to 11 norms into the NL norm dataset until we reached 1001 instances. Sentences that were discovered to be nonsensical or not considered norms were deleted from the NL norm dataset and new norms from the yes/no category of the Commonsense Norm Bank were added to replace them.

We extended the dataset by manually encoding the 1001 norms to their respective FOL representations according to the FOL delimitations defined in Section 4.1.1. This was done before the execution of the final round of automatic encoding and resulted in a NL-FOL validation set that was used in the evaluation of the encoding system. This ensured that we could consider the automatically encoded norms against the manually encoded norms. This was done to counteract confirmation bias when evaluating the accuracy of the automatic encodings. The manual FOL encoding was done by the author

[1]The code for all components can be found on GitHub: `https://github.com/emmabjor/nl-horn-master/tree/main`

and a bachelor's student in information science, both of whom have taken several courses in logic at the University of Bergen.

5.2 Encoding System

The encoding system converts natural language norms into first-order Horn representations. The implementation of the system was organized into separate connected parts by dividing the encoding task into multiple steps, as shown in Figure 4.1. The first step in this system is the encoding of an NL norm to a general FOL representation. The second step determines whether the FOL encoding is a well-formed formula. FOL encodings considered to not be well-formed are stored for analysis. All well-formed formulas proceed to the third step, where the FOLs are converted into conjunctive normal form (CNF). In the fourth step, we consider whether the CNF-converted FOLs follow the first-order Horn requirements. In the end, we are left with a set of FOL encodings considered to be Horn, a set of FOL encodings not considered to be Horn, and a set of invalid FOL encodings. The valid Horn encodings can be further processed to find inconsistencies in the set, while the invalid FOL and Horn encodings are stored for analysis.

The following subsections describe each step of the encoding system in detail. After implementing the encoding system as a complete and working pipeline, the system was used to encode all norms from the NL norm dataset to FOH to create the results used in the analyses. It is important to note that, as the implementation of the encoding system was an iterative process, some decisions were made at later stages in the implementation process, which are reflected in some of the descriptions and examples from earlier iterations. In the beginning of the implementation process, we talked about "translating" norms to first-order Horn clauses instead of "encoding" norms, and this is evident in some of the explanations and figures throughout this thesis. The decision to limit the evaluation part of the FOL encodings to strictly *evaluation(GOOD)* or *evaluation(BAD)* was also made at a later stage in the process, and therefore some examples may include other evaluation constants than strictly these.

5.2.1 Natural Language to First-Order Logic Encoding

The first step in the encoding system is to encode norms from natural language English to well-formed FOL formulas. This is done by instructing OpenAI's GPT-4 model using a prompt. The following section describes in detail the prompt strategy that was used in this task. We then describe the system settings and context used when creating the final FOL encodings of the NL dataset that were further processed and analyzed in the thesis.

Prompt Engineering Techniques

Much of the work in implementing the encoding system consisted of adjusting and specifying the encoding instructions provided to GPT-4 through prompt engineering. This prompt engineering process was performed and reviewed iteratively using subsets of the NL norm dataset. As prompt engineering is a process of trial and error using different techniques and prompt formulations, this phase brought some insight that affected the choices made in the remaining implementation process.

We constructed approximately 12 different variations of prompts using a combination of the techniques described in Section 4.2.3. We tested the different prompt variations on 10 to 60 NL norms and manually reviewed the resulting encodings to find the combination of techniques that yielded the best results based on syntactic and semantic validity. Fatal errors were recognized, and the prompt was reformulated to avoid the errors in the next variation. Example 5.2.1 shows one such reformulation that was done.

In one of the early prompt iterations, the phrase "cannot afford" was encoded to the predicate "$CannotAfford$" instead of the more logically substantial predicate "$\neg Afford$". We therefore added the following sentence to the prompt: *Use the negation-symbol in the antecedent when predicates express negation, such that "do/can/will not $< Predicate >$" encodes to "$\neg < Predicate >$".*

Example 5.2.1

In the remainder of this section, we will describe in detail how we tested each of the prompt strategy techniques described in Section 4.2.3, and consequently how we implemented the ones we chose to include.

Prompt system message: In the prompt system message we specify the symbols the encodings should contain, the desired format of the output, and clarifications on how the model should or should not encode specified cases. We found that a more detailed system instruction describing multiple do's and don'ts yielded the best results. The instructions we chose to include were based on multiple manual reviews of the results of the different system message variations. Listings 5.1 and 5.2 show the evolution from the first system message to one of the final system messages.

```
You are an assistant that translates English natural language sentences
into well formulated first-order logic formulas.
Strictly use these signs and operators: '∨', '∧', '→', '↔', '∀', '∃',
'¬', '(', ')', ','.
```

Listing 5.1: System message for first prompt variation

```
You are a system tasked with translating sentences from natural
language English to first-order logic formulas.
Ensure precise and valid first-order formulas that convey the explicit
sentiment of the original sentence as accurately as possible.
Use the following symbols: '∨', '∧', '→', '↔', '∀', '∃', '¬', '(', ')',
','.
Use variables in lowercase (e.g., "x", "y", "z"), constants in
uppercase (e.g., "BAD", "EMMA", "FACEBOOK"), and predicates (e.g.,
"People(x)", "Friends(EMMA,y)", "Explains(x,y,z)").
Ensure that the scope of a predicate does not contain other predicates
or nested structures.
Don't negate variables or constants.
Each formula should be formulated as an implication, where the
conclusion should always be "evaluation(GOOD)" or "evaluation(BAD)".
Use the negation-symbol in the antecedent when predicates express
negation, such that "do/can/will not <predicate>" translates to
"¬<Predicate>()".
When "you should" is stated it implies the consequence
evaluation(GOOD)".
When "you shouln't" is stated it implies the consequence
evaluation(BAD)".
When something is positively stated, such as "it's ok" or "it's
expected", it implies the consequence "evaluation(GOOD)".
When something is negatively stated, such as "it's rude" or "it's not
expected", it implies the consequence "evaluation(BAD)".
When "others" or "someone else" is mentioned, these should be
represented with existential quantifier, such that "other people"
translates to "∃x(People(x)).
When "sometimes" is expressed, all quantifiers should be existential,
such that "sometimes people eat meat" translates to "∃x∃y(People(x) ∧
Meat(y) ∧ Eat(x,y))".
Make sure each action/property that has an actor is connected to its
actor, e.g. "your things" translates to "Thing(x) ∧ BelongsTo(x,y)", or
"your family" translates to "FamilyOf(x,y)".
It's better to divide the formula into several predicates that express
separate parts rather than combining all information into a single
predicate, e.g. "old man" translates to "Man(x) ∧ Old(x)".
Double check that all requirements specified above are met before
giving your answer.
Try reading out loud your translation to make sure the sentiment aligns
with the English sentence before giving your answer.
Adjust if sentiment does not align.
```

Listing 5.2: System message for final prompt variation

Few-shot learning: We carefully constructed three examples for each prompt that show different NL norms and the ideal FOL encoding for each norm. We chose the examples based on the results from previous prompt testing that showed which cases needed to be exemplified, rather than just explained in the system message. We tested variations using zero-, one-, three-, and five-shot prompts and found that including three examples proved to be the middle ground between covering enough cases and avoiding overfitting. Overfitting occurs when a model is too closely fit to the training data in such a way that it does not generealize well to new data [Müller and Guido, 2016, p. 28]. The examples used in one of the final prompts can be seen in Listing 5.3.

```
{"role": "user", "content":
"it's fine to care about your family's history."},
{"role": "assistant", "content":
"∀x∀y∀z(FamilyOf(x,y) ∧ HistoryOf(z,x) ∧ CaresAbout(y,z) →
evaluation(GOOD))"},

{"role": "user", "content":
"you shouldn't talk to other people about your fetishes."},
{"role": "assistant", "content":
"∀x∀y∃z(Fetish(x) ∧ BelongsTo(x,y) ∧ People(z) ∧ TalkToAbout(y,z,x) →
evaluation(BAD))"},

{"role": "user", "content":
"it's not ok to be frustrated when you work at the University of
Bergen."},
{"role": "assistant", "content":
"∀x(WorksAt(x, UNIVERSITY_OF_BERGEN) ∧ Frustrated(x) → evaluation(BAD))"}
```

Listing 5.3: Example of a three-shot prompt strategy

Self-correction: We tested out the self-correction technique by creating two separate prompts. In the first prompt, referred to as "the initial prompt", we ask the GPT-4 model to simply encode the NL norm to FOL according to the FOL delimitation specified in Section 4.1.1. In the second prompt, referred to as "the adjustment prompt", we ask the GPT-4 model to improve and correct the result from the initial prompting. By giving the model a second attempt to improve an existing formula rather than creating one from scratch we give the model more time to reevaluate its own encoding and catch missing or incorrect parts of the initial encoding [OpenAI, 2024b]. In this second adjustment prompt, the GPT-4 model gets a similar system message as in the initial prompt, but with the focus on correcting an already existing formula to make it as syntactically correct and semantically equivalent to the original NL norm as possible. This adjustment prompt also makes use of few-shot learning by providing three new examples showing how we want GPT-4 to correct the encodings. If an encoding does not need correction, the model is instructed to return the formula as is. Listing 5.8 show the adjustment prompt that was created used to correct the result of the initial prompt seen in Listing 5.7.

We manually reviewed how well GPT-4 was able to encode norms using the initial prompt alone versus using the initial prompt followed by the adjustment prompt. We found that the encoding results improved when the model was asked to adjust the initial encodings. In one of the early prompt variations, we found that 12 of 33 FOL encodings were improved, semantically and/or syntactically, by the adjustment prompt. Six FOL encodings became worse, and the remaining 15 neither improved nor worsened from the initial prompt to the adjustment prompt. These results convinced us that applying self-correction using an adjustment prompt would prove valuable for the encoding system. It is important to note that as the phrasing of the prompts improved and the properties of the encodings were adjusted, the gap between the quality of the results from the initial prompt alone and the quality of the results from the initial prompt coupled with the adjustment prompt was reduced. In the final prompt variations (see Listings 5.7 and 5.8), we found that for some batches of norms the adjustment prompt improved just as many encodings as it worsened. For a few batches, the adjustment prompt worsened even more than it improved. We show an example of how the adjustment prompt worsened the encodings from the initial prompt in Example 5.2.2.

Norm: "it's bad for your health scare to be averted."
Initial prompt output: $\forall x \forall y(HealthScare(x) \land BelongsTo(x, y) \land Averted(x) \rightarrow evaluation(BAD))$
Adjustment prompt output: $\forall x \forall y(HealthScare(x) \land BelongsTo(x, y) \land Averted(x) \rightarrow evaluation(GOOD))$.

In this example, the adjustment prompt changed the evaluation constant from "BAD" to "GOOD", thus making the final FOL output semantically express the opposite of what the original norm intended.

Example 5.2.2

In most of the cases where the adjustment prompt worsened the result of the initial prompt, the error that occurred was consistent and predictable, and therefore also possible to correct. This will be elaborated on in Section 5.3 and in Chapter 7. For this particular reason, we decided to include the adjustment prompt in the final prompt variation.

Fine-tuning: We created two different fine-tuned models to determine whether or not this would improve the encoding results compared to using the GPT-4 model without fine-tuning. One model was fine-tuned using NL-FOL pairs (corresponding to the few-shot learning in the initial prompt), while another was fine-tuned using NL-FOL-CORRECTED tuples (corresponding to the few-shot learning in the adjustment prompt). Each of the fine-tuned models were trained on 50 manually encoded such NL-FOL pairs (or NL-FOL-CORRECTED tuples), using the GPT-3.5-turbo-1106 model. We chose to use the GPT-3-turbo model, as this was the latest model available for fine-tuning and the model recommended by OpenAI to use in fine-tuning tasks [OpenAI, 2024a]. Fine-tuning of the GPT-4 model was not an option at the time of writing.

We reviewed the fine-tuned models by prompting the models to encode 10 unseen NL norms and manually inspected the results. The results of the fine-tuned models were compared to the results of prompting the GPT-4 model on the same unseen NL norms, using a combination of the self-correction and few-shot methods described above. We call the approach that uses the prompts from Listings 5.7 and 5.8 without fine-tuning the “baseline model”. When prompting the fine-tuned models, we used the system message from the initial prompt and/or the adjustment prompt in Listings 5.7 and 5.8, without few-shot learning. We tested three different prompt strategies for the fine-tuned models. The first strategy was to use the NL-FOL fine-tuned model on its own. The second strategy was to use self-correction by having the NL-FOL-CORRECTED fine-tuned model correct the output of the initial prompting of the GPT-4 model. The last strategy was to use self-correction by having the NL-FOL-CORRECTED fine-tuned model correct the output of the NL-FOL fine-tuned model. Each of the three fine-tuning strategies resulted in only 3/10 correctly encoded FOLs, while the baseline model resulted in 7/10 correctly encoded FOLs.

On the basis of these results, we decided not to use the fine-tuned models. It is important to note that expanding the training data to include more than 50 examples, as well as evaluating more than 10 unseen examples, could yield better results. However, due to the limited time and resources, we chose not to further explore this possibility.

Chain-of-Thought (CoT): We tried using CoT prompting by splitting the encoding prompt into three separate prompts, each of which performs a step toward the overall goal of converting the NL sentence to FOL. Each step gets the original NL norm and the result of the previous step as input and is instructed to use this result to perform its task. We formulated each CoT prompt with independent system messages and three-shot learning meant to instruct GPT-4 on how to independently perform the specific tasks.

The first prompt in the CoT prompt strategy focused on identifying predicates and the moral evaluation of the norm. The first CoT prompt can be seen in Listing 5.4. The second CoT prompt was instructed to identify quantifiers and populate the predicates found in the first prompt with the appropriate variables. The second prompt can be seen in Listing 5.5. The third and final CoT prompt was tasked with formulating the final FOL as an implication using the quantifiers and predicates populated from the second prompt. The third prompt can be seen in Listing 5.6.

