Karlsruhe Institute of Technology
Department of Informatics
Institute for Anthropomatics and Robotics

Master Thesis

Extrapolating Beyond the Imitation Game - Teaching Large Language Models Reasoning in First-Order Logic

---

Author: Simon Döbele (2381466)

Thesis committee: Prof. Dr. Jan Niehues
Prof. Dr. Gregor Betz

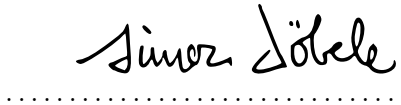
A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Systems.

September 28, 2023

Declaration in English: I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Erklärung auf Deutsch: Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, September 28, 2023



Simon Döbele

Abstract

Large language models (LLMs), such as GPT-3 and more recently Llama-2, have achieved ever more impressive results on many natural language understanding tasks. Benchmarks such as BIG Bench hard needed to exclude certain tasks, because LLMs have managed to perform so well on them. Finding ever more challenging reasoning tasks for LLMs has been of much interest. On the other hand, LLMs still make silly reasoning mistakes or hallucinate, that is, make false claims as if they were true. Alleviating these mistakes and hallucinations has equally been of much interest. That is why, in this thesis, we aim to teach LLMs to emulate reasoning in first-order (predicate) logic.

In order to address this challenge, we design and build a first-of-its-kind synthetic dataset that we call the "Synthetic Predicate Logic Corpus" (or SPLC), which includes three tasks in reasoning using both natural language and the artificial language of predicate logic. By making use of a model checker, we can automatically generate the labels; and by building (or modifying) semantic parsers, we can map between natural language and the language of logic. Besides automatic labeling, the big advantage of our dataset is that we can also adjust its difficulty. We produce a baseline that we compare our models' performance to.

In over 150 experiments, we show the first empirical demonstration that the Falcon, Llama, Orca, and Wizard LLMs can emulate logical reasoning in first-order logic when using LoRA adapters. We find that they are only able to generalize to more difficult tasks to a small extent, although scaling is not robust.

To the person who most discussed AI with me

Acknowledgements

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of this thesis. This journey has been rewarding and fulfilling, and it would not have been possible without your support and guidance.

First and foremost, I extend my deepest appreciation to my academic advisors, Prof. Dr. Gregor Betz and Prof. Dr. Jan Niehues. Your unwavering support, your time for regular discussions, your insightful guidance, and mentorship have been invaluable throughout this research. A big thank you to Prof. Dr. Gregor Betz, for preparing me for writing a thesis on language models during his seminar, for his expertise in the area of logic and language modeling, and for proposing to create something new for this exciting field. A big thank you to Prof. Dr. Jan Niehues for his expertise in the area of artificial intelligence for language technologies and thank you to him and his Phd student Sai Koneru for providing access to their GPU cluster. Both of my advisor's expertise and dedication to my academic growth have positively impacted not only this thesis but also my overall development as a researcher.

I also extend my thanks to the many professors and teachers who have shaped my academic journey: thank you to Prof. Rainer Hegselmann for instilling my love for logic and argumentation theory; thank you to Prof. Olivier Roy, professor of philosophy, who let me write my first essay at university about John Searle's Chinese Room Argument, which kickstarted my interest in AI; thank you to Dr. Donal Khosrowi Djen-Gheschlaghi, for teaching me how to write clearly and precisely; thank you to John DeNero, for growing my interest and love for coding; thank you to Prof. Johan Boye, for teaching me the basics of language models.

To my wife, friends and family, your unwavering support, patience, and encouragement were my pillars of strength throughout this journey. I am profoundly

grateful for your belief in me and your understanding during the challenging times.

Hardware (GPU) support for this research was provided both by Prof. Jan Niehues and by BW Unicluster and I am grateful for their generous access to use their resources, which enabled me to carry out this study.

Financial support during my thesis was provided by Max-Weber Programm of the Free State of Bavaria, and I thank them for their generous support. The financial assistance greatly alleviated the pressures associated with study expenses.

In conclusion, I am thankful to each and every individual and organization mentioned here, as well as those whose support may not be explicitly mentioned but was nonetheless significant. Your contributions have played an integral role in the completion of this thesis and my academic journey.

Thank you all for being part of this remarkable chapter in my life.

Simon Döbele

September 2023

Contents

List of Figures

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

In his seminal article "Computing Machinery and Intelligence", Alan Turing proposes to measure whether a machine can think by a test, that we can call the Turing Test or the Imitation Game. In this test, the machine communicates in writing with a human interrogator, who does not know whether he writes with a machine or a human being. The goal of the machine is to pass as a human being, such that the interrogator cannot distinguish between a human and the machine (Turing, 1950). By his thought experiment, Turing has motivated researchers with the idea of making machines master linguistic intelligence.

Current research directions in AI for natural language processing have largely abandoned this kind of measure, and turned to benchmark datasets as the norm for showing progress (Raji et al., 2021). Leaderboards for the language models that score the highest in these benchmarks abound (see e.g. (Beeching et al., 2023)). Current research directions also employ much more fine-grained measures than the one proposed by Turing - not necessarily to measure intelligence per se, however, but rather regarding specific tasks, such as translation, summarization, question answering or different kinds of reasoning tasks.

Large pre-trained transformer language models, such as GPT-2 and GPT-3, have achieved ever more impressive results on many such natural language understanding tasks (see e.g. (Radford et al., 2019), (Brown et al., 2020)), and even attracted media attention (especially in the form of ChatGPT). However, even these large language models (LLMs) still make silly reasoning mistakes (Lourie et al., 2021) or hallucinate, that is, they make false claims as if they were true (see e.g. (McKenna et al., 2023), (Mündler et al., 2023), (Kassner and Schütze, 2020)). Some would even go so far as to argue that, while large language models possess knowledge of linguistic rules and patterns (i.e. a formal linguistic competence), they lack a functional linguistic competence (such as formal reasoning skills) (Mahowald et al., 2023).

1. INTRODUCTION

Maybe language models are just eloquent stochastic parrots, being able to imitate enough aspects of producing language that humans consider them intelligent (Bender et al., 2021). Or maybe language models show at least some sparks of intelligence and we can explain their shortcomings where they exist: Maybe the misinformation, the hallucinations and the bad reasoning are inherent in the training data, the billions and trillions of text corpora from the world wide web that are not preprocessed enough to make high-quality datasets. Coherent with the latter view are the findings that humans make errors in reasoning (Kahneman, 2011), and these mistakes form part of that data.

In any case, both views seem to agree that it would be valuable to have language models that not only prevent hallucinations, but even show robust logical reasoning abilities to back up their claims. Being able to reason well, is one of the most fundamental skills for any inquirer into the nature of things. Logical reasoning is at the heart of our scientific enterprise. Making use of available evidence to form valid arguments, or hypothesizing one's way to a conclusion, all help in advancing knowledge.

Automating reasoning has been one of the central goals of AI. Benchmarks that include reasoning tasks to test language models have been developed. One such example is BIG BENCH (a benchmark going **B**eyond the **I**mitation **G**ame) (Srivastava et al., 2022). In one of its reasoning tasks, a language model is presented with a description of changes in the ordering of objects and then asked at which location an object ends up. An arguably more advanced task asks the language model to deduce whether an argument, presented in natural language, is valid (Betz et al., 2020). This benchmark still includes tasks that language models can only perform with random accuracy (Wei et al., 2022). Finding such tasks is a major endeavour of scientists that work on AI for natural language processing (NLP) for two reasons: first, to capture the shortcomings of currently existing language models, and second, in order to test when they manage to move past their shortcomings.

In this thesis, we also try to move beyond the imitation game, and even beyond the existing benchmarking approaches such as BIG Bench that aim to evaluate AI reasoning capabilities. Most of the logical reasoning tasks discussed in the literature are either based on natural language only, converting a natural language input to natural language output, or purely based on formal logic, converting a formal input to a formal output. We hypothesize that our tasks, involving a mix of natural and formal input and output, help language models learn to emulate semantic reasoning in first-order (predicate) logic. We base our tasks on how predicate logic might be taught to students in an introductory logic course.

To this end, we extrapolate BIG (both BIG Bench and beyond the imitation game) by designing and building a synthetic dataset that we call the "Synthetic Predicate Logic Corpus" (or SPLC). There is no need for hand-labelling the SPLC, as we make use of a model checker that automatically generates the labels as well as semantic parsers that map between natural language and the language of logic. While being able to automatically infer the labels is one of its biggest advantages, another big advantage is that we can adjust its difficulty as well.

In summary, the main contributions of this thesis are:

- Construct a first-of-its kind synthetic dataset (the "SPLC") for reasoning in predicate logic, that is automatically labelled (instead of by hand) and whose difficulty is adjustable.
- Present a thorough descriptive analysis of this dataset.
- Derive three tasks from the dataset that involve both natural language and the language of predicate logic.
- Build (and modify) parsers mapping between natural language and the language of predicate logic.
- Perform fewshot and zeroshot evaluation of current state-of-the-art (SOTA) large language models on the three tasks.
- Finetune and evaluate to what extent current SOTA large language models can emulate reasoning in predicate logic according to SPLC, including how well they generalize between tasks and to even harder tasks.
- In total, we perform 165 distinct experiments on the SPLC.

Note that this thesis is written such that it should be understandable by someone who has studied a core curriculum of computer science, but not necessarily anything related to neural networks or language models. Nevertheless, it may also be understood by someone from other disciplines with a rigorous mathematical background.

1.1 Problem Statement, Design and Research Questions

The general problem we aim to solve is: "How can we teach large language models reasoning in predicate logic?"

From this follows a **Design Question**: "How should a dataset look like that teaches large language models predicate logic?"

In order to tackle this problem after having designed a suitable dataset, we formulate the following research questions:

Research Question 1: What is a suitable **baseline** for the three tasks?

Research Question 2: How high do current pre-trained large language models score in these tasks in a **zeroshot evaluation**?

Research Question 3: How high do current pre-trained large language models score in these tasks in a **fewshot evaluation**?

Research Question 4: Does different **prompting** change the fewshot results?

Research Question 5: Can current SOTA large language models emulate reasoning in first-order predicate logic?

Since we have multiple different tasks, that we can train the models on, and the possibility to generate harder datasets, we also hope to answer the following questions:

Research Question 6: Given a finetuned model that does well on one task, does this generalize to one of the other tasks?

Research Question 7: Does multitask learning (i.e. finetuning a large language model on all three tasks) lead to higher accuracy in the respective tasks compared to the singletask setting?

Research Question 8: Given a finetuned model that does well on one task, does this generalize to a harder version of the same task?

1.2 Thesis outline

In chapter two, we explain the necessary concepts, techniques and models to help the reader understand the remainder of this thesis. This includes, first, an introduction to neural networks, from plain vanilla feedforward neural networks all the way to transformer models. Second, it will describe how language (or word tokens) are represented in a machine and then show how the task of language modeling has been tackled with simpler n-gram and more advanced neural models, such as transformer models. Third, we introduce techniques to finetune large language models even with computational resource constraints. Fourth, we provide an overview over the syntax and semantics of predicate logic. Finally, we dileneate the purpose of model checkers and automated theorem provers for this thesis.

The third chapter first gives an overview over related work on reasoning in general, before diving into three papers on logical reasoning in particular, that influenced the creation of

this thesis. This includes a discussion how our approach differs from these papers in terms of methodology. Secondly, we describe what benefits our approach might have compared to semantic parsing using neural networks. Third, we compare our approach to code generation (aka program synthesis).