The results of each of the prompts were manually evaluated, and the system messages and few-shot instances were reformulated as needed. However, by splitting up an encoding task into these individual steps, it seemed a crucial holistic view, which considers the sentence as a whole, was lost. This led to worse results compared to the baseline model approach.

```
{"role":"system", "content":
"""You are a system tasked with identifying first-order predicates and
moral evaluation in a sentence.
You should identify all predicates in the sentence and write them down
in the following format: 'Predicate(<var1>, <var2>, ...)'.
Predicates are often verbs, adjectives, or nouns.
Adjective- and noun-predicates usually take one argument, for example
'_ is good' translates to 'Good(<val1>)', or 'person' translates to
'Person(<val1>)'.
Most verb-predicates requre two arguments, for example '_ works at _'
translates to WorksAt(<val1>, <val2>)'.
Some verb-predicates requre three or more arguments, for example '_
wants _ to steal _' translates to 'WantsToSteal(<val1>, <val2>,
<val3>)'.
Each sentence will have a moral evaluation, which should be written as
'evaluation(<evaluation>)'.
'you should' implies a universally good evaluation ('evaluation(GOOD)')
and 'you shouldn't' a bad evaluation ('evaluation(BAD)').
When 'it's nice', 'it's not expected', etc. is stated, it implies a
consequence interpreted as 'evaluation(NICE)',
'evaluation(NOT_EXPECTED)', etc.
You get three examples showing correct identification and the wanted
output format."""},

{"role": "user", "content":
"you shouldn't pry into your child's personal buisness."},
{"role": "assistant", "content":
"evaluation(BAD), PryInto(<val1>, <val2>), ChildOf(<val1>, <val2>),
PersonalBusinessOf(<val1>, <val2>)"},

{"role": "user", "content":
"people are not expected to love their family."},
{"role": "assistant", "content":
"evaluation(BAD), Love(<val1>, <val2>), People(<val1>),
FamilyOf(<val1>, <val2>)"},

{"role": "user", "content":
"it's ok to not join in on fun activities in New York."},
{"role": "assistant", "content":
"evaluation(GOOD), JoinIn(<val1>, <val2>), FunActivity(<val1>),
In(val1, val2)"}
```

Listing 5.4: Prompt 1 in CoT approach.

```
{"role":"system", "content":
"""You are a system tasked with identifying first-order variables,
constants and quantifiers for predicates in a sentence.
You should fill in the <vals> in the provided predicates with the
apprpriate variable, using 'x', 'y', 'z' in lowecase, or with the
appropriate constant using for example 'EMMA', 'UNIVERSITYOFBERGEN',
etc. in uppercase.
Make sure the variables are consistent across predicates.
For each variable, you need to identify the correct quantifier, which
should be written as '∀<variable>' or '∃<variable>'.
The 'evaluation'-predicate shall not be filled in.
'you' and 'it' implies quantification 'everyone'.
You get three examples showing correct identification and the wanted
output format."""},

{"role": "user", "content":
"Sentence: it's good to care about your family's history. \
Predicates: evaluation(BAD), PryInto(<val1>, <val2>), ChildOf(<val1>,
<val2>), PersonalBusinessOf(<val1>, <val2>)"},
{"role": "assistant", "content":
"∀x, ∀y, ∀z, evaluation(BAD), PryInto(x, y), ChildOf(y, x),
PersonalBusinessOf(z, y)"},

{"role": "user", "content":
"Sentence: people are not expected to love their family. \
Predicates: evaluation(BAD), Love(<val1>, <val2>), People(<val1>),
FamilyOf(<val1>, <val2>)"},
{"role": "assistant", "content":
"∀x, ∀y, evaluation(BAD), Love(x, y), People(x), FamilyOf(y, x)"},

{"role": "user", "content":
"Sentence: it's ok to not join in on fun activities in New York. \
Predicates: evaluation(GOOD), JoinIn(<val1>, <val2>),
FunActivity(<val1>), In(val1, val2)"},
{"role": "assistant", "content":
"∀x, ∀y, evaluation(GOOD), JoinIn(x, y), FunActivity(y),
In(y, NEWYORK)"}
```

Listing 5.5: Prompt 2 in CoT approach.

```
{"role":"system", "content":
"""You are a system tasked with constructing valid first-order
sentences using the predicates, variables and quantifiers provided.
You should ensure precise and valid first-order formulas that convey
the explicit sentiment of the original sentence as accurately as
possible.
You should only use the predicates, variables and quantifiers provided.
```

```
You should formulate each sentence as an implication, where the
conclusion should always be the 'evaluation'-predicate.
You should not use nested predicates.
Use the following symbols: '∨', '∧', '→', '↔', '∀', '∃', '¬', '(', ')',
','.
You get three examples showing correct identification and the wanted
output format."""},

{"role": "user", "content":
"Sentence: it's good to care about your family's history. \
Predicates: ∀x, ∀y, ∀z, evaluation(BAD), PryInto(x, y), ChildOf(y, x),
PersonalBusinessOf(z, y)"},
{"role": "assistant", "content":
"∀x∀y∀z(PryInto(y,z) ∧ ChildOf(x,y) ∧ PersonalBuisnessOf(z,x) →
evaluation(BAD))"},

{"role": "user", "content":
"Sentence: people are not expected to love their family. \
Predicates: ∀x, ∀y, evaluation(BAD), Love(x, y), People(x),
FamilyOf(y, x)"},
{"role": "assistant", "content":
"∀x∀y(Love(x,y) ∧ People(x) ∧ FamilyOf(y,x) → evaluation(BAD))"},

{"role": "user", "content":
"Sentence: it's ok to not join in on fun activities in New York. \
Predicates: ∀x, ∀y, evaluation(GOOD), JoinIn(x, y), FunActivity(y),
In(y, NEWYORK)"},
{"role": "assistant", "content":
"∀x∀y(¬JoinIn(x,y) ∧ FunActivity(y) ∧ In(y,NEWYORK) →
evaluation(GOOD))"}
```

Listing 5.6: Prompt 3 in CoT approach.

Temperature: We tested the temperature values 0.2, 0.5, 1.0, 1.2, 1.5, and 1.8 for both the initial and adjustment prompt and manually reviewed the results. We found that the default temperature value of 1.0 gave the best results.

System Settings for Running Final FOL Encodings

After considering several combinations of the different prompting techniques, we decided that the best prompt strategy was to use a detailed system message, 3-shot learning with self-correction and a temperature of 1.0. The final initial prompt can be seen in Listing 5.7 and the final adjustment prompt in Listing 5.8. As there are indefinite combinations of system message formulations and prompt techniques used in prompt engineering, we realize that there may be other combinations of techniques and approaches that would yield better results than the one we chose. Furthermore, the combinations of approaches that we reviewed were tested on a small subset of the data, anywhere between 10 to 60 sample norms. This may have led to a small sample bias in the evaluation, which means that the data sample set is too small to ensure that the results are generalizable for the rest of the dataset [Van de Schoot and Miočević, 2020, p. 51].

```
{"role":"system", "content":
"""You are a system tasked with translating sentences from natural
language English to first-order logic formulas.
Ensure precise and valid first-order formulas that convey the explicit
sentiment of the original sentence as accurately as possible.
Use the following symbols: '∨', '∧', '→', '↔', '∀', '∃', '¬', '(', ')',
','.
Use variables in lowercase (e.g., "x", "y", "z"), constants in
uppercase (e.g., "BAD", "EMMA", "FACEBOOK"), and predicates (e.g.,
"People(x)", "Friends(EMMA,y)", "Explains(x,y,z)").
Ensure that the scope of a predicate does not contain other predicates
or nested structures.
Don't negate variables or constants.
Each formula should be formulated as an implication, where the
conclusion should always be "evaluation(GOOD)" or "evaluation(BAD)".
Use the negation-symbol in the antecedent when predicates express
negation, such that "do/can/will not <predicate>" translates to
"¬<Predicate>()".
When "you should" is stated it implies the consequence
"evaluation(GOOD)".
When "you shouln't" is stated it implies the consequence
"evaluation(BAD)".
When something is positively stated, such as "it's ok" or "it's
expected", it implies the consequence "evaluation(GOOD)".
When something is negatively stated, such as "it's rude" or "it's not
expected", it implies the consequence "evaluation(BAD)".
When "others" or "someone else" is mentioned, these should be
represented with existential quantifier, such that "other people"
translates to "∃x(People(x)).
When "sometimes" is expressed, all quantifiers should be existential,
such that "sometimes people eat meat" translates to "∃x∃y(People(x) ∧
Meat(y) ∧ Eat(x,y))".
Make sure each action/property that has an actor is connected to its
actor, e.g. "your things" translates to "Thing(x) ∧ BelongsTo(x,y)", or
"your family" translates to "FamilyOf(x,y)".
It's better to divide the formula into several predicates that express
separate parts rather than combining all information into a single
predicate, e.g. "old man" translates to "Man(x) ∧ Old(x)".
Double check that all requirements specified above are met before
giving your answer.
Try reading out loud your translation to make sure the sentiment aligns
with the English sentence before giving your answer.
```

```
Adjust if sentiment does not align.
You get three examples that show correct translations and the wanted
output format."""},

{"role": "user", "content":
"it's fine to care about your family's history."},
{"role": "assistant", "content":
"∀x∀y∀z(FamilyOf(x,y) ∧ HistoryOf(z,x) ∧ CaresAbout(y,z) →
evaluation(GOOD))"},

{"role": "user", "content":
"you shouldn't talk to other people about your fetishes."},
{"role": "assistant", "content":
"∀x∀y∃z(Fetish(x) ∧ BelongsTo(x,y) ∧ People(z) ∧ TalkToAbout(y,z,x) →
evaluation(BAD))"},

{"role": "user", "content":
"it's not ok to be frustrated when you work at the University of
Bergen."},
{"role": "assistant", "content":
"∀x(WorksAt(x, UNIVERSITY_OF_BERGEN) ∧ Frustrated(x) →
evaluation(BAD))"}
```

Listing 5.7: Initial prompt

```
{"role":"system", "content":
"""You are a system tasked with correcting a first-order formula to
make sure it captures the explicit sentiment expressed in the natural
language sentence as accurately and complete as logically possible.
If the first-order formula does not need correction, you should answer
with the formula exactly as it is without any changes or explanations.
Do not change the evaluation constant unless the evaluation sentiment
expressed in the natural language sentence is different from the one in
the provided formula.
Use the following symbols: "∨", "∧", "→", "↔", "∀", "∃", "¬", "(", ")",
",".
Use variables in lowercase (e.g., "x", "y", "z"), constants in
uppercase (e.g., "BAD", "EMMA", "FACEBOOK"), and predicates (e.g.,
"People(x)", "Friends(EMMA,y)", "Explains(x,y,z)").
Ensure that the scope of a predicate does not contain other predicates
or any nested structures.
Don't negate variables or constants.
Each formula should be formulated as an implication, where the
conclusion should always be "evaluation(GOOD)" or "evaluation(BAD)".
Use the negation-symbol in the antecedent when predicates express
negation, such that "do/can/will not <predicate>" translates to
"¬<Predicate>()".
```

```
When "you should" is stated it implies the consequence
"evaluation(GOOD)".
When "you shouln't" is stated it implies the consequence
evaluation(BAD)".
When something is positively stated, such as "it's ok" or "it's
expected", it implies the consequence "evaluation(GOOD)".
When something is negatively stated, such as "it's rude" or "it's not
expected", it implies the consequence "evaluation(BAD)".
When "others" or "someone else" is mentioned, this should be
represented with existential quantifier, such that "other people"
translates to "∃x(People(x))".
When "sometimes" is expressed, all quantifiers should be existential,
such that "sometimes people eat meat" translates to "∃x∃y(People(x) ∧
Meat(y) ∧ Eat(x,y))".
Make sure each action/property that has an actor is connected to its
actor, e.g. "someone's things" translates to "Thing(x) ∧
BelongsTo(x,y)", or "your family" translates to "FamilyOf(x,y)".
It's better to divide the formula into several predicates that express
separate parts rather than combining all information into a single
predicate, e.g. "old man" translates to "Man(x) ∧ Old(x)".
Double check that all requirements specified above are met before
giving your answer.
Try reading out loud your translation to make sure the sentiment aligns
with the English sentence before giving your answer.
Adjust if sentiment does not align.
Once again, if the first-order formula does not need correction, you
should answer with the formula exactly as it is without any changes or
explanations.
You get three examples that show correct adjustment and the wanted
output format."""},

{"role": "user", "content":
"NL English sentence: it is ok to kiss someone else's significant
other.
Formula: ∃x∀y∀z(HasSignificantOther(x,y) ∧ Kiss(z,y) →
evaluation(GOOD))"},
{"role": "assistant", "content":
"∃x∀y∀z(HasSignificantOther(x,y) ∧ Kiss(z,y) → evaluation(GOOD))"},

{"role": "user", "content":
"NL English sentence: you shouldn't drink someone's beer.
Formula: ∀x∀y(People(x) ∧ Beer(y) ∧ Drink(x,y) → evaluation(BAD))"},
{"role": "assistant", "content":
"∀x∀y∀z(Beer(y) ∧ BelongsTo(y,x) ∧ Drink(z,y) → evaluation(BAD))"},

{"role": "user", "content":
```

```
"NL English sentence: it's ok to not join in on fun activities.
Formula: ∀x∀y(Activity(x) ∧ Fun(x) ∧ NotJoinInOn(y,x) →
evaluation(GOOD))"},
{"role": "assistant", "content":
"∀x∀y(Activity(x) ∧ Fun(x) ∧ ¬JoinInOn(y,x) → evaluation(GOOD))"}
```

Listing 5.8: Adjustment prompt

The 1001 norms from the NL norm dataset were encoded into FOL using the final prompt variations seen in Listings 5.7 and 5.8. The norms were first encoded by GPT-4 using the initial prompt from Listing 5.7. The encoding result of this interaction was then corrected by GPT-4 using the adjustment prompt from Listing 5.8. We executed the encoding by dividing the norms into 10 batches of 100 to 101 norms in each batch. Each batch was run through the FOL encoding part of the system. The FOL encoding was executed between 8. February 2024 and 12. February 2024. Each batch had a runtime of 8 to 11 minutes and an average cost of $5,432, resulting in a cumulative runtime of 93 minutes and a total cost of $54,38. The batches were run on a MacBook Pro 13-inch 2020 model.

5.2.2 First-Order Logic Syntax Verifier

As the GPT-4 model used to encode NL norms to FOL can generate syntactically invalid FOL formulas, that is, FOL formulas that are not well-formed, it is important to have a separate syntax checker that can verify which encodings are well-formed and which are not. This is the second step of the encoding system. We utilize the FOL syntax verifier created by Yang et al. [2023] to solve this task by labeling those that cannot be parsed by the syntax verifier as *Syntactically invalid* encodings. We see an example of an FOL encoding that is labeled *Syntactically invalid* by the syntax verifier in Example 5.2.3.

Norm: "you should not always get registration information when in an accident".
FOL: $\forall x \forall y(Accident(x) \land In(y,x) \land Always(Get(y, RegistrationInformation(x)))$
$\rightarrow evaluation(BAD))$

The reason why this specific FOL encoding should be labeled *Syntactically Invalid* is that it contains a nested predicate $Always(Get(y, RegistrationInformation(x)))$, which violates the requirements of a WFF.

Example 5.2.3: FOL encoding labeled *Syntactically invalid* by the FOL syntax verifier

One limitation of this verifier is that it evaluates FOLs containing conjunctions of valid implications, where all implications use the same quantified variable, as invalid. These are FOLs such as Formula 5.1, where F, G, and H are predicates and x is a variable symbol that is used in two separate implications in a conjunction. In the definition of what constitutes a WFF defined in Section 2.2.4, there are no specifications that make this structure not a WFF.

$$\forall x(F(x) \to G(x)) \land \forall x(H(x) \to G(x)) \tag{5.1}$$

5.2.3 Converting First-Order Formulas to Conjunctive Normal Form

After encoding the natural language norms to FOL, and filtering out the FOLs that are not well-formed using a syntax verifier, we convert the well-formed FOLs to CNF. By converting the FOLs to CNF, we can at a later stage easily determine whether or not the FOLs are considered valid Horn Clauses. As explained in Chapter 2.2.3, a formula is in CNF if and only if it consists of a conjunction of one or more disjunctions.

The CNF conversion program[2] performs a number of actions step-by-step on an FOL formula. The actions consist of eliminating implications, moving negations, standardizing variables, skolemization, and distributing conjunctions and disjunctions. Each of these actions, in addition to the preprocessing required prior to the CNF conversion, will be explained in detail below.

Preprocessing

The CNF conversion program takes as input a list of well-formed FOL formulas and converts each FOL to CNF. The program requires that the FOL strings in the input list are formatted as follows. The FOLs have to be WFF, all quantifiers have to be on the leftmost side of the formula, all variables within a formula have to be unique, and the FOLs have to be expressed using a defined set of operator symbols. Through preprocessing, we ensure that each FOL follows this certain format before converting it to CNF. We describe the implemented preprocessing steps below using an example.

An NL norm, "a girl should be friends with some girls and enemies with all boys", can be represented in FOL as in Formula 5.2. This FOL example represents the structure of the output generated by the GPT-4 encodings. Using this FOL example, we will show how we process the encoded FOLs before converting them to CNF.

$$\forall x(\exists y(Girl(x) \land Girl(y) \land Friends(x,y)) \ \land \forall y(Boy(y) \ \land Enemies(x,y)) \to evaluation(GOOD)) \tag{5.2}$$

[2]Found on GitHub: `https://github.com/wjdanalharthi/First-order-Logic-resolution`

The first step in the preprocessing is to rename all duplicate variables. We iterate through each character in the formula and give each quantified variable its own unique name. The parentheses indicate for each variable which quantifier it lies within the scope of and the variables are renamed thereafter. The result of renaming all the variables in Formula 5.2 can be seen in Formula 5.3

$$\forall x0(\exists y0(Girl(x0) \wedge Girl(y0) \wedge Friends(x0, y0)) \ \wedge \forall y1(Boy(y1) \wedge Enemies(x0, y1)) \rightarrow evaluation(GOOD)) \tag{5.3}$$

In the next step, we move all quantifiers to the left side of the formula by using simple string manipulation. The result of applying this step to Formula 5.3 can be seen in Formula 5.4.

$$\forall x0 \exists y0 \forall y1((Girl(x0) \wedge Girl(y0) \wedge Friends(x0, y0)) \ \wedge (Boy(y1) \wedge Enemies(x0, y1)) \rightarrow evaluation(GOOD)) \tag{5.4}$$

The FOL syntax verfier described in Section 5.2.2 required the FOL string to consist of the following operator symbols: $\forall, \exists, \vee, \wedge, \rightarrow, \leftrightarrow, \neg$, while the CNF conversion program requires the operator symbols to be, respectively $\forall, \exists, |, \&, ==>, <==>, \sim$. We used the built-in Python method `replace` to replace each operator instance with its respective alternative representation. After applying this method to Formula 5.4, we get our final preprocessing output, seen in Formula 5.5.

$$\begin{gathered} \forall x0 \exists y0 \forall y1((Girl(x0) \ \& \ Girl(y0) \ \& \ Friends(x0, y0)) \ \& \ (Boy(y1) \ \& \\ Enemies(x0, y1)) ==> evaluation(GOOD)) \end{gathered} \tag{5.5}$$

CNF Conversion Program

The CNF converter program takes as input a list of strings that express FOL formulas in the format exemplified in Formula 5.5. For each FOL string in the input list, the conversion program starts by tokenizing and parsing the string to a nested array of elements, each element separated according to the parentheses' scopes in the FOL string.

Next, each element in the array is converted to an instance of Clause or Quantifier, which are defined class objects, based on the operators that exist in the element. Each Clause instance is then converted to CNF by eliminating implications from the clause, then moving negation inward, and finally distributing AND over OR in the clause. We describe each of these steps in detail.

The process of eliminating implications is based on the operator of the clause. If the clause's operator is conditional (==>) it transforms the clause into a disjunction of the negation of the antecedent and the consequent. If the clause's operator is a biconditional (<==>) it transforms the clause into a conjunction of two disjunctions. If the clause's operator is not an implication, it simply returns the clause with its arguments.

The process of moving negation inward is based on the clause's operator and the argument within the clause. If the clause's operator is a negation (¬), and the negated argument's operator is a Quantifier, then the quantifier is replaced by the alternative quantifier and the negation moved one place to the right. If the clause's operator is a negation and the negated argument's operator is a conjunction (&) it transforms the clause into a disjunction of the negations of the arguments. If the clause's operator is a negation and the negated argument's operator is a disjunction (|) it transforms the clause into a conjunction of the negations of the arguments. If the clause's operator is negated and the clause is a symbol or has no arguments, it simply returns the clause. If the clause's operator is not a negation, the clause is returned as is.

The process of distributing AND over OR is based on the clause's operator. If the clause's operator is a disjunction (|) it finds the first argument that is a conjunction and distributes the conjunction over the rest of the arguments. If there is no such argument, it simply returns the clause. If the clause's operator is a conjunction (&) it distributes AND over OR in each of the arguments. If the clause's operator is neither a disjunction nor a conjunction, it simply returns the clause.

Lastly, the program performs skolemization on the CNF converted clause. Skolemization, as described in Section 2.2.3 is the process of removing existential quantifiers from an FOL by replacing each existential quantifier with a Skolem function or a Skolem constant. When an existential quantifier is found in the clause, it checks if there are any universal quantifiers in its scope. If there are no universal quantifiers in scope, it generates a new unique Skolem constant and replaces the existentially quantified variable with this constant. If there are universal quantifiers in scope, it generates a new unique Skolem function and replaces the existentially quantified variables with this function. In addition, new unique variables are generated for each universal quantifier and its associated variables, and the universal quantifier itself is removed from the formula.

In the end, we are left with a skolemized FOL on a CNF format, stripped of quantifiers, and with unique variables. When the CNF conversion program is applied to Formula 5.5, the result can be seen in Formula 5.6.

$$evaluation(GOOD) \mid \sim Girl(v0) \mid \sim Girl(f1(v0)) \mid \sim Friends(v0, v1) \mid \sim Boy(v2) \mid \\ \sim Enemies(v0, v2) \tag{5.6}$$

5.2.4 Horn Clause Validation

The final part of the encoding system involves automatically checking whether the CNF-converted FOLs are valid Horn clauses or not. As explained in Section 2.2.4, valid Horn clauses are clauses in normal form with zero or one positive atomic formula. Valid Horn sentences are conjunctions of valid Horn clauses. Since the FOL formulas have already been converted to CNF, we simply need to check if the CNF clauses contain more than one positive atomic formula. If it does, then the formula is not a valid Horn clause. Otherwise, it is considered to be a valid Horn clause.

For each syntactically valid CNF-converted FOL formula, the process of determining whether the formula is valid Horn consists of two steps. First, each conjunction of disjunctions in the CNF-converted FOL is separated into an array containing the individual disjunctions. If the formula consists of one single disjunction, the list will contain only this single element. Second, for each disjunction in the list, we count the number of positive atomic formulas. If the number is greater than one for one or more of the disjunctions, the formula is classified as an invalid Horn clause. Otherwise, the formula is classified as valid Horn. We show the FOL and CNF notations of a valid Horn clause in Example 5.2.4, and the notations for an invalid Horn clause in Example 5.2.5

FOL: $\forall x \forall y (F(x) \land G(y) \rightarrow H(x, y))$
CNF notation: $\neg F(x) \lor \neg G(y) \lor H(x, y)$

The CNF notation shows that there is only one positive atomic formula, namely *H(x,y)* which means that the Horn clause is a valid definite Horn clause.

Example 5.2.4: Valid Horn clause FOL and CNF

The FOL encoding, syntactic evaluation, CNF conversion, and Horn clause validation described in this section have been integrated into a pipeline that makes up the complete encoding system, as illustrated in Figure 4.1. The pipeline iterates through a tsv file containing NL norms in the form of strings. Each NL norm is encoded to FOL by prompting the GPT-4 model twice, first to create an initial encoding and then to adjust the result

FOL: $\forall x \forall y(F(x) \land \neg G(y) \rightarrow H(x, y))$
CNF notation: $\neg F(x) \lor G(y) \lor H(x, y)$

The CNF notation shows that there is more than one positive atomic formula, namely *G(y)* and *H(x,y)*, which means that the formula is not considered to be valid Horn.

Example 5.2.5: Invalid Horn clause FOL and CNF

of the initial encoding. The syntax of the FOL encodings is automatically evaluated. All syntactically valid FOL encodings are converted to CNF in order to ultimately be evaluated as valid Horn clauses or not. The results of running 1001 sentences from the NL norm dataset through the encoding system pipeline will be explored in further detail in Chapters 6 and 7.

5.3 Sentiment Analysis

While iteratively testing and improving the NL to FOL encoding part of the encoding system described in Section 5.2, we noticed a recurring pattern in the wrongly encoded norms. That is, the GPT-4 model had a tendency to change the evaluation constant to "GOOD" if the norm part of the sentence could be commonly considered to be positive, even though the sentence expresses it to be negative, and vice versa. We call these instances "counter-intuitive norms", and list examples of these instances in Example 5.3.1. This property of the FOL-encodings will be elaborated in more detail in Chapter 7. Based on this observation, we wanted to explore the possibility of determining the evaluation constant of a norm using alternative methods to the GPT-4 model. We explored the possibility of using a sentiment analysis model for this task.

"It is good to hurt other people"
"You shouldn't do things you like"
"You should read the Communist manifesto because it is wrong"
"It's bad to pay back money you owe"

Example 5.3.1: Counter-intuitive norms

We manually created a dataset of 992 NORM-LABEL pairs where each instance consists of a norm from the Commonsense Norm Bank and a label, either 1 (positive) or 0 (negative), representing the sentences' interpreted evaluation sentiment. The selected norms were different from those used in the NL norm dataset. We used this NORM-LABEL dataset to fine-tune the language model DistilBERT, creating a sentiment analysis model capable of predicting the evaluation constant of a norm. The fine-tuned SA model takes an NL norm as input and outputs the appropriate label, 0 or 1, corresponding to the sentiment, "GOOD" or "BAD", of the evaluation part of the norm.

To determine how well the sentiment analysis model is able to label norms compared to the GPT-4 model used in the encoding system, we needed to know what the correct labeling should be for each of the 1001 norms encoded by GPT. We labeled the 1001 norms from the NL norm dataset with their correct evaluation constant, "GOOD" or "BAD", based on the evaluation constant used in the manually encoded baseline FOLs. Using this, we could compare how well the sentiment analysis model and GPT-4 encodings performed when applied to the same 1001 norms. The results of this comparison will be further elaborated on in Chapter 6.

5.4 Inconsistency Checker

While manually encoding norms to FOL representations, we noticed that several of the norms were inconsistent in the manner that two or more norms consisted of the same norm part with different, conflicting evaluation parts. Example 5.4.1 shows intuitively conflicting norms. We want to be able to identify these types of conflicts if they exist in a set of Horn clauses representing norms.

"it is wrong for a teacher to bully a student"
"it is right for a teacher to bully a student"

Example 5.4.1: Norms that are inconsistent with each other

We have implemented a solution to finding such Horn clause inconsistencies using Clingo, one of the most widely used ASP systems. All valid Horn clause representations encoded from the NL norm dataset were represented as ASP facts and rules and written to a Clingo script. When this script is run, it will automatically tell us if the facts defined in the script are satisfiable or unsatisfiable according to the defined rules and which norms cause the conflict. A set of facts that are unsatisfiable is a set of facts that contain inconsistencies, meaning that two or more facts cannot exist at the same time according to the defined rules. We have limited our encoding system to strictly create logical implications where the antecedent represents the norm part, which expresses the behavior that should or should not be encouraged, and the consequent represents the evaluation part of the norm, which expresses whether or not the norm part is encouraged or not. An inconsistency in these types of encodings thus occurs when two Horn clause encodings consist of the same norm part, with different evaluation constants in the evaluation part.

Example 5.4.2 shows how the Horn clause representations of the norms in Example 5.4.1 are inconsistent in this manner.

Norm: “it is wrong for a teacher to bully a student”
Horn clause: $\neg Teacher(v1) \vee \neg Student(v2) \vee \neg Bullies(v1, v2) \vee evaluation(BAD)$

Norm: “it is right for a teacher to bully a student”
Horn clause: $\neg Teacher(v3) \vee \neg Student(v4) \vee \neg Bullies(v3, v4) \vee evaluation(GOOD)$

In the two Horn clauses, the norm parts are identical (except for the use of different variables), while the evaluation constants in the evaluation parts are different, making the two encodings inconsistent.

Example 5.4.2: Inconsistent norms and their Horn clause representations

$$\begin{aligned}
6\{&fact(bad, (-teacher(v1), -student(v2), -bullies(v1, v2)));\\
&fact(bad, (-teacher(v1), -bullies(v1, v2), -student(v2)));\\
&fact(bad, (-student(v2), -teacher(v1), -bullies(v1, v2)));\\
&fact(bad, (-student(v2), -bullies(v1, v2), -teacher(v1)));\\
&fact(bad, (-bullies(v1, v2), -teacher(v1), -student(v2)));\\
&fact(bad, (-bullies(v1, v2), -student(v2), -teacher(v1)))\}6
\end{aligned} \tag{5.7}$$

To create an ASP program that will be able to recognize these inconsistent structures in a set of Horn clauses, we chose to represent each valid Horn clause as one ASP fact and one ASP rule. Each Horn clause fact is constructed to be a nested predicate, named *fact*, consisting of two arguments; the evaluation constant of the Horn clause evaluation part, and a tuple containing the predicates and variables of the norm part of the Horn clause. To ground the fact, we replace the Horn clause variables with constants. We show an example of how we want a Horn clause to be represented as a fact in Example 5.4.3.

Horn clause: $\neg Teacher(v1) \vee \neg Student(v2) \vee \neg Bullies(v1, v2) \vee evaluation(BAD)$
ASP fact: $fact(bad, (-teacher(v1), -student(v2), -bullies(v1, v2))).$

Example 5.4.3: Horn clause and its ASP fact representation

In order to ensure that the order of the predicates in the norm part is unimportant, we extend the rule to include all permutations of possible orders of predicates and specify that all permutations should be considered true at the same time. The final ASP fact representation of the Horn clause from Example 5.4.2, expressing the wrongness of a teacher bullying a student, is represented in Equation 5.7. The number six informs the program that all six permutations of the fact must be considered true simultaneously. The representation of the constricting Horn clause from Example 5.4.2, which expresses the rightness of a teacher bullying a student, can be seen in Equation 5.8.

$$\begin{aligned} 6\{&fact(good, (-teacher(v3), -student(v4), -bullies(v3, v4)));\\ &fact(good, (-teacher(v3), -bullies(v3), -student(v4)));\\ &fact(good, (-student(v4), -teacher(v3), -bullies(v3, v4)));\\ &fact(good, (-student(v4), -bullies(v3, v4), -teacher(v3)));\\ &fact(good, (-bullies(v3, v4), -teacher(v3), -student(v4)));\\ &fact(good, (-bullies(v3, v4), -student(v4), -teacher(v3)))\}6 \end{aligned} \tag{5.8}$$

We ultimately want the program to recognize "illegal" combinations of facts, where Equations 5.7 and 5.8 are examples that, when combined, would constitute such an illegal combination. We explain to the program what constitutes an illegal combination by creating ASP rules. We create one rule for each Horn clause fact, the rule expressing that this Horn clause fact cannot exist alongside a conflicting fact. That is, a fact containing the same norm arguments but a different evaluation argument. In the rule, the variables of the Horn clause are represented as ASP variables in order to ensure that facts using different constants are still encompassed by the same rule. We express this by creating a headless rule with the body consisting of both the "good" and "bad" fact representations of the norm.

A headless rule, or a constraint, entails that the elements of the body cannot be true at the same time. Equation 5.9 shows the rule that accompany the facts from Equations 5.7 and 5.8. As these two facts consist of the same norm part predicates, they will be represented by the same rule. This rule says that the *fact* predicate containing the constant "GOOD", and the tuple with those distinct predicates, and another *fact* predicate containing the constant "BAD" and the tuple with those distinct predicates (though with different variables) cannot both be true at the same time. When the facts from Equations 5.7 and 5.8 and the rule from Equation 5.9 occur in the same Clingo program, the program will terminate as "unsatisfiable".

$$
\begin{gathered}
:-fact(good, (-teacher(V1), -student(V2), -bullies(V1, V2))), \\
fact(bad, (-teacher(W1), -student(W2), -bullies(W1, W2))).
\end{gathered}
\tag{5.9}
$$

We created a Python script that iteratively converts each single valid Horn clause from the automatically encoded norms from the NL norm dataset into a fact and a rule on the form described above. The facts and rules were then automatically written into a Clingo ASP program. When running the Clingo program, the ASP solvers will search for stable models within the program. Inconsistencies ensure that no stable models can be found and the program terminates as "unsatisfiable". The termination of a program as "unsatisfiable" indicates the presence of two or more Horn clauses sharing identical norm parts but differing in their evaluation constants in the Horn clause set. If there are no inconsistencies, the program will terminate as "satisfiable". We isolated the clauses creating inconsistencies by iteratively removing clauses from the script until it became satisfiable. It is important to be able to determine whether or not a set of norm representations to be used as a moral KB in an AMA contains inconsistent instances, as this is unwanted.

5.5 Evaluating Faithfulness of First-Order Logic Encodings

To assess the encoding system's faithfulness in terms of intended meaning, we manually compared the 1001 automatically encoded FOL formulas with their respective manual encodings and evaluated the similarity based on structure, interpreted meaning, and predicate names. As there are several ways of representing a sentence in FOL, the automatically encoded FOL and the manually encoded FOL did not need to be identical for them to be considered equivalent.

We created three levels of faithfulness and classified each valid automatically encoded FOL into one of these three categories. Table 5.1 shows examples from all categories. The three classification categories are:

1. *Wrong* encodings: FOLs classified as *Wrong* are syntactically valid, but the most likely interpreted meaning of the FOL does not align with the meaning of its norm. In Table 5.1, we see that the FOL encoding of the norm "you should ask too much of your family" is categorized as *Wrong* because the evaluation constant in the FOL encoding is "BAD", when it should be "GOOD", thus expressing the complete opposite of what the original norm expresses.