The fourth chapter gives an overview over our methodology, including how we construct our datasets, how we define our tasks, how we evaluate our models (including the dataset sizes and splits) and which models we use (as well as how they differ). Chapter five concerns the experiments that we did regarding our research questions and the results we obtained. It also includes descriptive statistics of the dataset as well as experiments for constructing a baseline per task. Finally, in chapter six, we conclude our work and discuss opportunities for future research.

Chapter 2

Theoretical Background

The field of Artificial Intelligence (AI), a term first coined at the Dartmouth conference in 1956, has long been pursued by at least two competing paradigms: one symbolic (or logic- and rule-based) and one sub-symbolic (or statistics- and probability-based) (Garnelo and Shanahan, 2019). During that conference, Allen Newell and Herbert A. Simon presented Logic Theorist, an automated theorem prover for propositional logic (Russell and Norvig, 2010). Similar such deductive programs that used logical reasoning arose and this line of scientific enterprise had its first successes during much of the latter part of the last century.

However, disappointment followed suit in what came to be known as AI winter(s), which led to tremendous cuts in AI research funding: on the one hand, the successes of many early symbolic AI systems depended on the use of search trees, which suffer from combinatorial explosion. That is why more complex search trees could not be solved, even when using heuristics. Besides, rule-based approaches in machine translation did not lead to the promised results (Russell and Norvig, 2010).

On the other hand, the sub-symbolic paradigm did not fare much better, initially. The dawn of artificial neural networks can be traced back to Walter Pitts and Warren McCulloch's, who proposed the perceptron, which models a human neuron via propositional logic (McCulloch and Pitts, 1943). Unfortunately for the sub-symbolic paradigm, Marvin Minsky proved that a single perceptron, a single neural unit that does not have an non-linear activation function (as explained below), was not able to model the XOR (either-or) function, (that is, a function that checks whether only one of two inputs is true; which is a function that is not linearly separable). In other words, a perceptron was not capable of deciding whether two inputs were different from each other (Marvin and Seymour, 1969).

It was only due to the combination of three crucial developments over time, that neural networks and deep learning as we know them today, took off once again and ended the

AI winter(s). First, theoretical developments, such as backpropagation (Rumelhart et al., 1986) helped with training deeper networks, i.e. networks with more than a single layer. Second, there have been significant advancements in computing power with more specialized hardware to perform the mathematical operations required by deep neural networks. And finally, due to the building of the world wide web the amount of data and hence, corpora for training neural networks have grown substantially. The latter two aspects have enabled the recent rise of ever larger neural language models, ones that use more and more training data and require more and more parameters to be trained. LLMs in particular benefitted from this trend, because it has been shown that they get predictably better with increasing model size and training data (see (Hoffmann et al., 2022), (Kaplan et al., 2020)).

In this thesis, we teach LLMs reasoning using predicate logic. For this reason, the remainder of this chapter describes both LLMs and predicate logic, as well as all other necessary concepts to understand this thesis. Section 2.1 introduces neural networks, including plain vanilla feedforward neural networks all the way to transformer models. Afterwards (see 2.2), it will describe how language is (or word tokens are) represented in a machine and then show how the task of language modeling has been tackled with simpler n-gram and more advanced neural models, such as transformer models. Next, in 2.3 and 2.4, we describe the techniques (of LoRA adapters and quantization) that we use to finetune large language models under computational resource constraints. Fourth, in 2.5 we provide an overview over the syntax and semantics of predicate logic. Finally, we delineate the purpose of model checkers and automated theorem provers for this thesis (see 2.6.

2.1 Neural Networks and Deep Learning

2.1.1 Feedforward Neural Networks

There are many different kinds of neural networks. Let us start with the simplest kind, a vanilla, feedforward network (Jurafsky and Martin, 2022). These kinds of networks are composed of multiple neural units that only feed outputs from one layer to the next, i.e. there are no cycles in the data flow. In figure 2.1, one such neural unit is shown. It takes a set of inputs $x_1, ..., x_n$, multiplies and sums them with a set of weights $w_1, ..., w_n$, adds a bias term b and passes the result through an activation function, such as the sigmoid activation function σ (see figure 2.2).

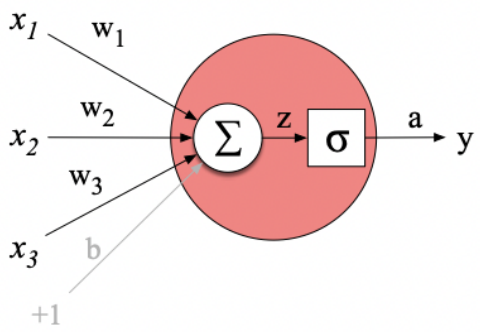


Figure 2.1: A neural unit (figure from (Jurafsky and Martin, 2022).

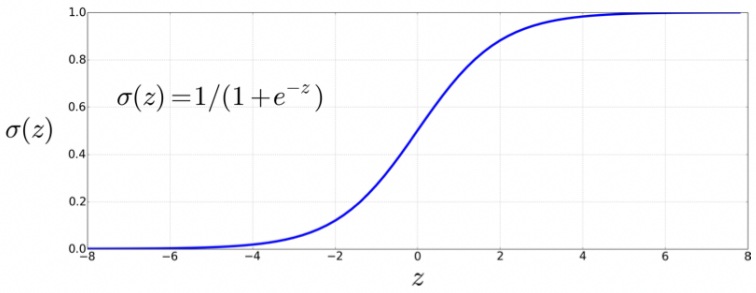


Figure 2.2: The sigmoid activation function (figure from (Jurafsky and Martin, 2022).

In vector notation, with y as the output of a single neural unit:

$$y = \sigma(\mathbf{w} * \mathbf{x} + b)$$

Next, we can combine these neural units to form a vanilla neural network, composed of neural units in multiple layers: an input layer, one or more hidden layers and an output layer, as depicted in figure 2.3.

Figure 2.3 shows how we input a vector $\mathbf{x}$ of scalar values, multiply these with weight matrices, such as $\mathbf{W}$ and $\mathbf{U}$ (a different one for each layer), add a bias term and pass the resulting term through an activation function f. So the hidden layer h is calculated by:

$$\mathbf{h} = f(\mathbf{W} * \mathbf{x} + \mathbf{b})$$

For the output layer, a similar such calculation is performed.

Going back to the small example of the XOR function from the introduction of this chapter, it was possible to show that a neural network with only two layers (one hidden and one output layer) was capable of computing the XOR function (Goodfellow et al.,

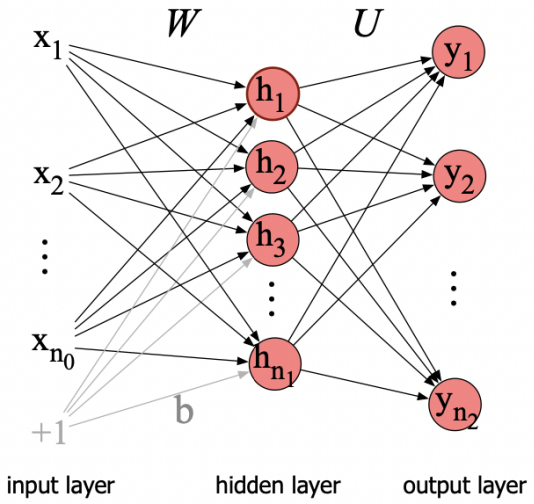


Figure 2.3: A feedfoward neural network (figure from (Jurafsky and Martin, 2022).

2016), an example that demonstrates the power of adding more layers and an activation function. While this is a very small neural network, deep neural networks are simply those that employ many hidden layers.

2.1.2 RNNs and LSTMs

Many different neural network architectures have been proposed over time; vanilla feedforward neural networks, as described above, are just one of them. Initially, those architectures were made specifically for the task at hand. For instance, Convolutional Neural Networks do well with image data. Recurrent Neural Networks (RNNs) work well with time series data, as they use a cycle such that each hidden layer activation depends not only on the current input (e.g. a word), but also on the hidden layer activation from the previous timestep (e.g. a word before the current word; and therefore also indirectly on the hidden layer activations of all previous timesteps (or words)). Therefore, these types of neural networks are suited to model the orderly nature of language as sequences of words (Mikolov et al., 2010).

However, it is not easy to train RNNs in a way that they take into account distant time steps (distant words). A different type of neural network, LSTMs (Long Short Term Memory networks), fare much better on long input sequences (Hochreiter and Schmidhuber, 1997). This is because the LSTM architecture includes gates which decide what information from past timesteps is still relevant for the current time step and what can be "forgotten".

2.1.3 The Transformer Architecture

The current state of the art architecture for language modeling is the transformer architecture (Vaswani et al., 2017). Just like LSTMs, the transformer architecture takes into account long-distance dependencies, but unlike LSTMs, they are not using recurrent connections (cycles), but self-attention. Because of self-attention, they are much easier to parallelize, and hence, scale better computationally, which is another reason why we have large language models today.

The following section explains the transformer architecture in more depth, including the central concepts of self-attention and positional encodings, as well as the architecture's most important building block, the transformer block.

2.1.3.1 Self-attention

The transformer block (see 2.4) is made up of multiple different layers, one of them being the self-attention layer. Self-attention is the key invention of the transformer architecture that helps with long-distance dependencies. In principle, it can take any distance into account.

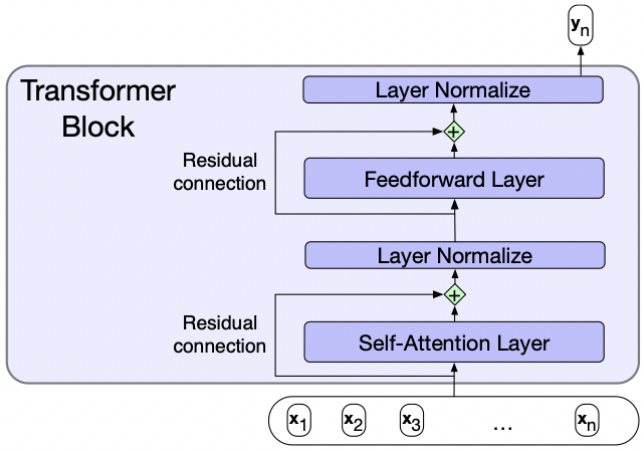


Figure 2.4: The transformer block as part of the transformer architecture (figure from (Jurafsky and Martin, 2022).

Figure 2.5 shows a self-attention layer: note how multiple inputs $\mathbf{x}_1$, ..., $\mathbf{x}_n$ (which are vectors representing e.g. word tokens) are mapped to output tokens of equal length ($\mathbf{y}_1$, ..., $\mathbf{y}_n$) by this layer: for the calculation of each $\mathbf{y}_i$, we only include inputs up to and including that time step i. In other words, inputs after the current time step i are not considered. Hence why it is also called a backward-looking (or causal) self-attention layer.

Furthermore, we can parallelize the passing through of the inputs in this way, as there are no recurrent or cyclic dependencies. Hence, we can parallelize the training of these networks, as compared to e.g. RNNs.

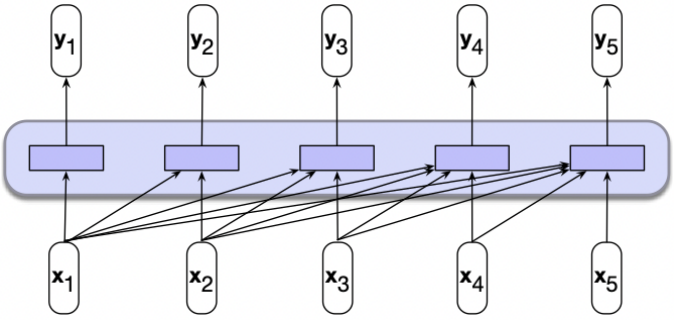


Figure 2.5: Self-attention as part of the transformer architecture (figure from (Jurafsky and Martin, 2022).

First, let us consider the purpose of self-attention before we look at the computations of a single neuron of the self-attention layer. The purpose of self-attention and the idea behind considering these long-distance dependencies is to compute a score of how relevant certain inputs are to certain other inputs. In terms of language modeling, we aim to produce a contextualized representation of words, that is, how relevant the previous words are to the word that is currently being attended to.

The way we achieve this can be seen in figure 2.6, where we zoom into one of the neurons of the self-attention layer. Here, we get a clearer picture of how one output vector (or word token) is calculated given an input sequence. First, we generate key, query and value vectors for each input token. The idea behind this approach is that an input (or a word) x_i can play three different roles when calculating a relevancy "score": as a query, it takes the role of being currently attended to by the inputs preceding it (that is, how much of the other words are relevant for this word); as a key, it takes the role of the preceding input (i.e. how much is this word relevant to the other words?); and as a value, it measures its role for the output (i.e. how relevant it is to the output).

So, for each of these three roles, there exist weight matrices $\mathbf{W^Q}$, $\mathbf{W^K}$ and $\mathbf{W^V}$, each of which is multiplied by each of the inputs $\mathbf{x}_i$ so that we get the key, query and value vectors:

$$\mathbf{k}_i = \mathbf{W^K}\mathbf{x}_i$$

$$\mathbf{q}_i = \mathbf{W^Q}\mathbf{x}_i$$

$$\mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Next, the so-called "key/query comparisons" from figure 2.6 refer to a multiplication of the respective $\mathbf{q}_i$ and $\mathbf{k}_i$ vectors. This gives a score between the vector (or word token) we currently focus on (which in figure 2.6 is $\mathbf{x}_3$) and each element preceding it. Those key/query comparisons are then passed through a softmax activation function. A softmax activation function is similar to the sigmoid activation function in that it extends the sigmoid activation function to multiple classes. Finally, a weighted sum between the result of that softmax calculation and the value vectors is created to get the output vector $\mathbf{y}_3$.

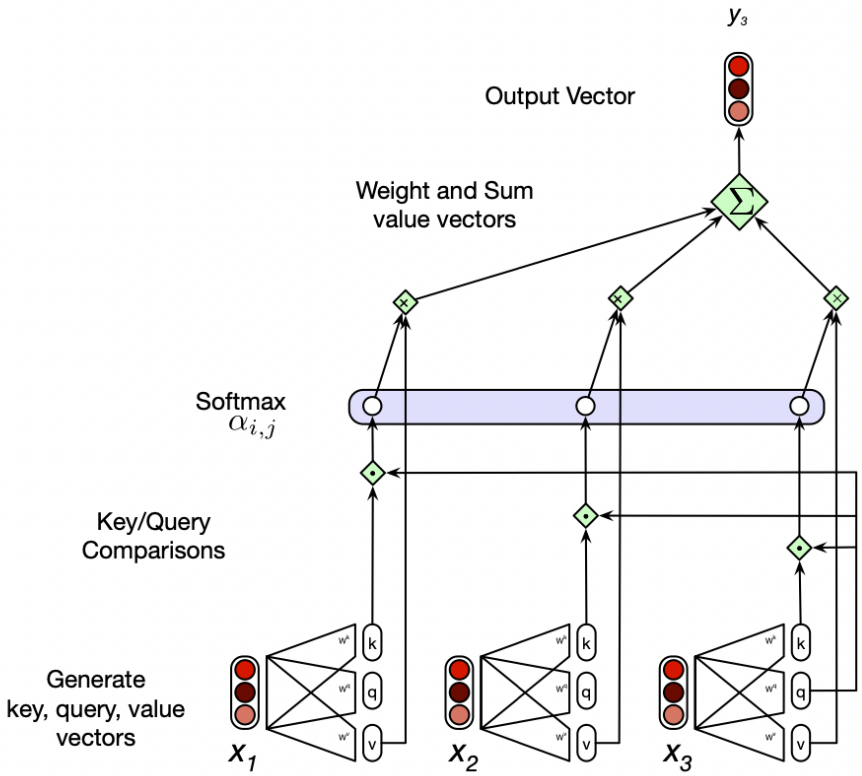


Figure 2.6: How one output token $\mathbf{y}_3$ is calculated given an input sequence $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ in one neuron of the self-attention layer (figure from (Jurafsky and Martin, 2022)).

As we stated before, we parallelize this process such that we calculate all outputs $\mathbf{y}_i$ of a sequence (of e.g. word tokens) at the same time. For this, we then have a matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, where each row is a vector representing one input token (so N is the number of input tokens and d the dimensionality of the vector or word token). We then get the matrices $\mathbf{Q} \in \mathbb{R}^{N \times d}$, $\mathbf{K} \in \mathbb{R}^{N \times d}$ and $\mathbf{V} \in \mathbb{R}^{N \times d}$ by multiplying $\mathbf{X}$ by the weight matrices $\mathbf{W}^{\mathbf{Q}}$, $\mathbf{W}^{\mathbf{K}}$ and $\mathbf{W}^{\mathbf{V}}$:

$$\mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

That is how we get the computation for the whole self-attention layer, which adds a scaling factor $\sqrt{d_k}$ that helps with numerical stability and gradient calculations:

$$\text{SelfAttention}\ (\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

This whole process so far is only a single self-attention "head". Usually, we use multiple attention heads. Multi-head attention refers to performing this same process many times in parallel. That is, using the same input tokens for each "head", but training different query, key and value weights independently. The intuition behind the reason for doing this (in language modeling) is that not only can a word play these different roles, but others as well, so it may need to "pay attention" to other words for different reasons, hence the multiple heads.

2.1.3.2 The transformer block

Other than the multi-head self-attention layer, a transformer block includes normalizing layers, feedforward layers and residual connections (see 2.4). We have already seen the feedforward layer in vanilla feedforward neural networks. Residual connections pass information from one layer to another bypassing a third layer. It has been shown that this improves learning (He et al., 2016).

Layer Normalization normalizes the sum of the vectors resulting from the residual connection and the respective (self-attention or feedforward) layer (Ba et al., 2016), i.e. it creates a vector with a mean of zero and a standard deviation of one. This also improves learning.

Finally, we stack transformer blocks or parts of it to create the transformer architecture. The first half of the transformer architecture, the so-called encoder is simply a stack of these transformer blocks such that a deep network is formed (see the left part of figure 2.7. The decoder (the right side of this figure) also includes elements from the transformer blocks. Finally, the last element needed to understand this architecture are input and output embeddings. What they are, will be explained in 2.2.1.

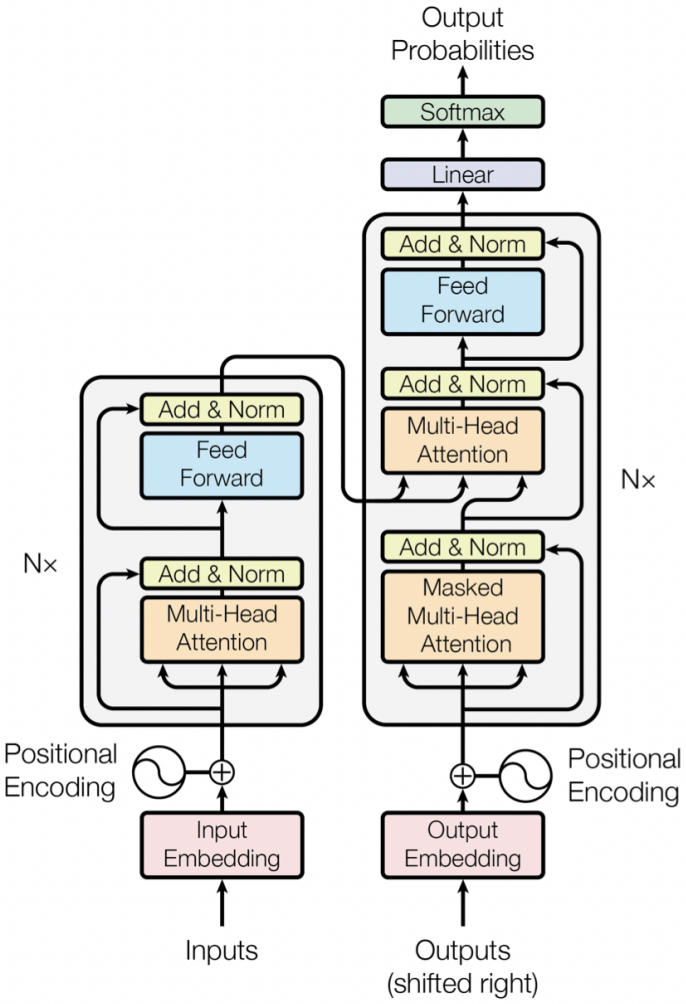


Figure 2.7: The transformer architecture (figure from (Vaswani et al., 2017).

2.1.3.3 Positional Encodings

So far as we have described it, the transformer architecture only encodes how different inputs (or words) are relevant to a word currently attended to. However, it is missing information about word-order. But word order is important, as can be illustrated by the following two sentences: "Bob is taller than Alice. Alice is smarter than Bob." Positional encodings (one for each word) add such information about word order for both the encoder and the decoder.

2.2 Language modeling

The previous sections gave a very general introduction into neural networks and the transformer architecture, as those models can be applied in many different domains, not just for modeling language. Nevertheless, this thesis is concerned with working with textual data, and therefore, we continue with how we model language, first, by describing how to represent the meanings of words, and secondly, by explaining different ways of predicting a sequence of words given previous words.

2.2.1 Word embeddings

There have been substantial discussions in both philosophy and linguistics as to what the meaning of a word is. In (Wittgenstein, 1953), the philosopher Ludwig Wittgenstein makes his case that the meaning of a word is decided by how it is used by people in the language (as opposed to being able to precisely logically define words by stating the necessary and sufficient conditions for the application of a word).

In a similar vein, how we represent the meaning of a word in computer science builds upon the distributional hypothesis from linguistics. This hypothesis states that words which are used in similar contexts tend to have similar meanings (Harris, 1954). A context in terms of machine learning is a certain number of words surrounding the word in question. By taking into account neighbouring words, we arrive at a distribution of which words occur in similar situations and therefore, which words have similar meanings. For example, the sequence "She pets the..." is likely followed by the word "dog", although it could also be followed by the word "cat". Therefore, the two words "dog" and "cat" are similar in a certain way in this context (but possibly different in other contexts). From this, we can represent a word by a vector, a point in high-dimensional space, where points that are closer to each other are more similar and points that are further apart are less similar (Li et al., 2015). Such a representation is called a word embedding.

On a high level, we distinguish static embeddings and dynamic embeddings (Jurafsky and Martin, 2022). In static embeddings, there exists one fixed vector for each word. Dynamic embeddings vary depending on the current context (i.e. the current neighbours) of the word.