2. *Lacking* encodings: FOLs classified as *Lacking* are syntactically valid and express a similar interpreted meaning as the original norm. However, there are some lacking

dimensions of the FOL in terms of precision when compared to the original norm. In Table 5.1, we see that the FOL-encoding of the norm "you should expect your friends to reveal everything about themselves to you" is classified as *Lacking* because the predicate $SelfRevealing$ lacks a variable representing the concept to be revealed. We can argue that the FOL encoding still expresses the most crucial elements of the norm, but we still find it lacking in some more trivial aspects.

3. *Accurate* encodings: FOLs classified as *Accurate* are syntactically valid FOLs where the interpreteded meaning of the FOLs aligns with that of the original norm. In Table 5.1, we see that the FOL encoding of the norm "you shouldn't try to steal your friend's significant other" is an example of a norm which FOL encoding is classified as *Accurate*. We recognize the FOL to be *Accurate* as even though some of the predicate names are different, the difference is trivial and the meaning is fully captured in the encoded FOL when compared to the original norm.

The classification of the automatically encoded FOLs was done by the author, alongside the supervisor for this thesis and a bachelor's student in information science. Although the classification categories and the criteria for what types of FOLs each category should include were established and informed prior to the classification process, it is important to note that the classifications are likely not fully consistent between the three individuals. The results of the faithfulness evaluation will be further elaborated on in Chapters 6 and 7.

NL norm	Manual FOL encoding	GPT-4 FOL encoding	Classification category
you should ask too much of your family.	$\forall x \forall y(Family(x, y) \land AskTooMuch(y, x) \rightarrow evaluation(GOOD))$	$\forall x \forall y(FamilyOf(y, x) \land AsksTooMuch(x, y) \rightarrow evaluation(BAD))$	Wrong
you should expect your friends to reveal everything about themselves to you.	$\forall x \forall y \forall z(Friends(x, y) \land ExpectToReveal(x, y, z) \land About(z, y) \rightarrow evaluation(GOOD))$	$\forall x \forall y(Friends(x, y) \land \forall z(SelfRevealing(y, z) \rightarrow evaluation(GOOD)))$	Lacking
you shouldn't try to steal your friend's significant other	$\forall x \forall y \forall z(Friend(y, x) \land SignificantOther(z, y) \land TryToSteal(x, z) \rightarrow evaluation(BAD))$	$\forall x \forall y \forall z(Friends(x, y) \land HasSignificantOther(y, z) \land TriesToSteal(x, z) \rightarrow evaluation(BAD))$	Accurate

Table 5.1: Three examples of faithfulness evaluations of GPT-encoded FOLs

Chapter 6

Results

In this chapter, we describe the results of running norms from the NL norm dataset through the encoding system and the results of the sentiment analysis model and inconsistency checker proposed as solutions to further improve the encoding system. Section 6.1 presents the results of encoding the 1001 natural language norms from the NL norm dataset into FOL, utilizing the methods outlined in Section 5.2. Section 6.2 details the results of the sentiment analysis model described in Section 5.3. Section 6.3 presents the result of using the inconsistency checker described in Section 5.4 on the valid Horn encodings from the NL norm dataset. Lastly, Section 6.4 describes the result of the evaluation of faithfulness of the encoding created by the encoding system[1].

6.1 Encoding System

The 1001 natural language sentences from the NL norm dataset were processed by the encoding system described in Section 5.2. The results were stored in a tsv file, containing the 1001 NL norms, their corresponding FOL encodings, and the syntactic assessments of these encodings. Table 6.1 illustrates the content of the tsv file using two example instances. We explain each column in the tsv file.

The column called "num" refers to the index number that the norm had in the Commonsense Norm Bank file. The "input_sequence" column contains the natural language norm. The "fol-translation" column contains the output from prompting the GPT-4 model to encode the norm from "input_sequence" to FOL using the initial prompt alone (see Listing 5.7). The "fol-eval" column contains the syntactic evaluation of the FOL encoding in "fol-translation" using the FOL syntax verifier described in Section 5.2.2. The value in this column is either 1, representing a syntactically valid FOL, or 0, representing a syntactically invalid FOL. The "fol_adjustment-translation" column contains the output

[1] The files containing the encoded norms with their faithfulness evaluation and the Clingo script containing valid Horn clauses converted to ASP can be found in the "results" folder of the GitHub repository containing related code (`https://github.com/emmabjor/nl-horn-master/tree/main`)

from prompting the GPT-4 model to adjust the result of the "fol-translation", using the adjustment prompt (see Listing 5.8).

The "fol_adjustment-eval" column contains the syntax evaluation of the adjusted FOL encoding from "fol_adjustment-translation" using the FOL syntax verifier. The "cnf" column contains the FOL from "fol_adjustment-translation" converted to CNF. The "cnf-eval" column contains the syntax evaluation of the cnf-converted FOL using the FOL syntax verifier. FOL. The "horn" column contains the Horn-verified CNF from the "cnf" column. If the CNF is not considered a valid Horn, this field will contain the string "INVALID HORN", and the last column, "horn-eval", will contain the number 0. Otherwise, the "horn" column contains the CNF formula using the original operator symbols, and the "horn-eval" column contains the number 1.

Of the 1001 FOL encodings returned from the initial prompt call, 967 were evaluated as syntactically valid, which means that they are considered well-formed formulas in the FOL language. After adjusting all 1001 initial FOL encodings using the adjustment prompt, 960 encodings were evaluated as syntactically valid. The remaining 41 adjusted FOL encodings were evaluated as syntactically invalid and were not processed further in the encoding system. Of the 960 syntactically valid FOL encodings, 816 encodings were considered syntactically valid Horn clauses. The remaining 152 FOL encodings were structured in a way that is not considered valid Horn.

6.2 Sentiment Analysis

As described in Section 5.3, while designing the prompts to be used in the FOL encoding part of the encoding system, we noticed that the GPT-4 model had a tendency to create correct norm predicates using incorrect evaluation constants. That means that the model used the evaluation constant "GOOD" if the norm part of the sentence can be commonly considered to be positive, even though the sentence expresses it to be negative, or vice versa. By manually examining the 1001 automatically encoded FOL formulas, we counted 75 encoding instances in which this phenomenon occurred.

In Section 5.3 we describe how we labeled the norms from the NL norm dataset with their respective "GOOD" and "BAD" labels, to be used as a baseline of correct labeling for this set. A total of 539 norms were labeled "GOOD" and 461 norms were labeled "BAD". One norm did not receive either label, as it was encoded using a different evaluation label than "GOOD" or "BAD". This instance was removed from the dataset. We used the fine-tuned sentiment analysis model to predict the evaluation constant labels of the norms in the NL norm dataset and compared the results with the correct baseline labels. In addition, we compared the evaluation constants of the GPT-4 model's encodings to the correct baseline labels. The result of these two comparisons can be seen in Table 6.2.

Column	Example instance 1	Example instance 2
num	1	2205
input_sequence	you should ask too much of your family.	you should not always punish people for their wrongdoings.
fol-translation	$\forall x \forall y(FamilyOf(y,x) \wedge AsksTooMuch(x,y) \rightarrow evaluation(GOOD))$	INVALID FOL: $\forall x \forall y(Wrongdoing(y) \wedge Committed(x,y) \wedge \neg\forall time(PunishAt(x,y,time)) \rightarrow evaluation(GOOD))$
fol-eval	1	0
fol_adjustment-translation	$\forall x \forall y(FamilyOf(y,x) \wedge AsksTooMuch(x,y) \rightarrow evaluation(BAD))$	$\forall x \forall y(People(x) \wedge Wrongdoing(y) \wedge Committed(x,y) \wedge \neg AlwaysPunish(x,y) \rightarrow evaluation(GOOD))$
fol_adjustment-eval	1	1
cnf	$evaluation(BAD) \mid \sim FamilyOf(v4,v3) \mid \sim AsksTooMuch(v3,v4)$	$evaluation(GOOD) \mid \sim People(v93) \mid \sim Wrongdoing(v94) \mid \sim Committed(v93,v94) \mid AlwaysPunish(v93,v94)$
cnf-eval	1	1
horn	$(\neg FamilyOf(v4,v3) \vee \neg AsksTooMuch(v3,v4) \vee evaluation(BAD))$	INVALID HORN
horn-eval	1	0

Table 6.1: Content of system encoding results

We see from Table 6.2 that the sentiment analysis model (SA model) correctly labeled 0.99 of the norms labeled “BAD” and 0.98 of the norms labeled “GOOD”. In comparison, the GPT-4 model correctly labeled only 0.83 of the norms labeled “BAD” and 0.92 of the norms labeled “GOOD”. This result shows that the sentiment analysis model does a better job of finding the correct evaluation labels than the GPT-4 model.

model / baseline	**SA model**		**GPT-4 model**	
	BAD	GOOD	BAD	GOOD
BAD	0.99	0.10	0.83	0.17
GOOD	0.20	0.98	0.08	0.92

Table 6.2: Overlap of predicted evaluation constant for sentiment analysis model and GPT-4 model compared to baseline evaluation constant

6.3 Inconsistency Checker

In Section 5.4, we describe how we use an ASP satisfiability solver to check whether or not the set of Horn Clause-represented norms outputed from the encoding system is consistent.

The 816 valid Horn Clause encodings were converted to ASP facts and rules and written to a Clingo script following the method outlined in Section 5.4. When running the script, the solver is unable to find any stable models and deems the program unsatisfiable. This means that there are inconsistencies in the Horn Clauses fed to the ASP program.

When we isolated the clauses causing the program to terminate as unsatisfiable, we found four clauses. These were the clauses encoded from the norms "it is right for a teacher to bully a student" and "it is wrong for a teacher to bully a student" and two individual clauses that contained a dash (-) in the predicate names, leading to unsatisfiable representations. The two norms mentioned have clauses that express conflicting attitudes towards a teacher bullying a student and we therefore expected these norms to cause an inconsistency.

6.4 Evaluation of Faithfulness

In Section 5.5, we describe how we evaluate how well the encoding system captures the interpreted meaning of the original NL norms in their respective FOL encoding. This was done by defining three categories with different levels of faithfulness and then manually classifying each FOL encoding into one of these categories. The three categories being *Wrong* encodings, *Lacking* encodings, and *Accurate* encodings.

After categorizing each of the 1001 encoded FOLs into one of these three faithfulness categories, we ended up with 284 norms classified as *Wrong*, 168 norms classified as *Lacking*, and 508 norms classified as *Accurate*.

Chapter 7

Analysis of Encoding Mistakes

This chapter means to point out common structures or patterns in faulty FOL encodings and invalid Horn clause representations found in the 1001 encoded norms from the NL norm dataset. The motivation behind this is to inherently figure out what types of norms the encoding system is not able to accurately convert to Horn clauses and what mistakes are made in the GPT-encoding process. Faulty FOL representations are representations that are syntactically invalid, meaning that they are not WFF, or that lack a faithful representation of meaning. We analyze them in an effort to recognize what types of mistakes GPT-4 makes when encoding FOL formulas. By figuring out what mistakes are being made, there is a greater chance of being able to mitigate these issues and gain more precise results in the future. Additionally, we use this section to look into what types of norms cannot be encoded to valid Horn clause structure, even though they might have been correctly represented in FOL. This is done to gain insight on what limitations there might be in representing norms as Horn clauses.

We have identified faulty representations of the FOL and Horn clauses on three different occasions:

1. Within the encoding system, we used a syntax verifier (described in Section 5.2.2) to identify FOL encodings that did not follow a well-formed FOL syntax. These encodings and belonging norms were classified as *Syntactically invalid* encodings.

2. In the evaluation of faithfulness of FOL encodings described in Section 5.5, we identify two additional classification categories containing faulty FOL encodings, called *Wrong* encodings and *Lacking* encodings. These categories include FOLs that are well-formed, but not accurate in terms of intended meaning.

3. The fourth and final classification category is found in the last step of the encoding system, where we filter out FOL encodings that do not have a valid Horn structure. These invalid Horn formulas belong to the classification category called *Not Horn*.

In the end, we are left with four faulty classification categories that will be analyzed in this chapter; *Syntactically invalid* encodings, *Wrong* encodings, *Lacking* encodings, and *Not Horn*.

We analyzed the FOL encodings and accompanying norms within the four faulty classification categories in the following way. First, for each encoding in each of the four classification categories, we identified the main general reason for that instance being classified in this particular category. In other words, what attribute exists in the FOL encoding of the norm that led to this instance belonging to this category over any other category? We grouped the instances into several attribute pattern categories for each classification category based on these observations. Some instances have attributes that belong to several attribute patterns and were therefore sorted into multiple pattern categories. Some attribute pattern categories were further divided into subcategories. Second, we went through each of these attribute pattern categories and tried to identify patterns in the NL norms accompanying the FOL encodings. This was done to generalize what types of norms led to specific FOL attributes when encoded by the encoding system. As many of the attribute pattern categories had few instances, it was difficult to find patterns in the NL norms within all attribute pattern categories. It is important to note that the category definitions and categorization of FOLs were done on the basis of the author's intuition alone. There may exist patterns within the FOL mistakes other than the ones mentioned here.

Similar attribute pattern categories and subcategories exist within both the *Wrong* and *Lacking* classification categories. The difference between the FOL encodings classified between the two comes down to how much the mistake impacts the change in meaning between the FOL and the NL norm. A minor change will be classified as *Lacking*, while a major change will be classified as *Wrong*. An overview of all attribute pattern categories and subcategories, including the number of instances in each category, can be found in Table 7.1.

7.1 *Syntactically Invalid* Encodings

As described in Section 5.2.2, FOL encodings that did not meet the criteria required for a well-formed FOL formula were classified as *Syntactically invalid*. For the 41 *Syntactically invalid* FOLs, we identified one recurring attribute pattern, *nested predicate*, in which 34 FOL encodings were classified. For the remaining FOL encodings, we did not find any recurring patterns. We describe what characterize the FOLs and accompanying norms in the nested predicate category below.

Classification category	**Attribute pattern categories and subcategories**
Syntactically invalid encodings (41)	Nested predicate (34)
Wrong encodings (283)	Content errors (85): - Missing part (43) - Actor-action disconnect (36) - Negation excluded/included (5)
	Interpretive errors (72): - Wrong evaluation constant (69) - Over-interpretations (3)
	Format errors (64): - Extra conditional (36) - You constant (9) - Quantifier in predicate (6)
	Variable/constant errors (47): - Mixed variable (27) - Missing variable (17) - Constant excluded/included (3)
	Predicate errors (18): - Split predicate (13) - Extra predicate (5)
	Time property errors (3)
Lacking encodings (168)	Predicate errors (89): - Split/unsplit predicate (28) - Extra predicate (26) - Imprecise predicate naming (20) - Missing predicate (18)
	Variable/constant errors (65): - Missing variable (37) - Mixed variable (13) - Extra variable (9) - Constant included (6)
	Wrong quantifier (20)
	Missing part (6)
Not Horns (152)	Negation phrase in norm part (101) Negation phrase in evaluation part (32) Negatively quantified (9) Inequality (6) Encoding errors (4)

Table 7.1: Overview of encoding error categories

7.1.1 Nested predicate

We call structures on the form $F(G)$, where F and G are both atomic formulas, for a nested predicate. That is, when one atomic formula is defined within the scope of another atomic formula. FOLs with this structure are not WFF. 34 of the 41 *Syntactically invalid* FOLs contained some variation of this nested predicate structure. An example from this category is presented in Example 7.1.1.

We find that 25 of the 34 nested predicate FOL encodings have a NL norm structure similar to that in Example 7.1.1. That is, norms in which someone performs an act over someone or something else. This includes phrases like “x tells y something”, “x thinks y is something”, or “x wants y to do something”, which tended to be encoded respectively

Norm: "you shouldn't be understanding when your spouse thinks you're up to something"
FOL encoding: $\forall x \forall y (SpouseOf(x, y) \wedge Thinks(y, UpToSomething(x)) \wedge Understanding(x) \rightarrow evaluation(BAD))$

We see in this FOL that the predicate *UpToSomething* exists within the scope of the predicate *Thinks*.

Example 7.1.1: Nested predicate

as *Tells(x, Something(y))*, *Think(x, Something(y))* and *Wants(x, DoSomething(y))*

7.2 *Wrong* Encodings

We classify FOL encodings that are WFF but cannot be said to express the same interpreted meaning as their NL norm as *Wrong*. For the 283 *Wrong* FOLs, we identified six recurring attribute pattern categories. The attribute pattern categories and the numbers of instances in each category are: *content errors* (85), *interpretive errors* (72), *format errors* (64), *variable/constant errors* (47), *predicate errors* (18) and *time property errors* (3). For five of the six attribute patterns, we identified several subcategories. Some FOL encodings were classified into several main- or subcategories. We describe what characterizes the FOLs and norms in each of the six categories and their subcategories below.