There are many ways for creating word embeddings (i.e. for deciding where they lie in this high-dimensional space). For creating static embeddings, one can inspect the Word2Vec algorithm (Mikolov et al., 2013). Another way is to train a neural network to learn to predict next words from prior words (Collobert et al., 2011). One example of

dynamic embeddings, which are the de-facto standard today, has been introduced by the BERT language model (Devlin et al., 2018). Referring back to the self-attention layers of the transformer architecture, it is these layers that produce the dynamic, contextualized embeddings.

2.2.2 N-gram language models

First, let us define what language modeling is: language modeling is predicting the next word(s) from prior words. Generally, a probabilistic language model returns a probability distribution for the next word w_t given the previous $t-1$ words, which we represent by $w_{1:t-1}$ (See et al., 2019). Hence, we are looking to calculate the following probability distribution:

$$P(w_t|w_{1:t-1})$$

Word embeddings can help with predicting the next words, but it is a more advanced idea and the early forms of language modeling did not use those vector models. Rather, a simpler n-gram language model was proposed. An *n-gram* is a sequence of n contiguous words in a text. So for instance, a bigram is a sequence of two words (Jurafsky and Martin, 2022).

In order to define a probability distribution on the n-grams of a text, we start by extracting all the sequences of n consecutive words present in a training corpus and we attribute to each one a probability of occurrence based on the frequency of appearance of this very n-gram. So for instance, in a bigram language model, we approximate all the previous words by just the last word and the above equation becomes:

$$P(w_t|w_{1:t-1}) \approx P(w_t|w_{t-1})$$

In general, we would like to make n as large as possible in order to get a better approximation. However, even then, neural language models outperform n-gram language models by being able to use a much bigger n and to better predict the next word, since neural networks can use word embeddings instead of relative frequency counts.

2.2.3 Neural Language Models

While previous neural network architectures, such as feedforward neural networks (Bengio et al., 2003) and RNNs (Mikolov et al., 2011) can be used as language models, the transfomer architecture is currently the state of the art. There are many variants of the

transformer architecture that was described in 2.1.3. In this thesis, we work with two types of variants, causal language models (LMs) and encoder-decoder LMs. So in the following, we first explain the differences between them, before we give a high-level description of how to train a language model. Finally, we end with some notable details of current large language models.

Causal LMs (or auto-regressive LMs) only use the decoder part of the original transformer architecture (see figure 2.7). Examples of these types of models include GPT-2 (Radford et al., 2019) and those that are used in this thesis (refer to section 4.5 for this). Encoder-decoder LMs (or sequence-to-sequence LMs) use both the encoder and decoder of the original transformer architecture. Examples of these types of models include BART (Lewis et al., 2019) and Flan-Ul2, which we use in this thesis (refer to section 4.5 for this).

When we train a causal language model, we feed input sequences into it and receive as output an equally long sequence, aiming for the language model to output the correct next word (see figure 2.8). Here, the softmax function outputs a probability distribution over all words of a pre-defined vocabulary and the word with the highest probability is chosen for the next word.

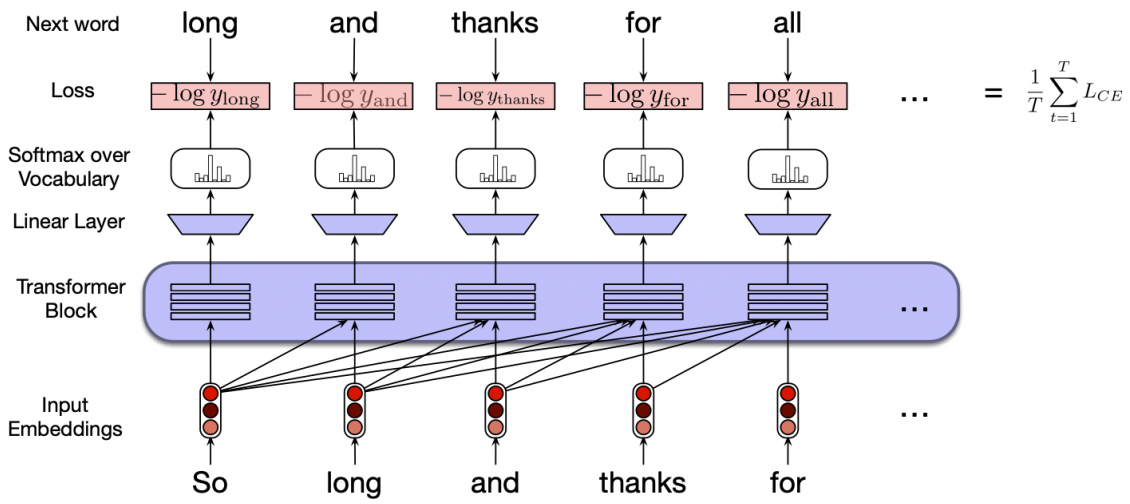


Figure 2.8: Training in the transformer architecture (image by (Jurafsky and Martin, 2022).

After feeding the inputs to the language model and receiving predicted word tokens, we compare the actual outputs (the predictions) with the desired outputs (the target word tokens) by computing the cross-entropy loss. Let $\mathbf{y}_t$ be the next token in a sequence (the target token) and $\hat{\mathbf{y}}_t$ the actual output by the neural network, then the cross-entropy loss

is defined as:

$$L_{CE}\left(\hat{\mathbf{y}}_t, \mathbf{y}_t\right) = -\log \hat{\mathbf{y}}_t\left[w_{t+1}\right]$$

The algorithm of gradient descent then tells us how to minimize this loss function, and the algorithm of backpropagation updates the weight and bias parameters step by step such that we minimize the loss function (Rumelhart et al., 1986). By minimizing the loss function, we aim to find optimal parameters (weights and biases) for our language model, such that it produces the desired word tokens.

Fortunately, we do not even need hand-crafted labels for training neural language models (Bengio et al., 2003). In machine learning, we differentiate between supervised and self-supervised learning. In supervised learning, someone has to decide upon the labels (the correct outputs) given certain inputs. In self-supervised learning, the language model simply takes the next word as the label, so any input text is automatically labelled and hand-labeling is not necessary. Self-supervised learning is how we can use huge text corpora to train large language models.

After a language model is trained, it can be used for text generation. In order to generate text, different so-called decoding strategies can be used. Greedy decoding means to generate one (word) token $\hat{\mathbf{y}}_t$ at a time by choosing the word with the highest probability from the vocabulary V of words (Jurafsky and Martin, 2022):

$$\hat{\mathbf{y}}_t = argmax_{\mathrm{w}\in\mathrm{V}} P\left(\mathbf{w} \mid \mathbf{y}_1, \ldots, \mathbf{y}_{t-1}\right)$$

Other decoding strategies involve searching and sampling multiple sequences of predictions, incorporating at each step e.g. the most probable words, and finally returning the most probable sequence. Examples are beam search (Shao et al., 2017), top-k sampling (Fan et al., 2018) or nucleus sampling (Holtzman et al., 2019).

2.3 Parameter-Efficient Fine-Tuning

In order to obtain better generations, language models have grown bigger and bigger in their number of parameters, as to what we now call large language models. For example, one of the biggest models in existence today, PaLM has 540 billion trainable parameters (Tay et al., 2022b). Parameters are the weight matrices and the biases; they are what changes during training.

2. THEORETICAL BACKGROUND

Next to having grown in size, large language models have been trained on more and more tokens of text, ranging into multiple billions as of today. Different existing language models have been trained on different corpora of text, with the biggest language models in existence today using mostly data scraped from the world wide web.

The huge amounts of data combined with the enormous amounts of trainable parameters of large language models make it computationally very resource intensive to train large language models. That is why in this thesis, we use pre-trained models, i.e. ones that have already been trained on large amounts of text. Those already have a representation of word (or sentence) meaning and can already produce meaningful sentences during generation. It has even been demonstrated that large pre-trained language models can solve tasks in a zero-shot (Wei et al., 2021) and few-shot setting (Brown et al., 2020), i.e. without getting any examples or just a few examples as additional input as a demonstration of a task.

Connected to pre-training is the notion of fine-tuning. A pre-trained language model is typically fine-tuned, i.e. trained further on a different training objective than language modeling, that is, on a downstream task, such as translation, summarization or logical reasoning. However, fine-tuning a large language model also needs a lot of computational resources (albeit typically less than pre-training), as we would still need to update all of the (typically billions of) parameters. Besides, fine-tuning the full large language model means that one also needs to store (all of the changed weights of) the model, and consequently, storing multiple such models for comparison takes up a significant amount of space. In other words, conventional fine-tuning is parameter **in**efficient, as it updates all parameters of such a model.

That is why, in this thesis, we do not fine-tune the pre-trained models, but instead use adapter-based parameter-efficient fine-tuning (PEFT). Generally, adapter-based PEFT, or "using adapters" to efficiently fine-tune large language models, refers to keeping the pre-trained model weights fixed and only training a very small number of (additional) parameters (i.e. training an adapter) compared to training the large number of parameters of a large language model, and then using these parameters (the adapter) together with the (entire) pre-trained large language model, in order to perform a task. In other words, a small "adapter" of parameters "adapts" or modulates the parameters of the pre-trained language model to produce desired results (e.g. on a downstream task). It has been shown that this can achieve near state-of-the-art results on a downstream task (Houlsby et al., 2019).

The major advantage of PEFT methods lies in needing much less GPU resources to achieve these results. Also, adapters are much smaller in size compared to large language

models, so we can easily store many adapters. Furthermore, adapters have the advantage that they are not prone to catastrophic forgetting, which refers to a language model forgetting how to solve the task that it was first trained one, once it is fine-tuned on another task (Goodfellow et al., 2013).

One such class of adapters (and the one we use) is called low-rank adapters (LoRA) (Hu et al., 2021). In order to understand LoRA, let us review fine-tuning a bit more formally. In fine-tuning, just like in training, our aim is to update the weights (let us denote them by **W**), such that we get a new set of weights **W'**, so:

$$\mathbf{W'} = \mathbf{W} + \Delta\mathbf{W}$$

Here, $\Delta\mathbf{W}$ represents the change in **W**. This formula is recomputed multiple (i) times during gradient descent, in order to attain values for **W'** that can fulfill the task that we are interested in solving with the language model. We usually discard these i weight matrix updates $\Delta_i\mathbf{W}$.

In essence, LoRA takes the view that we do not need discard those weight matrix updates, but rather store them separately from **W** and merging them with **W** when appropriate (e.g. during generation). This way, we update only $\Delta\mathbf{W}$ and keep **W** the same. One way to depict this, is in figure 2.9, where we keep the pre-trained weights **W** fixed and only train the weight updates $\Delta\mathbf{W}$, represented here by A and B.

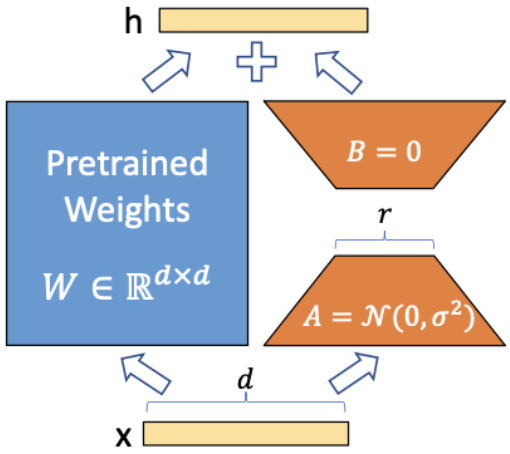


Figure 2.9: Low-rank adapters (image by (Hu et al., 2021).

The crucial add-on to this is that we can reduce the dimensionality of $\Delta\mathbf{W}$ (which has dimension $d \times k$, just like **W**) by decomposing it into two matrices A (with dimension

$r \times k$) and B (with dimension $d \times r$). We call this low-rank approximation, because A and B approximate $\Delta\mathbf{W}$ by a parameter r.

$$\Delta\mathbf{W} \approx B * A$$

We call r the rank of the matrix $\Delta\mathbf{W}$, and r is a hyperparameter, i.e. a parameter that we as experimenters change to see which value works best. The smaller r, the smaller the matrices A and B that we need to store, and in general $r << min(d, k)$. We can think of r analogously to a reduced number of singular values in a singular value decomposition, those that are considered sufficient to approximate the matrix represented by the singular value decomposition. Compare the matrices A and B to a truncated singular value decomposition (SVD) of $\Delta\mathbf{W}$: by truncating the smallest singular values, we shrink the respective matrices from SVD and get an approximation. Importantly, using small values of r means that LoRA adapters are much smaller in size than the pre-trained model weights. Note that we need to train the matrices A and B from scratch, so we initialize A from a normal distribution and B as a zero-matrix.

One reason why LoRA works is that it LoRA builds upon the finding that many pre-trained large language models have low intrinsic dimension (Aghajanyan et al., 2020). That means that the same language model could be described by a lot less weights than they have, i.e. there is a lot of redundancy in these LLMs. The LoRA article extends this by hypothesizing that the weights have low intrinsic rank and that it is therefore possible to use the above approximation (Hu et al., 2021). It concludes, that we can usually perform adapter-based fine-tuning instead of conventional, full fine-tuning and get comparable results.

So far, we have not specified, yet, what we mean by $\mathbf{W}$ with regards to the specific transformer architecture. In principle, we could apply LoRA to any of the weight matrices in the architecture of our large language model in question. That is why, to answer this question, we refer the reader to section 6.2 as to the design decisions we made with regards to LoRA, including the hyperparameter decisions. As a side note, next to r another hyperparameter of LoRA adapters is the lora α, which scales how much of the update $\Delta\mathbf{W}$ we want to incorporate into our pre-trained model.

2.4 Quantization

We can further speed up the calculations when training a neural network, and further reduce GPU use, and that is by quantization. Conventionally, i.e. without using quanti-

zation, we use 32-bit-precision when encoding our matrix elements for training (or 16-bit precision when using LoRA). By quantizing the parameters of a neural network, we approximate them by reducing their precision to e.g. four or eight bit. Therefore, quantization compresses an LLM. However, we need to be aware that quantization can negatively affect the accuracy of the language model, since we reduce the precision of the parameters.

Finally, we can combine LoRA and quantization into quantized LoRA, or QLoRA (Dettmers et al., 2023). This means that we quantize the parameters of the LLM and then train using the LoRA adapter method. The disadvantage of only using an adapter like LoRA is that we would still need a large GPU to load the pre-trained model weights plus the LoRA adapter. With QLoRA, on the other hand, we have much smaller memory requirements and can therefore load even bigger models. In more detail, the QLoRA introduces a new data type (4-bit Normal Float) to achieve this, and also proposes to perform double quantization, that is quantizing the quantization itself. What is more, while QLoRA reduces the memory usage when finetuning a LLM, it does so, according to a variety of experiments, without trading off performance compared to finetuning models with 16-bit precision (Dettmers et al., 2023).

2.5 First-order (Predicate) Logic

Logic, in essence, is the theory of reasoning (Gamut, 1991). As such, it includes a theory of formally valid conclusions. That is, in a logically valid argument, the truth of the premises guarantees the truth of the conclusion. In this thesis, we concern ourselves with a certain class of logic: first-order (predicate) logic, as initially developed by Gottlob Frege (Frege, 1879) and Charles Sanders Peirce (Hammer, 1998).

When describing any kind of logic, we need to define both a syntax and semantics. The syntax gives definitions of symbols and rules about what kinds of combinations of symbols into expressions constitute well-formed expressions, i.e. expressions that we allow to be formed. The semantics describes what the expressions mean, that is, it concerns itself with the interpretation of this language.

2.5.1 Syntax

The syntax of predicate logic is made up of an alphabet, i.e. a set of symbols that make up this formal, artificial language, as well as formation rules, i.e. rules that allow us to form well-formed expressions. As a consequence, using symbols that are not in the alphabet to

form expressions, or not abiding by the formation rules leads to expressions that are not well-formed.

The alphabet with which we concern ourselves in this thesis includes: quantifier symbols (quantifiers), logical connectives (operators), variables, predicate symbols (predicates), constant symbols (constants) and parentheses and punctuation symbols. Different notations exist for these symbols, so let us establish one notation in table 2.1, which also states all the symbols of our alphabet and closely follows the syntax used in the nltk representation of predicate logic (i.e. the python library that we use in this thesis) (Bird et al., 2009).

Table 2.1: A list of symbols from predicate logic.

Symbol class	Symbol name	Symbol notation
Quantifiers	Universal quantifier	all
	Existential quantifier	$exists$
Operators	Negation	$-$
	Conjunction	&
	Disjunction	|
	Implication	$\rightarrow$
	Biconditional	$\leftrightarrow$
Variables	Variable	x, y, z
Predicates	Predicate	$F, G, H, ...$
Constants	Constant	$a, b, c, ..., w$
Parentheses	Parentheses	( )
Punctuation	Dot	.
	Comma	,

For the variables, we use lowercase letters towards the end of the alphabet. In our case, only a few such variables suffice (x,y,z). As regards the predicates, we use uppercase letters from the alphabet (although we will not need the whole alphabet for our purposes). For the constants, we use lowercase letters from the beginning of the alphabet. With regards to punctuation and parentheses, we use a dot "." and a comma "," as well as round brackets "(" and ")".

Predicates have an arity n, that is, they take a number $n >= 0$ of variables as arguments. We exclude the case of $n = 0$ here. For $n = 1$, one example could be $F(x)$, that is the predicate F applies to the constant x. For $n = 2$, one example could be $G(x, y)$, that is x stands in the G-relation to y. While this remains an abstract explanation, in section 2.5.2

some examples are provided, what this could mean.

We can now inductively define the set of well-formed formulas (wffs) using this alphabet. The rules for forming this set are as follows:

1. Every P that is an n-ary predicate, followed by either n variables or n constants within parentheses, where $n > 0$, is a wff.
2. If ϕ is a wff, then $-\phi$ is also a wff.
3. If ϕ and ψ are both wffs, then $\phi \,\&\, \psi$, $\phi \,|\, \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$ are also wffs.
4. If ϕ is a wff and x is a variable, then $all\, x.\phi$ and $exists\, x.\phi$ are wffs.

Therefore, applying these rules finitely many times gives us the set of well-formed formulas.

Examples of such formulas are included in the following list. Note the use of parentheses to tell us which variables belong to which predicates, as well as the use of dots and parentheses to tell us which variables belong to which quantifiers:

- $P(x)$
- $all\, x.A(x)$
- $exists\, y.Q(y)$
- $all\, x.exists\, y.(B(y) \,\&\, -A(x))$
- $all\, x.exists\, y.(B(y) \,\&\, (-A(x) \rightarrow B(x)))$
- ...

Besides, parentheses are also important to denote operator precedence, as can be seen in the last example, although conventions for operator precedence have been developed so that we can exclude certain parentheses just like we do with brackets in multiplication and addition. The convention for the order of precedence is as follows: negation is evaluated first, then conjunction and disjunction, quantifiers are next, and implication and biconditionals are last.

2.5.2 Semantics

The semantics of predicate logic concerns itself with the interpretation the syntactic elements. An interpretation is a function which maps constants to objects in some universe of discourse, predicates to properties of objects and well-formed formulas to truth values. Here, the domain of discourse can be seen as a set of objects that we make statements about. For example, in figure 2.10, we make statements about the set of humans and the set of dogs, and in particular about one specific dog represented by the constant q and one specific human represented by the constant b. We assign truth values (T for true and F for false) depending on whether q or b belong to the set of humans or dogs. In our example, q is a dog (the predicate D applies to q) but not a human, and b is a human (the predicate H applies to b). Predicates of higher arity refer to relations between objects. For instance, the predicate T could refer to "being taller than", such that $T(b, q)$ means that "b is taller than q".

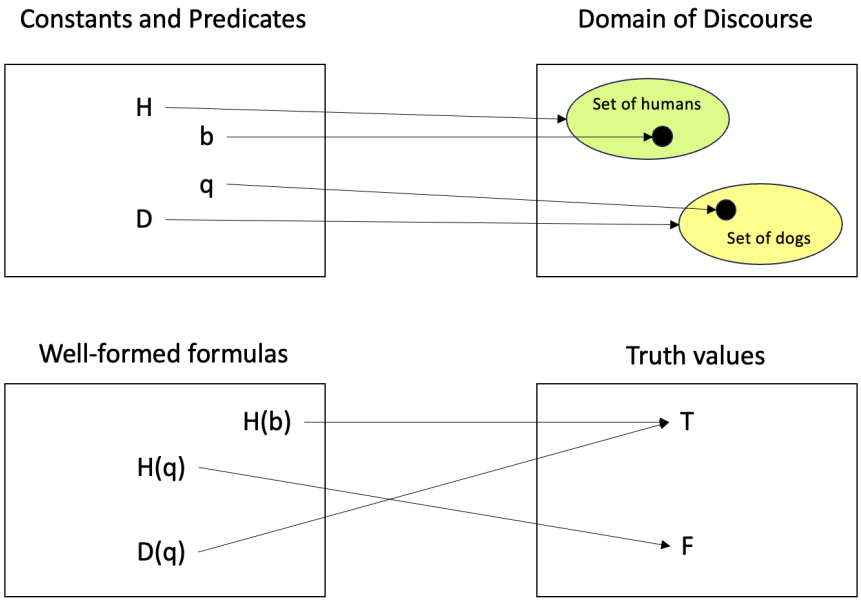


Figure 2.10: An example of an interpretation.

Another example could be the formula $exists\ y.H(y)$, which would state that there exists at least one object in our domain of discourse (in figure 2.10) which is a human. This formula could either be true or false, and in our small domain of discourse, it is true. We see this by finding at least one object in our domain of discourse that is human - in our case, that is b.

More generally, let us define when certain formulas evaluate to true or false:

- An n-ary predicate F evaluates to true, if for all n constants in the domain of discourse the relation F holds true.
- $-\phi$ is only true, if ϕ is false.
- ϕ & ψ is only true, if ϕ and ψ are true.
- $\phi \mid \psi$ is only false, if both ϕ and ψ are false.
- $\phi \rightarrow \psi$ is only false, if *phi* is true and *psi* is false.
- $\phi \leftrightarrow \psi$ is only true, if both ψ and *phi* evaluate to the same truth value (i.e. if either both are true or both are false).
- $exists\ x.\phi$ is true if and only if there is at least one constant α which makes ϕ evaluate to true.
- $all\ x.\phi$ is true if and only if every possible constant α makes ϕ evaluate to true.

Note that we do not include the XOR operator from the introduction to this chapter, but this logical operator is true, if either one of ψ or *phi* is true, but not both at the same time, i.e. only when they have different truth values. Finally, one note as to how we speak about predicate logic: we say that a well-formed formula ϕ is satisfiable, if there exists some interpretation under which it evaluates to true.