7.2.1 Content errors

We found 85 FOL encodings that displayed content errors. The FOLs in this category are WFF with the expected format and structure, but the content of the FOL makes the intended meaning wrong when compared to the original norm. We identified three subcategories within this content error category: *missing part* (43), *actor-action disconnect* (36) and *negation excluded/included* (5). We describe the three subcategories below.

Missing part errors occur when a vital part of the norm is left out of the FOL encoding. The consequence of not representing this part in the FOL encoding is that the interpreted meaning of the FOL becomes significantly different from the meaning of the NL norm. We found 43 encodings that fall into this subcategory. Two examples are presented in Example 7.2.1.

We found two recurring patterns in the norms that belong to the FOL encodings within this subcategory. The first is that six of the 43 norms contained some form of additional information to the actual norm. In the norm, "you shouldn't always be careful

Norm: "you shouldn't always be careful when swimming in case a wave hits you"
FOL encoding: $\forall x \forall y(People(x) \wedge Swimming(x) \wedge AlwaysBeCareful(x) \rightarrow evaluation(BAD))$

In this example, we see that the "in case a wave hits you" part of the norm is not included in the FOL encoding.

Norm: 'you have to be faithful to a partner if you don't feel like it"
FOL encoding: $\forall x \forall y(PartnerOf(y, x) \wedge \neg FeelsLikeBeingFaithful(x, y) \rightarrow evaluation(BAD))$

In this example, we see that the "have to be faithful" part of the norm is not included in the FOL encoding.

Example 7.2.1: Missing part error

when swimming in case a wave hits you" from Example 7.2.1, the part that was excluded in the FOL encoding, "in case a wave hits you" is more of a justification of the actual norm, "you shouldn't always be careful when swimming". The second norm structure that we found to be repeated within this subcategory were norms that express both intent toward and execution of an action. For example, for the norm "you have to be faithful to a partner if you don't feel like it" from Example 7.2.1, an FOL encoding would ideally represent both the intent of being faithful to a partner and the execution of being faithful to a partner. However, in the actual encoding, only one of these dimensions is included. There were four norms with this type of structure that fell into this particular attribute pattern subcategory.

Actor-action disconnect errors occur when an action requires an actor, but the actor is either not mentioned, or is mentioned but not connected to the action. This means that the action and actor are represented independently of each other, and we lose the important aspect of someone actually performing the action. We found 36 FOL encodings that fall into this subcategory. Two norm examples from this category are presented in Example 7.2.2.

Apart from the fact that all the norms that fall under this attribute pattern subcategory have in some way an actor and action, we found that 15 of the norms included ditransitive verbs. Ditransitive verbs are verbs that take two objects. In the sentence "it's bad to feed wild animals domesticated food" from Example 7.2.2, the verb "feed" is a ditransitive verb, as it takes both "wild animals" and "domesticated food" as objects. In these 15 cases, the ditransitive verbs were the ones that lacked the actor-action connection in the FOL encodings.

Norm: "it's bad to feed wild animals domesticated food"
FOL encoding: $\forall x \forall y(WildAnimal(x) \wedge DomesticatedFood(y) \wedge Feed(x, y) \rightarrow evaluation(BAD))$

This encoding lacks an actor to do the leaving. A more precise encoding would be to change the predicate *Feed(x,y)* to $\forall x(Feed(x, y, z))$.

Norm: "it's okay to regret leaving someone"
Encoding: $\forall x \forall y(Leaves(x, y) \wedge Regrets(x) \rightarrow evaluation(GOOD))$

In this encoding, the encoding does not connect the regretting to the leaving, only that individual x regrets in general.

Example 7.2.2: Actor-action disconnect error

Negation excluded/included errors occur when a predicate should have been negated in accordance with the norm but is not, or when a predicate that should not be negated in accordance with the norm is negated. The consequence of excluding or including a negation is that the meaning expressed in the FOL encoding directly contradicts the sentiment expressed in the NL norm. Five FOL encodings contained this error. An example from this category is presented in Example 7.2.3.

Norm: "you shouldn't console your loved ones when they are upset"
FOL encoding: $\forall x \forall y(LovedOnes(y, x) \wedge Upset(y) \wedge \neg Consoles(x, y) \rightarrow evaluation(BAD))$

We see that by negating the predicate *Consoles(x,y)*, the meaning in the FOL encoding can be interpreted as "you shouldn't **not** console your loved ones when they are upset", which is the complete opposite of what the original norm expresses.

Example 7.2.3: Negation excluded/included error

Four out of the five norms with FOL encodings that had this negation excluded/included error expressed what we call counter-intuitive norms. That means they express norms that seem to be counter-intuitive to what the most people would think. Examples of these types of norms are listed in Example 5.3.1. We can assume that some over-interpretation from GPT-4 has been present during the encoding of these norms.

7.2.2 Interpretive errors

FOL encodings identified as *Wrong* as a result of GPT-4 performing its own interpretations of the norm fall into the category *interpretive errors*. We identified 72 FOL encodings that contain this attribute. We divide this attribute pattern category into two subcategories: *wrong evaluation constant* (69) and *over-interpretation* (3).

Wrong evaluation constant errors occur when the system encodes a norm to FOL using the opposite evaluation constant to what would be the correct one according to the interpretation of the norm. Either the encoding uses the evaluation constant "GOOD" when it is supposed to be "BAD", or vice versa. An example from this category is presented in Example 7.2.4.

Norm: "it's okay to verbally abuse someone"
FOL encoding: $\forall x \forall y (AbusesVerbally(x, y) \rightarrow evaluation(BAD))$

In this case, we see that the FOL encoding of the norm is syntactically correct and faithfully expresses the same meaning as the norm, with the exception of the evaluation constant. The norm says that it is okay to verbally abuse someone, which means that the evaluation constant should be "GOOD", but instead the FOL encoding uses the evaluation constant "BAD".

Example 7.2.4: Wrong evaluation constant error

We found that 63 of the 69 FOL encodings classified in the wrong evaluation constant category were counter-intuitive norms.

Over-interpretation occurs when the FOL encoding is based on an overly free interpretation of the norm, rather than strictly sticking to the explicit content expressed in the norm. Three FOL encodings were classified with this attribute. We can in some of the cases argue that the intended meaning expressed in the norm is not changed in the over-interpreted FOL encoding. However, as the system has been instructed to create encodings that convey the explicit meaning of the norm (see Listings 5.7 and 5.8), interpretations that go beyond this were classified as an interpretive mistake. An example from this category is presented in Example 7.2.5.

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Norm: "people should leave online slang online"
FOL encoding: $\forall x \forall y (People(x) \land OnlineSlang(y) \land UsesOutsideInternet(x, y) \rightarrow evaluation(BAD))$

This FOL does not express the exact same meaning as the original sentence, although we can argue that it expresses a consequence of the original sentence.

Example 7.2.5: Over-interpretation

7.2.3 Format errors

FOL encodings are classified as containing format errors when the FOL is WFF but does not follow the format specified in the prompt. We found 64 norms that fall into this attribute pattern category. These 64 norms can be further divided into three different subcategories, these being *extra conditional* (36), *you constant* (9), and *quantifier in predicate* (6). These subcategories are described in detail below.

Extra conditional errors occur in all FOL encodings consisting of a nested implication structure. This means that there exists an implication within the scope of another implication, on the form $p \rightarrow q \rightarrow r$, where p, q and r are atomic formulas. This structure conflicts with the instruction prompt used in the system, which says that each FOL encoding should consist of a single implication (see Listings 5.7 and 5.8). An example from this category is presented in Example 7.2.6.

Norm: "you're expected to make sure food you serve to others is safe to eat"
FOL encoding: $\forall x \forall y \forall z (ServesTo(x, y, z) \land Food(z) \rightarrow SafeToEat(z) \rightarrow evaluation(GOOD))$

We see that the norm part of the FOL encoding is an implication of its own.

Example 7.2.6: Extra conditional error

The most apparent pattern within the norms that have FOL encodings that fall into this attribute pattern subcategory is that many of the norms are structured on an if-then form. That means that the norms express an implication within their norm part using the words "if" or "when". For example "**if** you don't like a particular subreddit, you should visit it" or "**when** you leave food out in the open, you shouldn't be surprised when animals get into it." We found that 17 out of the 36 norms that fall into this subcategory has this particular if-then form.

You constant errors occur when one or several predicates in the FOL encoding contain a specific constant, "YOU". This error is present in nine FOL encodings. We identify this as a format error because the few-shot learning in the prompts (see Listings 5.7 and 5.8) clearly show that norms expressing "you" should be encoded using quantified variables instead of a constant. An example from this category is presented in Example 7.2.7.

Norm: "it's not okay to let something little take over your thoughts"
FOL encoding: $\forall x \forall y(Little(x) \land Thought(y) \land BelongsTo(y, YOU) \land TakeOver(x, y) \rightarrow evaluation(BAD))$

The actor in the norm is represented by the constant "YOU", instead of as a universally quantified variable.

Example 7.2.7: You constant error

Seven out of the nine norms that have encodings in this category use the word "you" explicitly in the sentence.

Quantifier in predicate errors are FOL encodings where the scope of a variable is expressed though the quantifiers, as well as in the predicate names. In these cases, the correct encodings should only rely on using correct quantifiers to express this dimension of the norm. However, in six cases, this dimension was also included in the predicates. An example from this category is presented in Example 7.2.8.

Norm: 'you should always pay off your student loans"
FOL encoding: $\forall x \forall y(BelongsTo(x, y) \land StudentLoan(x) \land AlwaysPaysOff(y, x) \rightarrow evaluation(GOOD))$

In the FOL encoding, the word "always" is unneccesarily included in the predicate *AlwaysPaysOff*.

Example 7.2.8: Quantifier in predicate error

In all six cases where this mistake is apparent, the norm expresses "always" or "nothing" explicitly in the norm part.

7.2.4 Variable/constant errors

We identified 47 FOL encodings that contain variable/constant errors, which means the FOLs are WFF, have the desired structure and format, and all necessary predicates. However, some errors have been made with regard to the variables or constants within the predicates that have led to an incorrect FOL representation of the original NL norm statement. We identified three subcategories within this attribute pattern category: *mixed variable* (27), *missing variables* (17) and *constant excluded/included* (3). These subcategories are described in detail below.

Mixed variable errors occur when the necessary variables are present, but the placement of variables within the predicates is not as we would expect. This means that either, the variables within one predicate have swapped positions, that the wrong variables are used, or that extra variables are included. We found 27 FOL encodings that belong in this subcategory. An example from this category is presented in Example 7.2.9.

Norm: "you shouldn't balance relationships and friendships"
FOL encoding: $\forall x \forall y \forall z(Relationship(x, y) \wedge Friends(y, z) \wedge Balance(x, z) \rightarrow evaluation(BAD))$

We would expect the relationship and friends relations to be represented as one-place predicate and the balance relation to be represented as a three-place predicate containing the actor, the relationship variable and the friends variable. By representing the relationship and friends relations as two-place predicates, the variables used in the balance predicate relate to the people in these relations rather than to the relations themselves.

Example 7.2.9: Mixed variable error

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Missing variable errors occur when a predicate in the FOL encoding is missing a variable that is essential to convey the sentiment of the norm. The attributes characterizing the FOLs in this subcategory are similar to the attributes in the actor-action disconnect subcategory described in Section 7.2.1, as some actor-action disconnect errors are due to missing variables. However, actor-action disconnect errors apply strictly to the instances where these missing variables create a disconnect between an action and the one performing the action. All other instances of missing variables are classified into this missing variables subcategory. We found in total 17 FOL encodings belonging to this subcategory. An example is presented in Example 7.2.10.

Norm: "it's fine to want good things for yourself"
FOL encoding: $\forall x \forall y(GoodThings(y) \wedge WantsFor(x,y) \rightarrow evaluation(GOOD))$

The encoding lacks a variable in the predicate *WantsFor(x,y)*. As the individual, x, wants the good thing, y, for the individual, x, the precise encoding should be *WantsFor(x,y,x)*.

Example 7.2.10: Missing variable error

In five of the 17 norms that had FOL encodings belonging to the missing variables subcategory, the missing variable should have represented a reflexive pronoun. Reflexive pronouns are pronouns ending in -selv or -selves, where the individual acts as both the subject and the object of the sentence. In the norm from Example 7.2.10, "it's fine to want good things for yourself", the individual who acts as the subject of the sentence, the actor wanting good things for someone, also acts as the object of the sentence, which is the person for whom they want good things for.

Constant excluded/included errors occur when the FOL encoding use a constant to represent entities that we would expect to be represented using variables or predicates, or vice versa. We found three FOL encodings that contain this particular mistake. An example from this category is presented in Example 7.2.11.

Norm: "it's okay to prefer certain brand"
FOL encoding: $\forall x \forall y(Brand(y) \wedge Prefers(x,y) \rightarrow evaluation(GOOD))$

We would expect the phrase "a certain brand" to be represented as a constant, as it describes a very specific instance.

Example 7.2.11: Constant excluded/included error

We found no distinct patterns in the NL norms that had encodings within this subcategory.

7.2.5 Predicate errors

We identified 18 FOL encodings that contain predicate errors, which means that the FOLs are WFF, have the desired structure and format, but errors have been made with respect to the predicates used, leading to an incorrect representation of the original NL norm statement. We identified two subcategories within this attribute pattern category: *split predicate* (13) and *extra predicate* (5). Both subcategories are described in detail below.

Split predicate errors are errors where a part of the norm that should ideally be expressed in FOL as a single predicate has been split up into multiple separate predicates in the FOL encoding. The consequence of this is that we lose a vital connection between dependent parts of a sentence. We found 13 FOL encodings with this attribute. An example is presented in Example 7.2.12.

Norm: "it is immature to spend your time watching cartoons"
FOL encoding: $\forall x \forall y (Time(x) \land Cartoons(y) \land SpendTime(x, y) \land Watch(x) \rightarrow evaluation(BAD))$

In this case, we see that the predicates with their appropriate variables *SpendTime(x,y)* and *Watch(x)* would more accurately represent the phrase "x spend time watching y" if they were combined into a single predicate *SpendTimeWatching(x,y)*.

Example 7.2.12: Split predicate error

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Extra predicate errors can be defined as FOL encodings that express dimensions not mentioned in the NL norm. We found five FOL encodings with extra predicate error leading to a major change in the intended meaning of the FOL encoding compared to the original NL norm. An example from this category is presented in Example 7.2.13.

Norm: "it's unwholesome to seek out young girls for sexual gratification"
FOL encoding: $\forall x \forall y \forall z (Man(x) \land Girl(y) \land Young(y) \land SeeksFor(x, y, SEXUAL\_GRATIFICATION) \rightarrow evaluation(BAD))$

The FOL expresses that the individual x seeking sexual gratification also has the property *Man*. However, this is not expressed in the NL norm. This particular extra predicate mistake is likely due to bias in the GPT-4 training data.

Example 7.2.13: Extra predicate error

We found no distinct patterns in the NL norms that had encodings within this subcategory.

7.2.6 Time property errors

Three norms were encoded incorrectly as a consequence of the way the FOL encodings expressed time dimensions using temporal phrases. Temporal phrases provide a temporal context for the situation, such as "after a while", or "from a while ago". An example from this category is presented in Example 7.2.14.

Norm: "it's irrational to get tired of guests after a while"
FOL encoding: $\forall x \forall y(Guest(y) \wedge AfterAWhile(x, y) \wedge TiredOf(x, y) \rightarrow evaluation(BAD))$

We see here that the encoding becomes wrong in terms of meaning because the temporal phrase "after a while" is represented as an isolated predicate that is not connected in any way to the situation.

Example 7.2.14: Time property error

The only pattern in the norms accompanying the FOL encodings in this attribute pattern category is that they all express temporal dimensions in one way or another.