2.6 Model Checkers & Automated Theorem Provers

In order to reach the conclusion, whether a well-formed formula is satisfiable, rules of inference have been invented, that we can mechanically apply by hand, one after another to decide that question (Gamut, 1991). However, in this thesis, we do not ourselves determine whether a formula is satisfiable, but rather make use of a model checker.

Model checkers (are computer programs that) take as input a well-formed formula and a model, where a model is set of statements about our universe of discourse plus an interpretation function that maps predicates and constants as well as truth values (deciding which statements are true and which are false). Below, we refer to the set of statements as the "world model" and the mapping as the key mapping.

Model checkers return whether a formula is satisfied given a model. Remember Logic Theorist, one of the earliest automated theorem provers, from the introduction of this

chapter. Contrary to model checkers, automated theorem provers check the validity of a formula for all possible models (Loveland, 1986). We also make use of an automated theorem prover in our experiments, in order to detect tautologies and contradictions. Tautologies are formulas that are valid given any model. Contradictions are formulas that are invalid given any model.

Chapter 3

Related Work

There is an ever-increasing literature on probing language models and teaching them reasoning, as this area can still be seen as an unsolved problem (Helwe et al., 2021), where ever-harder tasks have been developed. We follow this tradition of developing such tasks. Many different kinds of reasoning tasks and benchmarks have been developed, including but not limited to mathematical reasoning, commonsense reasoning and logical reasoning. In mathematical reasoning, for instance, tasks range from performing function calculations involving multiplication and addition (Saxton et al., 2019), to providing mathematical proofs (Li et al., 2020). Commonsense reasoning refers to a certain background knowledge that humans commonly have about typical circumstances they find themselves in, such as "rain makes the road wet" (see e.g. (Liu et al., 2021)). The most related to what we do in this thesis is the overarching area of teaching LLMs logical reasoning. In the following sections, we first give a brief overview of related articles, how they are similar in terms of methodology and goals, and crucially, also where they differ.

3.1 Logical Reasoning

There already exists a wide array of logical reasoning tasks for neural (language) models, such as deductive reasoning (Clark et al., 2020), detecting logical fallacies (Jin et al., 2022), abductive reasoning (Young et al., 2022), answering analytical questions of admissions tests, such as the law school admissions test from the United States (Zhong et al., 2021), asking a language model to provide logical proofs similar to an automated theorem prover (see e.g. (Saha et al., 2020) and (Tafjord et al., 2020)), or even mapping neurons directly to elements of a logic formula (Riegel et al., 2020); (see also (Helwe et al., 2022) for additional research directions into this subfield).

In the following, we focus on three papers in particular, that are closest to what we do: the first one is about RuleTakers, experimenting whether language models can make deductive inferences based on rules and premises (Clark et al., 2020). The second paper is about RuleReasoner, testing whether language models can perform boolean satisfiability problems (Richardson and Sabharwal, 2022). And the third paper is by Betz, Voigt and Richardson (Betz et al., 2020), who ask language models to perform inference in natural language based on predicate logic.

3.1.1 RuleTakers

Clark et. al (Clark et al., 2020) train language models to become "soft theorem provers" and call them RuleTakers, as they are fed rules (that is, generalized conditionals of the form: "condition (& condition) $\rightarrow$ condition") and facts (both in natural language) and then respond to binary true-false questions regarding these inputs (see figure 3.1). This is quite comparable to our "deductive sat task" (task three) (see chapter 4), as in this task, we also ask the language models to provide a binary answer (whether a formula is satisfied or unsatisfied given a formula and a world model). Here, a formula in predicate logic from our task can be seen as a rule from the RuleTakers paper, and a world model from our task can be seen as the facts from the RuleTakers paper.

While we also aim for a language model to emulate a reasoning algorithm, our approach and theirs differ in that we do not bypass a formal logical representation. While their input to the language model is completely in natural language, our input includes both natural language and the symbols of the artifical language of predicate logic. Therefore, our task makes for a much harder task, as the model needs to not only understand natural language and make inferences based on it, but it is also asked to understand the syntax and semantics of predicate logic.

Moreover, instead of only teaching language models to deduce novel facts from rules and facts (i.e. to merely apply rules of logic), in our "create world model task" (task two), we ask the model to come up with a world model that either satisfies or dissatisfies a formula. In other words, we expect it to produce an intermediate step, before verifying that a given formula is satisfied given the language model's generation. Finally, we also consider the reverse case (an inductive case) in our "create formula task" (task one), where the language model has to come up with a formula given a world model (i.e. to create abstractions).

(Input Facts:) Alan is blue. Alan is rough. Alan is young.
Bob is big. Bob is round.
Charlie is big. Charlie is blue. Charlie is green.
Dave is green. Dave is rough.

(Input Rules:) Big people are rough.
If someone is young and round then they are kind.
If someone is round and big then they are blue.
All rough people are green.

Q1: Bob is green. True/false? **[Answer: T]**
Q2: Bob is kind. True/false? **[F]**
Q3: Dave is blue. True/false? **[F]**

Figure 3.1: Example of logical reasoning task by and figure from (Clark et al., 2020).

3.1.2 RuleReasoner: Solving SAT problems with LMs

In (Richardson and Sabharwal, 2022), the authors test whether neural language models can perform boolean satisfiability (SAT) problems in natural language. A boolean satisfiability problem asks, given a boolean formula, whether there is an interpretation that satisfies this formula. That is, for a formula of boolean logic, it asks whether we can replace the variables with either true or false values, such that the formula is satisfied. If so, then the formula is satisfiable.

They construct natural language theories based on these formulas (which, if we use the RuleTaker terminology, include both rules and facts) and ask the language model whether such natural language theories are satisfiable (or not), plus whether they are satisfiable if we add a conjecture (which can be seen as a hypothesis) (see figure 3.2). This depends on whether the conjecture is entailed by the natural language theory or whether it contradicts it. For this, they use a SAT solver, i.e. a program that can automatically label their datasets.

NL Theory	$\Gamma = \{$ *Bob is round. Alan is blue, rough and young*. If someone is round then they are big. All rough people are green. Big people are not green. $\}$
Conjectures	**1.** Alan is green (**entailment**, $\Gamma \models$ **1**) **2.** Bob is green (**contradiction**)
Satisfiability	Γ has an interpretation (**sat**) $\Gamma \cup \{\neg$**1**$\}$ (**unsat**), *indirectly proves* $\Gamma \models$ **1** $\Gamma \cup \{$**2**$\}$ (**unsat**), *indirectly proves* $\Gamma \models \neg$**2**

Figure 3.2: Example of logical reasoning task by and figure from (Richardson and Sabharwal, 2022).

They create a dataset that pays particular attention to creating hard training instances, that is not creating datapoints with a certain very high or very low clause-variable ratio (a clause is a disjunction of literals and a literal is a variable or negated variable) (Vizel et al., 2015). They empirically show that very high or very low values of this ratio correlate with a very high or very low SAT probabilities, which could be used implicitly by the language model to infer the satisfiability without learning to emulate logical reasoning. Thereby, they can not only distinguish between easy and hard problems, but also systematically increase the complexity of their task and show that training on hard problems is beneficial for language model outputs.

Similar to RuleReasoner, we also make use of automatical labeling of our dataset as well as developing a methodology for systematically increasing the difficulty of our datasets. Contrary to their paper, ours differs in three ways: First, the expected outputs of our models differ. While RuleReasoner asks the language models whether there is some way to satisfy the given input (by changing its truth values), the closest that one of our task comes to this, is task two (create world model), where we ask the language to generate in natural language one specific set of (true) "conjectures" (to stay in the RuleReasoner terminology) that makes a given input (the formula) true. Our task three ("deduce sat") also differs in that there, we expect the output to be whether a formula is *satisfied* (instead of *satisfiable*) given the inputs.

Second, in contrast to them, we do not just give the language model natural language sequences as input, but instead natural language combined with the artificial language of predicate logic.

Third, and most importantly, Richardson and Sabharwal make language models perform a task in propositional logic (Richardson and Sabharwal, 2022), and given that our dataset contains formulas in predicate logic, we arguably provide a more difficult task, since propositional logic is contained within predicate logic and predicate logic is more expressive than propositional logic (using existential and universal quantifiers).

3.1.3 Predicate Logic Inference

In the article by Betz et al. (2020), the authors create a synthetic corpus of eight different types of argument schemes in predicate logic, mapping them to natural language and training and evaluating GPT-2 with it. The argument schemes are depicted in figure 3.3, and an example mapping to natural language for the negation variant of the generalized modus ponens is:

> "Here comes a perfectly valid argument: To begin with, Susan is a friend of Chloe. Moreover, no sister of Lisa is a friend of Chloe. In consequence, it is false that Susan is a sister of Lisa." (see p. 6 of Betz et al. (2020)).

This is what the language model receives as input and it is asked a binary question of whether the argument is deductively valid or not. Note that this is one of the datasets that are part of the benchmark BIG Bench (Srivastava et al., 2022).

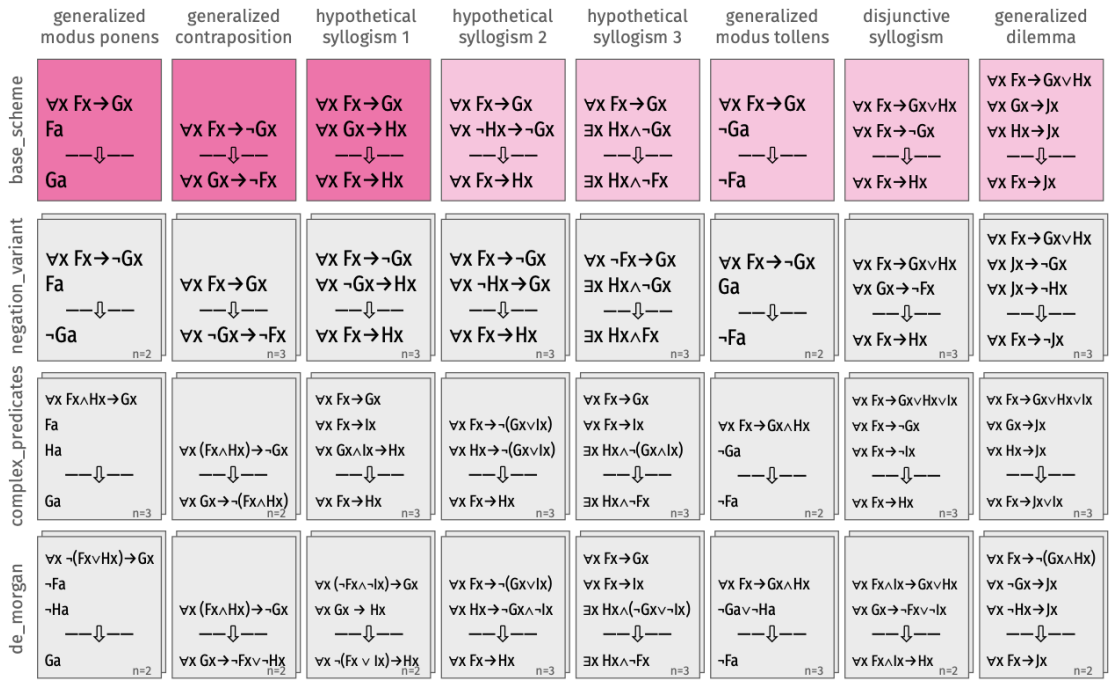


Figure 3.3: Logical reasoning task by and figure from Betz et al. (2020).

While we also make use of predicate logic, we extend their approach of modeling eight different types of argument schemes (and their variants, leading to 32 argument types), to a possibly infinite number of argument types. Plus, we build these arguments fully automatically and have both built and modified existing parsers to automatically label our datasets - while in their paper, they vary the argument schemes manually. Nevertheless, some of their argument schemes are not present in our dataset: those that deduce formulas from formulas (in natural language).

Besides, all of our tasks, once again, make use of both natural language and the language of predicate logic (as input to the language models) - something which they did not do in

their paper, as their inputs were in natural language only.

Plus, only one of our tasks concerns a binary question (satisfiable or not). The other tasks include asking the model to produce statements (task two) or formulas (task one) itself. So, we do not only ask our model to make deductions, but also to produce statements that are then checked whether they satisfy a formula. Finally, we train a multitude of SOTA large language models, that are much larger than GPT-2.

In summary, what all of the previous works on fine-tuning language models to reason logically did was to create a synthetic dataset, just as we do. However, our work distinguishes itself through the following aspects: first, since we are able to model any predicate logic formula with finite length, in principle, we can make our dataset harder and harder. Second, we do not only consider deductive abilities of the LM, but also the inductive (or abstractive) abilities. Third, we ask the language models to reason not just in natural language, but also in the language of predicate logic.

Next to these related papers on logical reasoning, there are two more (albeit rather tangentially related) topics to our thesis: semantic parsing and program synthesis.

3.2 Semantic Parsing

Semantic parsing takes natural language input and translates it into a logical form, such that a formal reasoning system (such as a model checker or automated theorem prover) can be applied afterwards (Kamath and Das, 2018). One can distinguish rule-based from neural approaches for accomplishing this task, both of which are prone to errors (Saha et al., 2020).

In this thesis, we perform the overarching goal to have the machine reason formally given natural language input in an end-to-end fashion: we let the language model directly generate the desired formal reasoning. By doing this, we can show that language models are capable of working with both the natural language and the artificial language of logic at the same time. In this sense, although our main goal is to have language models reason (in both natural language and predicate logic), one useful byproduct of this research could be a functioning neural semantic parser for predicate logic.

Besides, in this thesis, we do not fully bypass semantic parsers, but build and modify (a rule-based version of) them, that we use in order to map the (postprocessed) LLMs' generations to their respective required representation (of the nltk model checker or automated theorem prover). Postprocessing (in the form of extracting the desired part of the output)

in our case is usually necessary, as LLMs tend to "overshoot", that is, they produce more output than what is desired: for instance, they explain their answers after giving them.

3.3 Program Synthesis

Finally, our task is related to some degree to the task of program synthesis. Program synthesis is the task that takes as input a problem description (in natural language) for a program (something that one would like to have solved in code) and outputs a coded program, i.e. a solution in the (artificial) coding language (Le et al., 2022). In this task, similar to our task, a language model that generates code also needs to translate between a sequence in natural language and a sequence in a formal language, the programming language. Programming languages are at their core logic languages, so there might be a link to predicate logic. Besides, when translating from natural language to a programming language, the neural language model also needs to produce a correct mapping, and that it can only do by having some higher level abstraction of the rules that make the program (and analogously, the logical argument) "work". Plus, it needs to keep track of variables that reoccur, which is akin to keeping track of variables in logic.

Chapter 4

Methodology

Hand-labeling a dataset for supervised learning is a time-consuming and/or expensive process. This has meant that hand-labeled data sets have remained relatively small compared to the number of data points that large language models like PaLM have been pre-trained on (Tay et al., 2022b) using self-supervised learning. This increase in the size of data sets, however, has been one reason behind the success of large language models. This again has been due to the finding that self-supervised learning works for language modeling (Bengio et al., 2003).

Nevertheless, the way this has been done for pre-trained models currently, also has its drawbacks. Generally, although the exact data sets are usually not open-sourced by the creators of the currently available large language models, it consists in its majority of preprocessed data from the world wide web (e.g. from CommonCrawl (Touvron et al., 2023)), plus some additional curated corpora. While that way of training has lead to the remarkable achievement of large language models being very good at predicting next words into fluent, coherent and meaningful sentences, it seems to have reached a qualitative ceiling: It is suggested that this way of "dumping" data into large language models leads to all sorts of biases (see e.g. (Liang et al., 2021) and (Ouyang et al., 2022)) as well as hallucination (see e.g. (McKenna et al., 2023) and (Mündler et al., 2023)), which refers to giving incorrect information as if it were a fact.

One way to account for these shortcomings of large language models is to use reinforcement learning from human feedback (RLHF) (Ziegler et al., 2019). RLHF can be used to finetune language models by teaching it which of two generated sentences a human being prefers. Once again, as in hand-labeling, however, human labelers are needed. Furthermore, it has been shown that aligning language models to human preferences, using processes such as RLHF, can be reversed (Wolf et al., 2023). This suggests that these

alignment processes only weaken the probability of undesired behavior, but cannot remove them altogether.

If we could instead develop synthetic datasets (in an automatic fashion), that are automatically labelled and that are examined to be free from biases, then we could not only generate datasets of any desirable size (and save the expense and time of human labeling). It would also be possible to expand this approach to "RLMF" or reinforcement learning from machine feedback, where an automatic theorem prover or model checker tells the language model when it is correct and when not. Instead of a (fallible) human being who also might introduce his/her own biases by judging what he/she prefers, we could use RLMF together with such synthetic datasets, as our SPLC (synthetic predicate logic corpus). This, we hypothesize, could lead to the language model acquiring the skill of logical argumentation (a core skill in critical thinking), rather than learning to constrain its outputs to what a human being wants it to state. It could elaborate on the premises it believes to be true and construct valid arguments based on them. Instead of submitting to human judgement, it could form its own judgements.

The dataset that we propose provides a crucial part that could be further developed to RLMF: we create a synthetic dataset for predicate logic that can be automatically labelled, and it can even automatically evaluate whether the language model output is correct using a model checker. In this thesis, we train via supervised finetuning, but an extension to RLMF is conceivable and will be further discussed in chapter 6.

In the first part of this chapter, we give an overview of how we designed our system. Next, we algorithmically describe how our dataset is constructed. Third, we precisely define each of the tasks that we ask the language models to perform. After that, we investigate how we evaluate our language models on the tasks we propose. Finally, we introduce the pre-trained language models that we use.

4.1 Overview

In figure 4.1, we depict the design of our system and the methodology used. In order to investigate a language model's ability to emulate reasoning in predicate logic, we generate a synthetic dataset from which we derive three different tasks requiring different kinds of (inductive and deductive) inference. We can automatically label the data using the nltk model checker. Furthermore, we define a grammar to constrain our natural language, so that we can easily map between it and the language of logic.

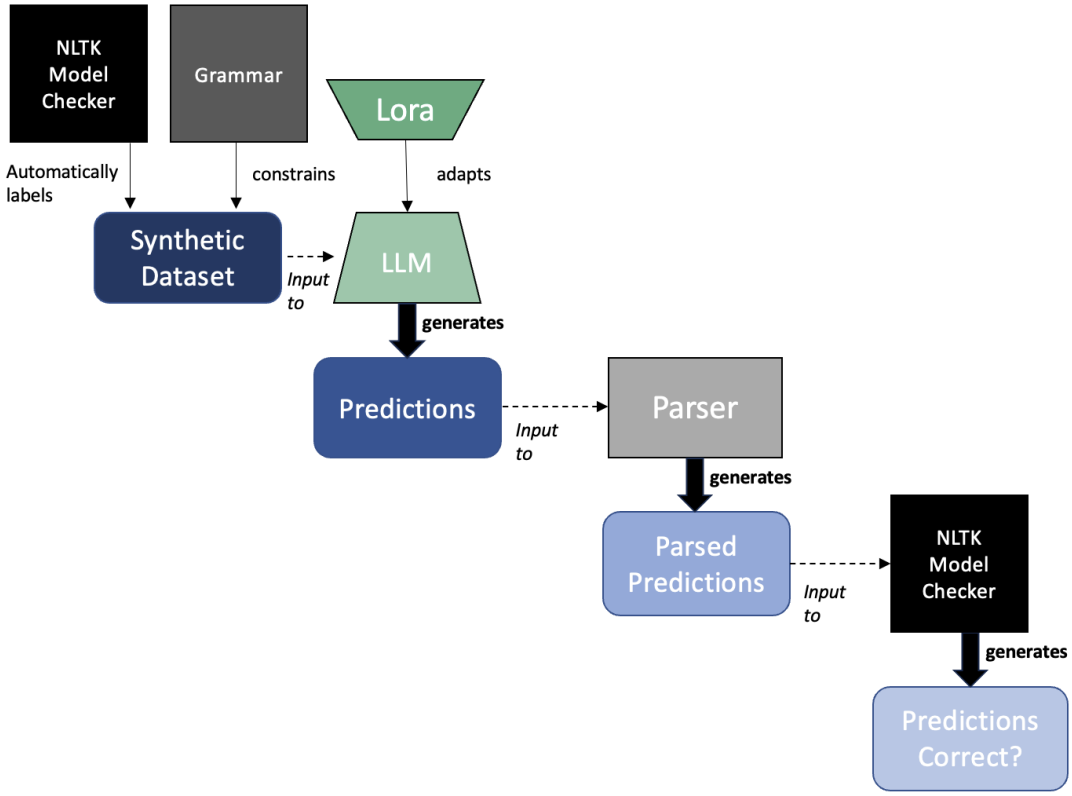


Figure 4.1: An overview of our methodology and system design.

The synthetic dataset is used as input to the large language models (LLMs) in the form of three different tasks. Each LLM generates predictions given the tasks (either during fewshot or zeroshot evaluation or after supervised finetuning). For supervised finetuning, each LLM is quantized and trained using a LoRA adapter. In order to evaluate the predictions (the generated sequences by the LLMs), we built and modified different parsers, one for each task, that parse the predictions into a form that can be understood by the nltk model checker. By entering the parsed predictions together with parts of the synthetic dataset into the NLTK model checker (Bird et al., 2009), we can judge whether the model made the correct predictions.

4.2 Dataset Construction

Our synthetic dataset starts as one dataset that will subsequently be converted into three tasks. Figure 4.2 shows that there are four (core) elements to our synthetic dataset: a

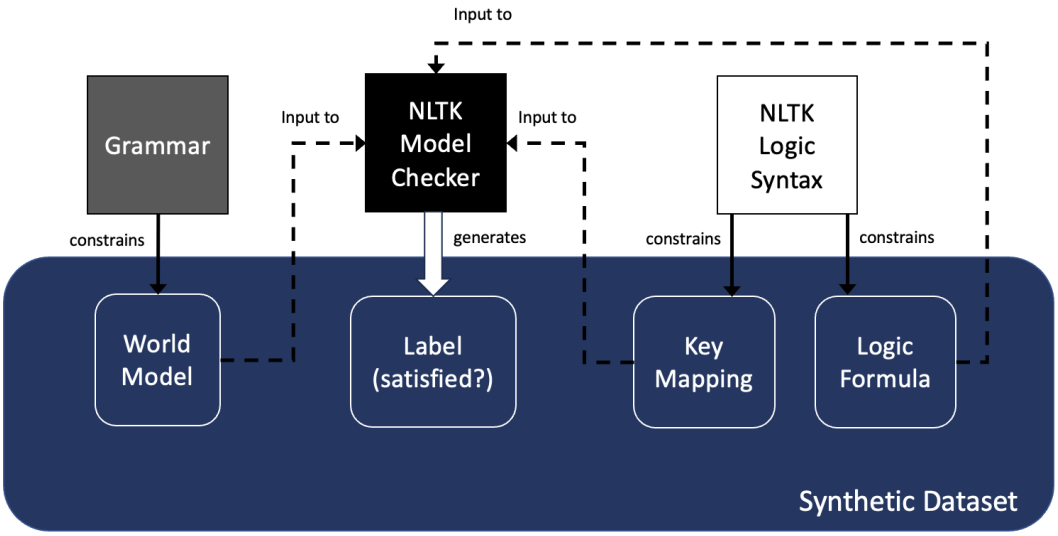


Figure 4.2: How we construct our dataset.

world model, a label, a key mapping and a logic formula. The world model, in terms of predicate logic, is our universe of discourse; i.e. a set of statements that we consider to be true about a small hypothetical world, but written in natural language. These statements are each constrained by some grammar that we define and the difficulty of the dataset that we want to create (which in turn depends on the number of constants that we want to include in our universe of discourse).