7.3 *Lacking* Encodings

We classify FOL encodings that are WFF and express a similar interpreted meaning as its NL norm as *Lacking*. Compared to FOL encodings classified as *Wrong*, the FOL encodings classified as *Lacking* can be argued to express a much more similar interpreted meaning as the original norm. For the 168 FOL encodings that were classified as *Lacking* we identified four attribute pattern categories: *predicate errors* (89), *variable/constant errors* (65), *wrong quantifier* (20) and *missing part* (6). For two of the four main attribute pattern categories, we identified several subcategories. Some FOL encodings were categorized into several main- or subcategories. We describe what characterize the FOLs and norms in each of the four main categories and their subcategories below.

7.3.1 Predicate errors

We found 89 FOL encodings that have some sort of predicate error. This means that the FOLs are WFFs, have the desired structure and format, but mistakes have been made in regard to the predicates that are being used, which leads to a suboptimal FOL representation of the original NL norm statement. We identified four subcategories within this attribute pattern category: *split/unsplit predicate* (28), *extra predicate* (26), *imprecise predicate naming* (20), and *missing predicate* (18). Each subcategory is described in detail below.

Split/unsplit predicate errors occur when a representation that should ideally be captured by a single predicate is split into multiple predicates. Or, vice versa, when a representation that should be expressed through multiple predicates in order to capture specific details is merged into a single predicate. We found 28 FOL encodings that belong in this split/unsplit predicate subcategory. Two examples are presented in Example 7.3.1.

Norm: "it's bad to be very smart"
FOL encoding: $\forall x(Smart(x) \wedge Very(x) \rightarrow evaluation(BAD))$

In this example, the phrase "very smart" should be merged into one predicate, *VerySmart* to make the phrase make sense, as something cannot simply have the quality "very" without another attribute associated with it.

Norm: "it's not a good thing to eat fruit when you're craving something sweet"
FOL encoding: $\forall x \forall y(Fruit(x) \wedge CravingSweet(y) \wedge EatWhen(y, x) \wedge evaluation(BAD))$

In this example, the FOL would be more generalizable if the predicate *CravingSweet* had been further broken down into two predicates *Sweet* and *Craving*.

Example 7.3.1: Split/unsplit predicate error

The only recurring pattern found within the norms that had FOL encodings belonging to this category was that four of the norms use some kind of temporal preposition, such as "after", "before", or "ahead". We see that these prepositions are often treated as individual predicates, when we would ideally like for them to be merged in with another predicate. For example, in the sentence "relax after a hard day", the preposition "after" should be included in the predicate that expresses "relax" rather than as its own predicate, creating the joint predicate *RelaxAfter*.

Extra predicate errors occur when the FOL encodings include a predicate that is unnecessary to capture the interpreted meaning of the NL norm. We found 26 FOL encodings that were categorized within this extra predicate subcategory. An example is presented in Example 7.3.2.

Norm: "you shouldn't be a bad influence on kids"
FOL encoding: $\forall x \forall y(Person(x) \land Kids(y) \land BadInfluenceOn(x, y) \rightarrow evaluation(BAD))$

The norm does not specify that the actor in this scenario is a person, but this attribute is nonetheless included in the FOL. Note that within the context it is safe to assume that these norms mainly concern people as actors, and therefore it is not deemed a crucial error but rather imprecise in terms of interpreted meaning.

Example 7.3.2: Extra predicate error

There are 20 FOL encodings within this category that make the same mistake as in Example 7.3.2, where the encoding adds an extra *Person* or *People* predicate without this being specified in the norm. We found no distinct patterns in the NL norms that are encoded with this additional *Person* predicate when compared to the norms that are encoded without the addition *Person* predicate.

Imprecise predicate naming occurs when a predicates name does not fully capture the phenomenon it is supposed to represent. As predicate names do not necessarily affect the logical representation of a sentence, they are not categorized as crucial errors. However, the naming of predicates is important to those who use the logic, and imprecise predicate naming could have logical consequences if two different phenomena share the same predicate name, or two identical phenomena use different predicate names. An example from this category is presented in Example 7.3.3.

Norm: "you should physically fight"
FOL encoding: $\forall x(Fight(x) \rightarrow evaluation(GOOD))$

The structure of the FOL and the variables included are as we would expect, but the predicate does not specify the physical dimension in the *Fight* predicate. A more accurate predicate name would be *PhysicallyFight*.

Example 7.3.3: Imprecise predicate naming

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Missing predicate errors occur when an FOL encoding is missing a dimension expressed in the NL norm that could be solved by adding a single predicate to the FOL encoding. There are 18 of these missing predicate mistakes in the FOL encodings. An example from this category is presented in Example 7.3.4.

Norm: "it's bad if you have romantic feelings for your doctor"
FOL encoding: $\forall x \forall y(Doctor(y) \land HasRomanticFeelings(x, y) \rightarrow evaluation(BAD))$

This encoding can be interpreted to mean that it is bad if you have romantic feelings for any doctor. The FOL misses some kind of relational predicate, such as *BelongsTo(y,x)*, to ensure that the doctor is yours.

Example 7.3.4: Missing predicate error

We found that seven of the 18 FOL encodings in this subcategory lack a relation predicate that expresses possession, such as in Example 7.3.4. Possessive pronouns are pronouns that express someone's possession of something else, such as "your", "his", "someone's", etc. The inclusion of a relation predicate in cases where possessive pronouns are used has been instructed in the prompt system message (see Listings 5.7 and 5.8).

7.3.2 Variable/constant errors

We found 65 FOL encodings that contain a variable/constant error. FOLs within this category are WFFs and have the desired structure and format and all the necessary predicates. However, some mistake has been made with regard to the variables or constants within the predicates, which has led to an imprecise representation of the original NL norm statement. We identified four subcategories within this attribute pattern category: *missing variable* (37), *mixed variable* (13), *extra variable* (9), and *constant included* (6). Each subcategory is explained in detail below.

Missing variable errors occur when a predicate in the FOL encoding is missing a variable that should preferably have been there. We found 37 FOL encodings classified within this subcategory. An example is presented in Example 7.3.5.

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Mixed variable errors occur in the FOLs where all the necessary variables are included, but the placement of the variables within the predicates could lead to a misinterpretation in terms of meaning when compared to the original norm. We found 13 FOL encodings classified within this category. An example is presented in Example 7.3.6.

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Norm: "it is comforting to ease your friends paranoia"
FOL encoding: $\forall x \forall y(Friends(x, y) \land Paranoia(y) \land Eases(x, y) \rightarrow evaluation(GOOD))$

In the encoding, "paranoia" is not represented with its own variable, and therefore the meaning of the FOL encoding is most likely interpreted as "it is comforting to ease your friend who has paranoia" (as long as we interpret the predicate *Paranoia(y)* to mean "individual y has paranoia"), which is not explicitly equivalent to what the original norm expresses.

Example 7.3.5: Missing variable error

Norm: "if you love your job, you should quit it"
FOL encoding: $\forall x \forall y(JOB(y) \land BelongsTo(x, y) \land Loves(x, y) \land Quits(x, y) \rightarrow evaluation(GOOD))$

In this encoding, we see that the order of the variables within the predicate *BelongsTo(x,y)* would arguably be more correct if they swapped positions. In the FOL encoding, we can interpret the FOL to mean that the actor x belongs to the job y. It can be argued that this is a valid way of representing someone's job. However, since the norm states "your job" using a possessive pronoun to represent the actor, we would prefer the order of the variables within the *BelongsTo(x,y)* predicate to be the other way around, *BelongsTo(y,x)*, expressing that the job, y, belongs to the actor, x.

Example 7.3.6: Mixed variable error

Extra variable errors occur when a superfluous variable is added to a predicate. That is, a variable that does not necessarily contain any information. Nine FOL encodings were identified as having this particular attribute. An example from this category is presented in Example 7.3.7.

We found no distinct patterns in the NL norms that had encodings within this subcategory.

Constant included errors occur when the FOL encoding uses a constant in a setting where it would be more natural to use a variable or predicate. The use of a constant instead of a variable or predicate may affect the generalizability and preciseness of the FOLs. Six FOL encodings belong to this subcategory. An example is presented in Example 7.3.8.

We found no distinct patterns in the NL norms that had encodings within this category.

Norm: "it's understandable if blind dates don't go well"
FOL encoding: $\forall x \forall y(BlindDate(x,y) \wedge \neg GoesWell(x,y) \rightarrow evaluation(GOOD))$

In the NL norm, a blind date is described as a single object and should therefore be represented using a single variable. It could be argued that the predicate *BlindDate* should naturally take two objects, namely the two individuals on the date. However, the norm does not specify the blind date to be between two people, so we do not assume that information here. Furthermore, the FOL encoding would be imprecise since the predicate *GoesWell(x,y)* would then not describe the date itself as going well, but rather the two individuals going well, which would ultimately be an imprecise encoding of the norm.

Example 7.3.7: Extra variable error

Norm: "it's bad for a college student to live at home"
FOL encoding: $\forall x \forall y(Student(x, COLLEGE) \wedge Home(y) \wedge LivesAt(x,y) \rightarrow evaluation(BAD))$

It would be preferable to represent "college" as a predicate, as a college is more naturally thought of as a general category than a specific individual, since multiple instances can be defined as a college.

Example 7.3.8: Constant included error

7.3.3 Wrong quantifier

We found 20 FOL encodings that use one or more wrong quantifiers. This means that the FOL uses an existential quantifier over a variable that should be universally quantified, or vice versa. We identify this category as *Lacking* rather than *Wrong* because, although the use of wrong quantifier changes the specification of who is included in the variables, it is predictable and a small mistake that can be argued to not affect the core norm part of the encoding. An example from this category is presented in Example 7.3.9.

Norm: "people should date others based solely on their age"
FOL encoding: $\forall x \forall y \forall z(People(x) \wedge People(z) \wedge AgeOf(y,z) \wedge DateBasedOn(x,z,y) \rightarrow evaluation(GOOD))$

The variable representing "others" is universally quantified in this FOL encoding. As the phrase "others" refers to a specific set of people, or, in other words, all people except you, this variable should be represented existentially. This has been explicitly instructed in the prompt system message (see Listings 5.7 and 5.8).

Example 7.3.9: Wrong quantifier

We found that 12 of the 20 norms with FOL encodings belonging to this category referred in some way to "others", "someone else", or "somewhere else". In other words, phrases that refer to anyone or anywhere except for you or here. These are phrases that should be represented using the existential quantifier, $\exists$, but in these cases are represented using the universal quantifier, $\forall$.

7.3.4 Missing part

Six FOL encodings were identified as containing a missing part error, which is when a less vital part of the NL norms is left out of the FOL encoding. That means that each norm within this category could be rewritten in a way where the part left out in the FOL encoding was not included in the norm, but the norm would still mean more or less the same. An example from this category is presented in Example 7.3.10.

Norm: "one should be honest with people about their past whether it's good or bad"
FOL encoding: $\forall x \forall y (People(y) \land Past(x) \land HonestAbout(x, y) \rightarrow evaluation(GOOD))$

The phrase "whether it's good or bad" from the norm is excluded from the FOL. We note that the exclusion of this phrase does not change the intent of the norm, as the phrase expresses a tautology.

Example 7.3.10: Missing part

Five out of six norms that had FOL encodings within this category included additional information that arguably is redundant to the actual norm part of the sentence. These additional parts are either a further specification of the situation or a reason for acting according to the norm.

7.4 *Not Horn*

We classify valid FOL encodings that do not meet the first-order Horn criteria specified in Section 2.2.4 as *Not Horn*. Unlike the attribute pattern categories described for the classification categories *Semantically invalid* encodings, *Wrong* encodings and *Lacking* encodings, all FOL encodings within the *Not Horn* classification category are classified as such for the same reason. This reason being that, when restructured into CNF, each clause contains more than one positive atomic formula. This means that we will not identify the characteristics of the FOL encodings, as they are all the same, but rather we will focus only on the characteristics found in the norms belonging to the FOL encodings classified as *Not Horn*. We analyze the *Not Horn* FOL encodings from all syntactically valid categories, *Wrong*, *Lacking* and *Accurate* to cover as many instances as possible. For the 152 norms belonging to the FOL encodings that were classified as *Not Horn*, we identified five norm attribute patterns: *negation phrase in norm part* (101), *negation phrase in evaluation part* (32), *negatively quantified* (9), *inequality* (6), and *encoding errors* (4). We describe what characterizes the norms in each of the five categories below.

7.4.1 Negation phrase in norm part

We found that 101 of the *Not Horn* norms contain some sort of explicit or implicit negation phrase in the norm part of the sentence. This means that the word “not”, or any equivalent or related expressions, is mentioned in the norm part. Explicit mentions include the word “not”, as well as abbreviations of the word, such as “won’t”, “can’t” and “isn’t”. Implicit mentions include “not”-equivalent words, such as “never” and “without”, or “not”-adjacent words, such as “no longer”, “doubt”, “stop”, “avoid” and so on. All explicit and implicit negation phrases are listed in Definition 1. An example of a norm using explicit negation in the norm part is presented in Example 7.4.1, and an example of a norm using implicit negation in the norm part is presented in Example 7.4.2.

Definition 1. *Negation phrases:* not, won’t, can’t, isn’t, never, stop, without, no longer, doubt, avoid, nothing, no one, not always, not everyone.

Norm: “it’s expected that you won’t cut off family members”
FOL encoding: $\forall x \forall y(FamilyOf(x,y) \land \neg CutsOff(x,y) \rightarrow evaluation(GOOD))$
CNF notation: $evaluation(GOOD) \lor \neg FamilyOf(x,y) \lor CutsOff(x,y)$

In the FOL encoding, we clearly see how the “won’t cut off” part of the norm has resulted in a negated predicate, $\neg CutOff$, in the antecedent of the FOL encoding. When converting an implication to CNF, each predicate in the antecedent is negated and together with the consequence form a conjunction of disjunctions. We see that the CNF converted FOL encoding of the norm contains more than one positive atomic formula, *evaluation(GOOD)* and *CutsOff(x,y)*, and is therefore classified as *Not Horn*. The *Not Horn* classification is a direct cause of the FOL implication that has a positive consequence and one or more negated predicates in the antecedent. As all FOL encodings in this system should be encoded with a positive consequence, as instructed in the prompt system message (see Listings 5.7 and 5.8), it is the use of the word “won’t” in the norm part of the norm, leading to a negated predicate in the norm part of the encoding, that ultimately causes the FOL encoding to become *Not Horn*.

Example 7.4.1: Explicit negation phrase in norm part

The encoding patterns shown in Examples 7.4.1 and 7.4.2, where a negated atomic formula in the norm part occurs as a consequence of a negation phrase in the norm part of

Norm: "it's wrong to take pictures of people without their consent"
FOL encoding: $\forall x \forall y \forall z (Picture(x) \wedge People(y) \wedge Take(z, x, y) \wedge \neg HasConsent(y, z) \rightarrow evaluation(BAD))$
CNF notation: $evaluation(BAD) \vee \neg Picture(x) \vee \neg People(y) \vee \neg Take(z, x, y) \vee HasConsent(y, z)$

The positive predicate *HasConsent* in the CNF notation is a direct consequence of the use of the phrase "without their consent" in the norm, and together with the positive evaluation part, this ensures that the FOL encoding is *Not Horn*.

Example 7.4.2: Implicit negation phrase in norm part

the norm, are consistent for all 101 norms classified as *Not Horn* that use some variation of a negation phrase in the norm part.

7.4.2 Negation phrase in evaluation part

We found that 32 norms were classified as *Not Horn* as a consequence of an expressed negation in the evaluation part, such as "should not", or "not expected", being incorporated into the norm part of the FOL. This resulted in two positive predicates when converted to CNF, which consequently led to the formula being *Not Horn*. An example of a norm from this category is presented in Example 7.4.3.

Norm: "people are not expected to take care of their loved ones"
FOL encoding: $\forall x \forall y (Loves(x, y) \wedge \neg TakesCareOf(y, x) \rightarrow evaluation(GOOD))$
CNF notation: $evaluation(GOOD) \vee \neg Loves(x, y) \vee TakesCareOf(y, x)$

In the FOL we see that the negation from the evaluation part, "not expected", is attached to the predicate *TakesCareOf* rather than being encoded to *evaluation(BAD)*. The consequence of this is that we get two positive predicates in the CNF converted FOL and the formula is therefore considered *Not Horn*. We can argue that the result of this encoding is imprecise, since the sentence "people are not expected to take care of their loved ones" does not necessarily mean the same as "people are expected to not take care of their loved ones", which is what we would interpret the FOL encoding to express.

Example 7.4.3: Negation phrase in evaluation part

It is clearly stated in the prompt instruction in the encoding system (see Listings 5.7 and 5.8) that negatively stated evaluations, such as "not expected", should be encoded to *evaluation(BAD)*. By including the negation in the norm part instead of in the evaluation part, the encoding disregards the prompt instructions and the intended meaning of the resulting FOL encoding changes. We find this to be a fault in the encoding system rather than in the Horn clause format. All 32 norms and their respective FOL encodings within this category adhere to the same pattern as in Example 7.4.3.

7.4.3 Negatively quantified

For nine norms, the crucial attribute that led to their FOL encoding being *Not Horn*, was the negation of quantifiers. The negation of quantifiers in the FOL encodings is a direct result of the norms containing the phrases "no one", "not always", "not everyone", or "nothing". An example from this category is presented in Example 7.4.4.

Norm: "you should stay with someone you have nothing in common with"
FOL encoding: $\forall x \forall y (StaysWith(x, y) \ \wedge \ \neg\exists z(Common(x, y, z)) \quad \rightarrow evaluation(GOOD))$
CNF notation: $evaluation(GOOD) \vee \neg StaysWith(x, y) \vee Common(x, y, f0(x, y))$

We see here that the phrase "nothing in common with" is encoded to $\neg\exists z(Common(x, y, z)$. In the CNF conversion process described in Chapter 4, negated existential quantifiers are replaced by universal quantifiers, and the negation is moved inward. This leads to the predicate *Common* being negated, and we get two positive predicates in the CNF converted formula, causing the formula to be *Not Horn*.

Example 7.4.4: Negatively quantified

A similar structure to the norm presented in Example 7.4.4 is evident with all *Not Horn*-classified norms containing the phrases "no one", "not always", "not everyone" or "nothing" in the norm part.

7.4.4 Inequality

Six norms within the *Not Horn* category were encoded using some version of a negated predicate, $\neg Equal(x, y)$, supposed to convey that y is "other people" or "someone else" compared to x. It is the use of this predicate that ultimately leads to the FOL encoding being *Not Horn*. An example from this category is presented in Example 7.4.5.

The use of an *Equality* predicate is either superfluous or not preferred when encoding norms containing the phrases "other people" or "someone else". The prompt instruction (see Listings 5.7 and 5.8) clearly states that these phrases should be encoded using existential quantifiers instead of inequality predicates. We find this to be a fault in the encoding system rather than in the Horn clause format.

Norm: "you shouldn't call other people besides your partner"
FOL encoding: $\forall x \forall y \exists z(PartnerOf(x,y) \wedge People(z) \wedge \neg SamePerson(y,z) \wedge Calls(x,z) \rightarrow evaluation(BAD))$
CNF notation: $evaluation(BAD) \vee \neg PartnerOf(x,y) \vee \neg People(f1(x,y)) \vee SamePerson(y,f1(x,y)) \vee \neg Calls(x,f1(x,y))$

In this example, we see the introduction of the negated predicate $SamePerson$ used to emphasize that the individual "partner", x, is different from "other people", z.

Example 7.4.5: Inequality

7.4.5 Encoding errors

For the final four norms, the main attribute that led to their FOL encoding being *Not Horn* were based on random errors made during the encoding. These errors were either a consequence of over-interpretation done by the system or by including wrongly negated predicates. No patterns were found in the norms within this category.

Chapter 8

Discussion

In this chapter, we present some topics worthy of discussion based on the findings in this thesis. We start by discussing the faithfulness of the encoded FOLs in a bigger picture and how these insights are valuable. In Section 8.2 we discuss the limitations of the Horn clause mistakes following the FOL encodings. We finish the chapter by discussing the use of GPT-4 as a knowledge engineer simulator in Section 8.3.

8.1 First-Order Logic Faithfulness

In the analysis of faulty FOL encodings described in Chapter 7, we found numerous types of encoding mistakes. These are the mistakes that follow from the GPT-4 encoding of norms from the NL norm dataset in terms of syntax and intended meaning. Among the three classifications of FOL encodings, *Syntactically Invalid*, *Wrong* and *Lacking*, we count 11 distinct attribute pattern categories, which are listed in Table 7.1. Each attribute pattern category contains FOL encodings that have the same fault attribute. Each FOL that was encoded wrong in terms of syntax or meaning was categorized into one or more of these categories, based on the attribute that led to the FOL being wrong. In addition, we count 25 attribute subcategories, which are categories that further specify the mistake attributes. The high number of attribute pattern categories indicates that the types of mistakes GPT-4 makes when encoding FOLs are relatively unpredictable.

Some of the observed attribute pattern categories are more unpredictable than others, in terms of what specifically causes the mistake. In general, it seems to be more difficult to predict which norms would be encoded with variable/constant- or predicate errors than to predict which norms would be encoded with interpretative errors, time property errors, or missing parts. This is likely due to the fact that predicates and variables are present in multiple places in each FOL encoding, and the mistake can occur in any one of these places. As long as we cannot find any patterns in which a specific norm attribute leads to this predicate/variable error, the occurrence of such an error is hard to predict. Interpretive errors, time property errors, and missing parts, on the other hand, are limited to distinct parts of the FOL and may only be relevant in certain norm structures. These norm structures include norms that express counter-intuitive moral judgments, norms that contain temporal dimensions, and norms that include redundant additional information. For the more predictable classification patterns, it is easier to implement improvement solutions. We show an example of how this can be done with the FOLs classified with wrong evaluation constant (subcategory of interpretative errors) by creating a sentiment analysis model able to correct the majority of these mistakes. This shows how analyzing the mistakes made in GPT-4 encoding can help us find solutions that will mitigate these issues and produce better results.

In the encoding system described in this thesis, we found that GPT-4 is able to encode 51% of norms from the NL norm dataset to FOLs containing no faults using the specific instructions and parameters described. Although these are arguably good results, we still need to turn the focus to the FOL encodings that were not encoded without fault. For systems such as AMAs that perform actions that could lead to consequences for people, it is crucial to ensure that the knowledge bases on which they are built do not contain faulty rules or instructions. By analyzing the potential faults of current alternatives for constructing such KBs, it is easier to improve these alternatives.

The findings in this thesis present both opportunities and challenges for the task of developing ethical autonomous systems. By automatically aligning machine behavior with societal norms through social choice aggregation, we can efficiently create systems that behave in ways consistent with human values.

GPT-4 is a large language model that, simply put, creates responses by constructing "educated guesses" based on user input and patterns from its training data. Although the model has learned from a vast amount of data exactly how to make these guesses, it lacks a human-level understanding of the concepts it considers. Additionally, the data on which GPT-4 is trained is human-based text, which contains common human errors, such as over-generalizing and biases [Van Dis et al., 2023]. The constructed guesses of GPT-4 will therefore undoubtedly contain occasional errors. Exactly how faulty the model's responses are will depend on the tasks and the topics that are considered, in addition to how well the users are able to design the instructions that are given to the model.

8.2 Horn Clause Mistakes

For the *Not Horn* mistakes, we find a much narrower and more predictable set of mistake patterns. In general, we find that norms that in some way express negation in the norm part of the sentence lead to FOLs that are not Horn. These are the norms described in the *negation phrase in norm part* pattern attribute category. Examples of such norms can be seen in Example 8.2.1. Since the encoding system is set up in a way where we always want positive predicates in the consequent (the evaluation part), we simply need one or more predicates in the norm part to be negated in order for the FOL representation to not be Horn. The distinct phrases that led to negation in the norm part of the FOL encoding are listed in Definition 1.

"it's expected that you won't cut off family members"
"you shouldn't trust anyone who isn't trustworthy."
"it's bad to not take things seriously."
"it's rude to borrow money from a friend and never pay them back."
"it's wrong to take pictures of people without their consent."

Example 8.2.1: Norms with negation phrase in norm part

The second largest group of norms leading to invalid Horn representations is the *negation phrase in evaluation part*-category. Examples of norm belonging to this group can be seen in Example 8.2.2.

"people are not expected to take care of their loved ones"
"getting circumcised isn't normal"
"it is not creative work to write an english paper."
"if you get a flat tire your spouse isn't expected to help you change it."

Example 8.2.2: Norms with negation phrase in evaluation part

We argue that the faults in this category is due to faulty encoding by GPT-4 rather than an invalid Horn pattern in the norm. This is because it is clearly stated in the prompt instruction in the system that negated evaluations should be represented in the evaluation part of the encoding. There are some additional norm categories that lead to *Not Horn* classification such as the use of negated quantifiers and inequality representations which are less predictable than the "negation phrase" categories, but these only make up a minority of the *Not Horn*s.

By limiting norm structure inputs and using pattern matching tools, such as Regular Expressions (RegEx), we believe that some of these norm patterns leading to invalid Horn representations can be detected and rephrased. Limiting the variability of moral judgments in the inputs by restricting phrasing of the evaluation part to be "it's good to..." or "it's bad to..." ensures that no norms should contain a negation phrase in the evaluation part.

Using pattern matching tools, the specific phrases leading to invalid Horn structures when used in the norm part can be easily identified in a norm input and requested to be rephrased. Although these suggested methods will not ensure that all norms are phrased in a way that guarantees they are encoded with valid Horn clause structure, we argue that implementing a filter system based on the findings in this thesis could generally improve the process of encoding norms to Horn clause structure.

8.3 GPT-4 and Knowledge Engineering

We clearly see how the "independent thinking" of GPT-4 allows the model to encode a wide variety of sentence structures with limited domain-specific training data. However, it also allows the model to consider irrelevant interpretations in the encoding process, which can lead to syntactically valid FOLs that are unfaithful in terms of meaning. These results can be hard to automatically recognize, as there are currently no sufficient methods to measure and compare meaning. This demonstrates the need for expert evaluations in combination with automatic encoding methods. Manually encoding FOLs from scratch takes time and requires knowledge engineers with a good understanding of logic and encoding. Manually reviewing and correcting suggested FOL encodings, on the other hand, requires less time and training. This was experienced first-hand during this thesis, as the time spent on evaluating the faithfulness of the proposed FOL encodings was much shorter than the time spent encoding FOLs from scratch for the NL norm dataset. The benefits of incorporating human correction to automatic encoding are something we find through this thesis to be worth exploring further.

The use of AI systems has the potential to contribute to a number of different fields by serving as an aid to human expertise. At this point, we cannot and should not expect these systems to replace human expertise, as demonstrated by the GPT encoding results of this thesis. However, by identifying and addressing the mistakes that GPT makes in specific tasks, we can still leverage its capabilities. By using GPT's ability to encode norms in combination with sentiment analysis corrections and human expert verification, we believe that one can significantly reduce the need for human expertise.

Chapter 9

Conclusions and Future work

In this thesis, we have explored automatic encoding of natural language norms into first-order Horn clauses, with the aim of aligning autonomous machine behavior with societal norms. Our study used OpenAI's GPT-4 model as a key component of the encoding system, employing prompt engineering strategies and utilizing tools such as sentiment analysis to mitigate mistakes made by GPT-4 and answer set programming to find inconsistencies in the encodings.

9.1 Research Questions and Success Criteria

We discuss how the thesis addresses each success criterion listed in Chapter 1 and how the research question is answered.

First, the thesis demonstrates a comprehensive understanding of the theoretical framework upon which it is based. Chapters 2 and 3 introduce the theoretical concepts and literature related to the subjects and objectives included in this thesis. We outline the problem of encoding natural language to logic by presenting the basis of natural language parsing and LLMs and summarizing the latest relevant research on the topic. We present the problem of moral conflict and inconsistency in AMAs and approaches to how this can be solved by explaining the basics of logic programming and summarizing the research in this field.

Second, the research introduces an encoding system that shows GPT-4's ability to encode natural language norms to FOL formulas. This is done by using GPT-4 as the main tool for encoding natural language norms into FOL representations. By manually counting the instances that are encoded accurately in terms of syntax and intended meaning, we show how well GPT-4 is able to encode norms to FOL.

Third, the thesis identifies and describes the norm patterns that GPT-4 is unable to convert to FOL. We present a thorough analysis in Sections 7.1, 7.2 and 7.3 on the types of norms that GPT-4 is not able to encode well, both in terms of syntax and meaning. The analysis describes the FOL attributes that lead to faulty representations and whether there are patterns in the norms that cause the faults.

Finally, in Section 7.4 the thesis describes the norm patterns that cannot be transformed to first-order Horn clauses and the reasons why this is the case. We identify multiple such patterns that highlight the limitations of the Horn logic structure.

Through the course of our research, we addressed the research question and met the success criteria established in Chapter 1. We demonstrated the feasibility of encoding human norms into Horn clause representations using GPT-4, in accordance with the RQ, and evaluated the effectiveness of our encoding system against the established benchmarks. We used the results of the encoding system to identify norms that cannot be expressed as Horn clauses. We identified norms that explicitly or implicitly mention a negation phrase in the norm part of the sentence to be the most common of the norm structures incompatible with the Horn clause format.

9.2 Contribution

The key contributions of this thesis include:

- Encoding System: We developed a system that converts natural language norms into machine-readable first-order Horn clauses, contributing to the field of AI ethics.

- Analysis of GPT-4 Limitations: We conducted a detailed analysis of the types of natural language norms that GPT-4 struggles to accurately encode into FOL.

- Identification of Horn clause limitations: We identified specific norm structures that are inherently incompatible with Horn clauses, contributing valuable insights into the limitations of this logic format.

- Datasets: We produced two datasets, one with natural language norms and their respective manually created FOL encodings, called the NL norm dataset. The second dataset contains the norms from the NL norm dataset along with their automatically encoded first-order and Horn clause representations, including evaluations of their faithfulness.

The findings are a contribution to the field of machine ethics, but there are many other areas in which logic reasoning and consistency checking are valuable, such as moral conflict and knowledge representation and reasoning. By showcasing how GPT-4 and similar LLMs can serve as knowledge engineers, we open new possibilities for developing autonomous systems that align with social values. Additionally, our inconsistency checker ensures logical consistency, an important aspect in machine ethics that prevents conflicting norms from undermining the ethical behavior of autonomous agents.

9.3 Limitations and Future Work

Although the encoding system performed well, it has some limitations. The reliance on restricted moral judgments in the evaluation part of the encodings may limit expressiveness, and the handling of more complex norm structures could be improved. Future work should focus on refining the prompt engineering strategies, exploring alternative logical formats, and exploring how alternative language models compare to GPT-4 in terms of encoding. Additional strategies for correcting the mistakes made by GPT-4 should be explored and implemented into the system. The inconsistency checker should be adjusted to create full-scale knowledge bases to be used in autonomous moral machines. We recommend finding ways to filter out norms that tend to lead to invalid Horn structures based on the findings in this thesis. Finally, it should be verified whether the collaboration between automated encoding and manual correction is more effective than employing either approach independently.

9.4 Final Thoughts

In conclusion, this thesis presents a proof of concept for automated encoding of natural language norms for autonomous moral agents, contributing to the broader goal of ensuring ethical machine behavior. As autonomous systems become more integrated into our lives, aligning their decision making with human values becomes increasingly important, and the work presented here offers a foundational step in that direction.

Chapter 10

Appendix

A Literature Search Method

The motivation for this thesis comes from the works of Ozaki et al. [2022] and Jiang et al. [2021]. Hence, these articles served as a starting point for our literature review. We started with a very narrow focus, solely on the topic of natural language to Horn clause encoding. We searched the following online repositories: Google Scholar, dblp.org, Papers with Code, and arXiv. We chose these academic databases because they are perceived as the best for providing research in computer science. The databases allow different parts of the articles to be searched; Google Scholar lets you search in the full text of the publication, including title, abstract, and authors, arXiv and Papers with Code searches are based on title, author, and abstract, while DBLP restricts the search to titles only. The initial search was carried out using the following search phrases:

- *"natural language to horn clause logic"*
- *"natural language horn clause translation"*
- *"horn clause parser"*
- *"NL to horn"*
- *"horn logic translation"*
- *"natural language horn parsing"*
- *"NLP horn logic"*

In the narrow search, only articles published between 1990 and the early 2000s appeared. The field of modern NLP combines the use of new-age technologies, such as machine learning and artificial intelligence, with classical natural language processing techniques. This field experienced a great development in the early 2010s with the introduction of digital assistants such as Siri, Apple's voice-controlled assistant, and modern deep learning algorithms such as the Transformer model [Vaswani et al., 2017, Schmidhuber, 2015, Apple Inc., 2023]. Based on this, we chose to limit the search even more and only include articles published after 2010.

We were unable to find any out-of-the-box tools that could convert natural language norms into Horn clauses. However, we were able to find tools that encode NL sentences in FOL. As Horn is a fragment of FOL, these are useful to us. We found more than five articles on the topic of NL to FOL translation and decided to prioritise the newest publications. Furthermore, we manually searched the reference list of the latest article on the topic, Yang et al. [2023], and found five additional relevant articles. Articles older than 2010 or with newer available research were excluded. All included articles are primary sources.

The final phase of the literature review consisted of searching for work within the fields of moral conflict and prompt engineering. We wanted to provide a general understanding of the problem of moral conflicts and the most common approaches to solving the problem. We found a literature review on the topic by Santos et al. [2017] that provides an overview of the most prominent research in the field. In addition, we searched Google Scholar to find well-established prompt engineering techniques on which we base the GPT-4 interaction. We found an in-depth overview of recent advances in LM by Min et al. [2023], as well as a range of articles exploring the application and results of several prompt engineering techniques.

The literature search lasted from August 2023 to May 2024, and all research is limited to up until this point in time.

Bibliography

Wjdan Alharthi. First-order logic resolution. `https://github.com/wjdanalharthi/First-order-Logic-resolution`, 2019. Accessed January 2024.

K. Allan. *Natural Language Semantics*. Wiley, 2001. ISBN 9780631192961. URL `https://books.google.no/books?id=xX-5QgAACAAJ`.

Michael Anderson and Susan Leigh Anderson. The status of machine ethics: a report from the AAAI Symposium. *Minds and Machines*, 17(1):1–10, March 2007. ISSN 1572-8641. doi: 10.1007/s11023-007-9053-7. URL `https://doi.org/10.1007/s11023-007-9053-7`.

Apple Inc. Siri. `https://www.apple.com/siri/`, April 2023. Accessed April 2024.

Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, and Christopher Re. Ask me anything: A simple strategy for prompting language models. In *The Eleventh International Conference on Learning Representations*, 2022.

Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003. ISBN 978-0-521-81802-5.

Kinjal Basu, Sarat Varanasi, Farhad Shakerin, Joaquin Arias, and Gopal Gupta. Knowledge-driven natural language understanding of english text and its applications, 2021.

Seth D Baum. Social choice ethics in artificial intelligence. *Ai & Society*, 35(1):165–176, 2020.

Emily M. Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.463. URL `https://aclanthology.org/2020.acl-main.463`.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL `https://doi.org/10.1145/3442188.3445922`.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59:65–98, 2017. doi: 10.1137/141000671.pdf.

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

Per Kristian Bjørkeng. *Kunstig intelligens: Den usynlige revolusjonen.* Vega Forlag AS, 2 edition, 2018.

George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic.* Cambridge University Press, 5 edition, 2007.

Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558609326.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

K. R. Chowdhary. *Natural Language Processing*, pages 603–649. Springer India, New Delhi, 2020. ISBN 978-81-322-3972-7. doi: 10.1007/978-81-322-3972-7_19. URL https://doi.org/10.1007/978-81-322-3972-7_19.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Vincent Conitzer. Why should we ever automate moral decision making? In *CEUR Workshop Proceedings*, volume 3547. CEUR Workshop Proceedings, 2023.

Vincent Conitzer, Rachel Freedman, Jobst Heitzig, Wesley H Holliday, Bob M Jacobs, Nathan Lambert, Milan Mossé, Eric Pacuit, Stuart Russell, Hailey Schoelkopf, et al. Social choice for ai alignment: Dealing with diverse human feedback. *arXiv preprint arXiv:2404.10271*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Sabit Ekin. Prompt engineering for chatgpt: A quick guide to techniques, tips, and best practices. *Authorea Preprints*, May 2023. doi: 10.36227/techrxiv.22683919.v2. URL http://dx.doi.org/10.36227/techrxiv.22683919.v2.

Denis Emelin, Ronan Le Bras, Jena D. Hwang, Maxwell Forbes, and Yejin Choi. Moral stories: Situated reasoning about norms, intents, actions, and their consequences. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 698–718, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.54. URL https://aclanthology.org/2021.emnlp-main.54.

Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, Oct. 2016. doi: 10.1609/aimag.v37i3.2678. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2678.

Salvatore Claudio Fanni, Maria Febi, Gayane Aghakhanyan, and Emanuele Neri. *Natural Language Processing*, pages 87–99. Springer International Publishing, Cham, 2023. ISBN 978-3-031-25928-9. doi: 10.1007/978-3-031-25928-9_5. URL https://doi.org/10.1007/978-3-031-25928-9_5.

Maxwell Forbes, Jena D. Hwang, Vered Shwartz, Maarten Sap, and Yejin Choi. Social chemistry 101: Learning to reason about social and moral norms, 2021.

LTF Gamut. *Logic, language, and meaning, volume 1: Introduction to logic*, volume 1. University of Chicago Press, 1991.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Motivation*, pages 1–10. Springer International Publishing, Cham, 2013. ISBN 978-3-031-01561-8. doi: 10.1007/978-3-031-01561-8_1. URL https://doi.org/10.1007/978-3-031-01561-8_1.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Chaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019. doi: 10.1017/S1471068418000054.

Claude Cordell Green. *The application of theorem proving to question-answering systems*. Stanford University, 1969.

Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*, 2023.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. Folio: Natural language reasoning with first-order logic, 2022.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning {ai} with shared human values. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=dNy_RKzJacY.

Linmei Hu, Zeyi Liu, Ziwang Zhao, Lei Hou, Liqiang Nie, and Juanzi Li. A survey of knowledge enhanced pre-trained language models. *IEEE Transactions on Knowledge and Data Engineering*, 36(4):1413–1430, 2024. doi: 10.1109/TKDE.2023.3310002.

Hugging Face. Transformers, what can they do? `https://huggingface.co/learn/nlp-course/en/chapter1/4`, 2023. Accessed May 2024.

Hugging Face. Distilbert documentation. `https://huggingface.co/docs/transformers/en/model_doc/distilbert`, 2024. Accessed May 2024.

Liwei Jiang, Jena D. Hwang, Chandra Bhagavatula, Ronan Le Bras, Maxwell Forbes, Jon Borchardt, Jenny T Liang, Oren Etzioni, Maarten Sap, and Yejin Choi. Delphi: Towards machine ethics and norms. *ArXiv*, abs/2110.07574, 2021. URL `https://api.semanticscholar.org/CorpusID:238857096`.

Epaminondas Kapetanios, Doina Tatar, and Christian Sacarea. *Natural language processing: semantic aspects*. CRC Press, 2013.

Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

Robert Kowalski. Logic programming in the 1970s. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 11–22, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40564-8.

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, 2019.

Changcheng Li. JuliaCall: an R package for seamless integration between R and Julia. *The Journal of Open Source Software*, 4(35):1284, 2019. doi: 10.21105/joss.01284.

Beishui Liao, Pere Pardo, Marija Slavkovik, and Leon van der Torre. The jiminy advisor: Moral agreements among stakeholders based on norms and argumentation. *Journal of Artificial Intelligence Research*, 77:737–792, 07 2023. doi: 10.1613/jair.1.14368.

Vladimir Lifschitz. What is answer set programming? *Association for the Advancement of Artificial Intelligence*, 2008.

Vladimir Lifschitz. *Answer set programming*, volume 3. Springer Heidelberg, 2019.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, 2021.

John W Lloyd. *Foundations of logic programming.* Springer Science & Business Media, 2012.

Nicholas Lourie, Ronan Le Bras, and Yejin Choi. Scruples: A corpus of community ethical judgments on 32,000 real-life anecdotes, 2021.

Xuantao Lu, Jingping Liu, Zhouhong Gu, Hanwen Tong, Chenhao Xie, Junyang Huang, Yanghua Xiao, and Wenguang Wang. Parsing natural language into propositional and first-order logic with dual reinforcement learning. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5419–5431, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL `https://aclanthology.org/2022.coling-1.481`.

D. Makinson. *Sets, Logic and Maths for Computing.* Undergraduate Topics in Computer Science. Springer London, 2012. ISBN 9781447125006. URL `https://books.google.no/books?id=zz9wDBjsqlAC`.

Dastan Hussen Maulud, Subhi RM Zeebaree, Karwan Jacksi, Mohammed A Mohammed Sadeeq, and Karzan Hussein Sharif. State of art for semantic analysis of natural language processing. *Qubahan academic journal*, 1(2):21–28, 2021.

Terrance McConnell. Moral Dilemmas. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, Fall 2022 edition, 2022.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014. ISSN 2090-4479. doi: https://doi.org/10.1016/j.asej.2014.04.011. URL `https://www.sciencedirect.com/science/article/pii/S2090447914000550`.

Dennis Merritt. *Building expert systems in Prolog.* Springer Science & Business Media, 2012.

Microsoft. Prompt engineering techniques, 2024a. URL `https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/advanced-prompt-engineering?pivots=programming-language-chat-completions`. Accessed January 2024.

Microsoft. Use follow-up prompts to create multiple turns of a conversation, 2024b. URL `https://learn.microsoft.com/en-us/azure/ai-services/qnamaker/how-to/multi-turn`. Accessed January 2024.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Comput. Surv.*, 56(2), sep 2023. ISSN 0360-0300. doi: 10.1145/3605943. URL https://doi.org/10.1145/3605943.

A.C. Müller and S. Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016. ISBN 9781449369903. URL https://books.google.no/books?id=1-4lDQAAQBAJ.

OpenAI. Fine-tuning, 2024a. URL https://platform.openai.com/docs/guides/fine-tuning. Accessed January 2024.

OpenAI. Prompt engineering, 2024b. URL https://platform.openai.com/docs/guides/prompt-engineering. Accessed January 2024.

OpenAI. How chatgpt and our language models are developed. https://help.openai.com/en/articles/7842364-how-chatgpt-and-our-language-models-are-developed, 2024c. Accessed January 2024.

OpenAI. Openai python api. https://platform.openai.com/docs/api-reference/introduction, 2024d. Version 1.11.1.

OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2023.

Ana Ozaki, Anum Rehman, Philip Turk, and Marija Slavkovik. Finding common ground for incoherent horn expressions. *CoRR*, abs/2209.06455, 2022. doi: 10.48550/ARXIV.2209.06455. URL `https://doi.org/10.48550/arXiv.2209.06455`.

Federico Pascual. Getting started with sentiment analysis using python. https://huggingface.co/blog/sentiment-analysis-python, February 2022.

Giulio Petrucci, Marco Rospocher, and Chiara Ghidini. Expressive ontology learning as neural machine translation. *Journal of Web Semantics*, 52-53:66–82, 2018. ISSN 1570-8268. doi: https://doi.org/10.1016/j.websem.2018.10.002. URL https://www.sciencedirect.com/science/article/pii/S1570826818300507.

Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar, and Gopal Gupta. Reliable natural language understanding with large language models and answer set programming. *Electronic Proceedings in Theoretical Computer Science*, 385:274–287, September 2023. ISSN 2075-2180. doi: 10.4204/eptcs.385.27. URL http://dx.doi.org/10.4204/EPTCS.385.27.

Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 1105–1116, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385626. doi: 10.1145/3468264.3468588. URL https://doi.org/10.1145/3468264.3468588.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

Mohammed Saeed, Naser Ahmadi, Preslav Nakov, and Paolo Papotti. Rulebert: Teaching soft rules to pre-trained language models, 2021.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

Jessica Santos, Jean Zahn, Eduardo Silvestre, Viviane Silva, and Wamberto Vasconcelos. Detection and resolution of normative conflicts in multi-agent systems: a literature survey. *Autonomous Agents and Multi-Agent Systems*, 31, 11 2017. doi: 10.1007/s10458-017-9362-z.

Maarten Sap, Saadia Gabriel, Lianhui Qin, Dan Jurafsky, Noah A. Smith, and Yejin Choi. Social bias frames: Reasoning about social and power implications of language. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5477–5490, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.486. URL https://aclanthology.org/2020.acl-main.486.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2014.09.003. URL `https://www.sciencedirect.com/science/article/pii/S0893608014002135`.

Pankaj Sharma, Naveen Malik, Naeem Akhtar, and Hardeep Rohilla. Parsing techniques: A review. *International Journal of Advanced Research in Engineering and Applied Sciences*, 2(10):65–76, 2013. URL `http://www.garph.co.uk/IJAREAS`.

Barry Shore. Bias in the development and use of an expert system: implications for life cycle costs. *Industrial Management & Data Systems*, 96(4):18–26, 1996.

Hrituraj Singh, Milan Aggrawal, and Balaji Krishnamurthy. Exploring neural models for parsing natural language into first-order logic, 2020.

Marija Slavkovik. Automating Moral Reasoning. In Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza, editors, *International Research School in Artificial Intelligence in Bergen (AIB 2022)*, volume 99 of *Open Access Series in Informatics (OASIcs)*, pages 6:1–6:13, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-228-0. doi: 10.4230/OASIcs.AIB.2022.6. URL `https://drops-dev.dagstuhl.de/entities/document/10.4230/OASIcs.AIB.2022.6`.

Suzanne Tolmeijer, Markus Kneer, Cristina Sarasua, Markus Christen, and Abraham Bernstein. Implementations in machine ethics: A survey. *ACM Comput. Surv.*, 53(6), dec 2021. ISSN 0360-0300. doi: 10.1145/3419633. URL `https://doi.org/10.1145/3419633`.

M. Onat Topal, Anil Bas, and Imke van Heerden. Exploring transformers in natural language generation: Gpt, bert, and xlnet, 2021.

Alan M. Turing. Computing machinery and intelligence. *Mind*, LIX:433–460, 1950. URL `https://api.semanticscholar.org/CorpusID:14636783`.

Rens Van de Schoot and Milica Miočević. *Small sample size solutions: A guide for applied researchers and practitioners*. Taylor & Francis, 2020.

Eva AM Van Dis, Johan Bollen, Willem Zuidema, Robert Van Rooij, and Claudi L Bockting. Chatgpt: five priorities for research. *Nature*, 614(7947):224–226, 2023.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Juan David Velásquez-Henao, Carlos Jaime Franco-Cardona, and Lorena Cadavid-Higuita. Prompt engineering: a methodology for optimizing interactions with ai-language models in the field of engineering. *DYNA*, 90(230):9–17, nov. 2023. doi: 10.15446/dyna.v90n230.111700. URL https://revistas.unal.edu.co/index.php/dyna/article/view/111700.

Wendell Wallach and Colin Allen. *Moral Machines: Teaching Robots Right from Wrong*. Oxford University Press, 02 2009. ISBN 9780195374049. doi: 10.1093/acprof:oso/9780195374049.001.0001. URL https://doi.org/10.1093/acprof:oso/9780195374049.001.0001.

Wendell Wallach, Colin Allen, and Iva Smit. Machine morality: bottom-up and top-down approaches for modelling human moral faculties. *AI & SOCIETY*, 22(4):565–582, 2008. ISSN 1435-5655. doi: 10.1007/s00146-007-0099-0. URL https://doi.org/10.1007/s00146-007-0099-0.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt, 2023.

Yangjian Wu and Gang Hu. Exploring prompt engineering with GPT language models for document-level machine translation: Insights and findings. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 166–169, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.15. URL https://aclanthology.org/2023.wmt-1.15.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. Harnessing the power of large language models for natural language to first-order logic translation, 2023.

Caiming Zhang and Yang Lu. Study on artificial intelligence: The state of the art and future prospects. *Journal of Industrial Information Integration*, 23:100224, 2021. ISSN 2452-414X. doi: https://doi.org/10.1016/j.jii.2021.100224. URL https://www.sciencedirect.com/science/article/pii/S2452414X21000248.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue Zhang, Neil Zhenqiang Gong, and Xing Xie. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts, 2023.