The grammar defines what can be possible predicates in natural language, and in our case, we chose a list of adjectives (so we only constrain our dataset to the case of unary predicates). It further defines what can be possible constants, and in our case, we chose a list of names. So an example statement for the world model could be:

> Melvin is charming.

Negations are also possible and are decided randomly. So a full world model with three different constants and one predicate could be:

> **World Model:** Houston is charming. Melvin is charming. Gwendolyn is not charming.

The grammar could look quite different, but is already quite expressive as it is. In order to make the dataset harder, it is, for instance, conceivable to expand it to n-ary predicates,

with $n > 1$. In that way, much of what can be stated using natural language, can also be stated in predicate logic.

The second core element is a key mapping, that is a mapping from the predicates and constants in natural language (that appear in the world model) to a representation in the language of logic that the NLTK model checker understands. Continuing with the above example, this could be:

Key mapping: 'G': 'charming', 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn'

Here, 'G': 'charming' is the predicate mapping and 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn' is the constant mapping (we decided that the world model contains statements about one to ten constants).

All formulas are written in the syntax of the nltk library of python. Note that this is also the syntax that the language models have to work with. One example of such a formula could be:

Formula: -exists x.G(x)

The formulas are created by defining what kinds of operators and quantifiers we allow, as well as the number of variables and predicates we allow, plus how deep the recursion depth should be. The recursion depth refers to the number of replacements of atomic formulas $\phi(x)$ we allow by predicates. For example, given a recursion depth of one, we could replace $\phi(x)$ once in the formula:

$$all\ x.\phi(x)$$

by some predicate $F(x)$, such that we have:

$$all\ x.F(x)$$

.

A recursion depth of two would allow a second replacement, introducing a binary operator (and possible brackets): if we introduced the & operator, the first replacement could be

$$all\ x.F(x)\ \&\ \psi(x)$$

or

$$all\ x.\psi(x)\ \&\ F(x)$$

and the second replacement would use a different predicate for replacing ψ, so as to get, e.g.

$$all\ x.G(x)\ \&\ F(x)$$

Finally, all of these three core elements (formula, world model and key mapping) are passed to the NLTK model checker which generates the label, i.e. whether the formula is satisfied given the world model and key mapping. One key aspect to keep in mind regarding the NLTK model checker is that it makes a closed world assumption, that is whatever is not known to be true (i.e. stated as true in the world model), is considered false.

Regarding our example from above

World Model: Houston is charming. Melvin is charming. Gwendolyn is not charming.

Key mapping: 'G': 'charming', 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn'

Formula: -exists x.G(x)

Satisfied?: unsatisfied

it would deduce that the formula is unsatisfied given the world model, as for instance, Houston is in fact charming, which is a counterexample to that formula, so the formula is not satisfied.

To summarize, consider algorithm 1: we first generate all formulas depending on the set of variables, predicates, quantifiers, operators and the recursion depth. Next, we sample a world model: depending on the maximum world model size, a random number of objects (constants) in this world is created as well as the world model itself (statements about those objects). From this, we can create a key mapping. Finally, the model checker determines the label (satisfied or unsatisfied) given the formula, world model and key mapping. The tuple of (formula, world model, key mapping, label) is then added to the data set, if the ADD condition is fulfilled. The ADD condition checks that we have a balanced dataset of half satisfied and half unsatisfied formulas. This whole process is repeated until the STOP condition is fulfilled. The STOP condition depends on the number of datapoints we want to create.

Algorithm 1 Dataset construction

Input: Variables set V, predicates set P, quantifiers set Q, operators set O, recursion depth r, model checker MC, max world model size w, STOP condition, ADD condition.
Output: PredLogic dataset D.

$F \leftarrow$ generateFormulas(V, P, Q, O, r);
$D \leftarrow \{\}$;
while not STOP condition **do**
    for f **in** F : **do**;
        $m, c \leftarrow$ sampleWorldModel(w, P); ▷ world model m and set of constants c
        $k \leftarrow$ generateKeyMapping(m, c, P);
        $sat \leftarrow$ MC(f, m, k);
        if ADD condition **then**
            $d \leftarrow (f, m, k, sat)$;
            $D \leftarrow D \cup d$;
        end if
    end for
end while
return D;

As can be seen from this algorithm, not only can we create a potentially infinite amount of training data, we can also modify the difficulty of the tasks by adding more variables, predicates, quantifiers and operators as well as increasing the recursion depth.

For our experiments, we generate both a base dataset and a hard dataset. For the base dataset, we allow one variable x, both existential and universal quantifiers, the conjunction (the operator "&"), the implication (the operator $\rightarrow$) and the negation operator $(-)$, up to two predicates F and/or G and we create formulas for a recursion depth of two. That is, we exclude the equivalence and the disjunction operators. While keeping formula complexity low, we still express quite a big fragment of predicate logic this way, as by the DeMorgan's laws, there are some equivalences between certain formulas with conjunction and formulas with disjunction. For the hard dataset, we augment the difficulty by introducing the following into the logic formulas (and consequently also into the world model, and key mapping): one more variable than above (so x as well as y), two quantifiers per formula (instead of just one) as well as more predicates (up to three in total, i.e. F, G and H). The purpose of the harder dataset is to find out how well the LLMs generalize after having been finetuned on the base dataset described above. An example of this dataset could look like so:

World Model: Andrea is not witty. Juniper is witty. Belle is witty. Andrea is not distressed. Juniper is distressed. Belle is distressed.

Key mapping: 'H': 'witty', 'G': 'distressed', 'o': 'Andrea', 'k': 'Juniper', 'n': 'Belle'

Formula: exists y.all x.(G(y) & (G(x) → H(x)))

Satisfied?: satisfied

Besides the four core elements (formula, world model, key mapping, label) of both the base and the hard dataset, we also keep track of certain metadata that can be derived from these elements, such as the number of variables, the number of predicates, the number and the types of operators, the types of quantifiers, which adjectives and names are used as well as an nltk representation of the keys and the world model (i.e. one that the nltk parser understands). This will help us in the quantitative analyses of the language model outputs.

4.3 Task Definitions

From the above synthetic dataset, let us call it the "synthetic predicate logic corpus" (or SPLC), we derive three tasks: a task to create a formula (task one - the "create formula task"), a task to create a world model (task two - the "create world model task") and a task to infer whether the formula is satisfied or not (task three - "deduce sat"). In figure 4.3, we show which parts of the synthetic datasets form the inputs of each respective task and which outputs are expected from the language model.

For task one, the "create formula" task, we give the language model a world model, a key mapping and a label and ask it to generate a formula that is either satisfied or not (depending on the label). For task two, the "create world model" task, we ask it to infer a world model and a key mapping from a formula, such that the formula is satisfied (or unsatisfied). Note that here, we ensure that we do not give it a task that it cannot fulfill; we would, for instance, never ask it to generate something such that a tautology (a formula that is always true) is unsatisfied, or such that a contradiction (a formula that is always false) is satisfied. For task three (deduce sat), the goal is to correctly predict the label, i.e. to indicate whether a formula is satisfied or not, given a formula, a world model and a key mapping. How the illustrative example from earlier in this chapter is used to derive the three tasks is depicted in table 4.1.

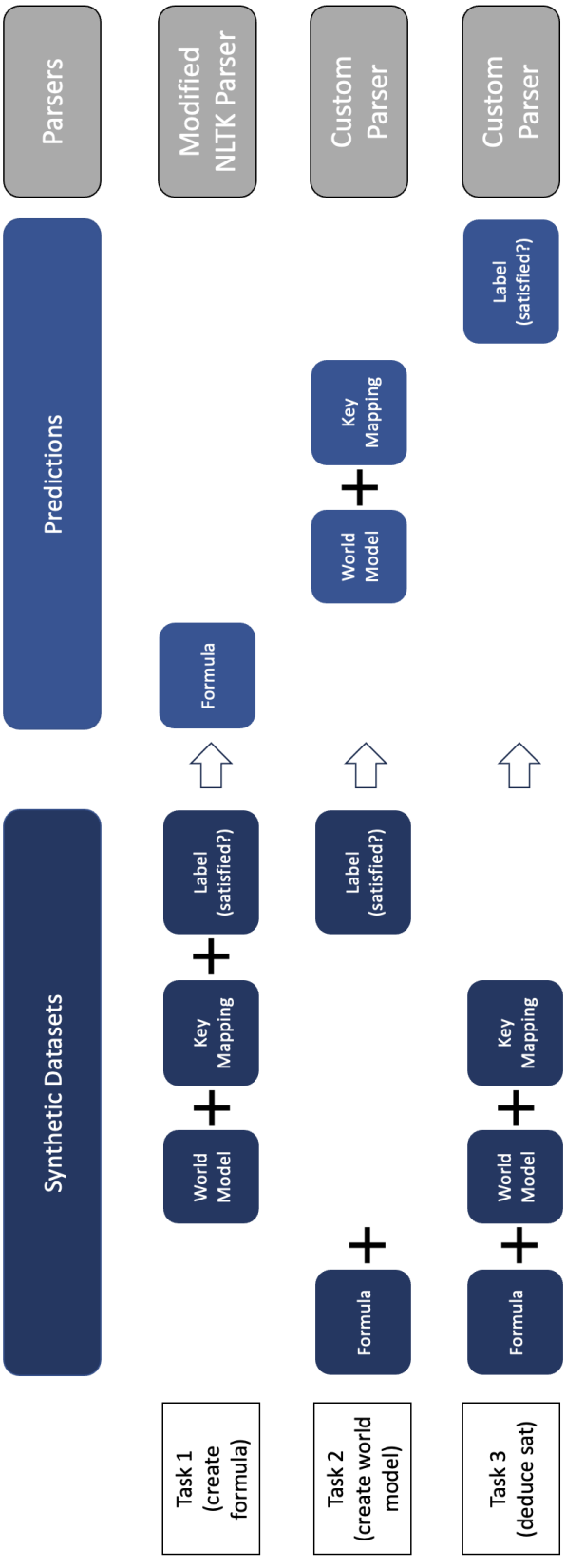


Figure 4.3: Inputs for each task and expected outputs. Parsers map between the language of logic and natural language.

With regards to all of the three tasks, we aim to erradicate any imbalances in our dataset during creation - imbalances that could help the model learn the wrong features. For instance, we create an exact 50/50 balance for the labels. However, more such imbalances might be present in the dataset, something which we investigate further in chapter 5.

Table 4.1: Illustrative example for each task derived from the base dataset.

Task	Input	Expected Output
Task 1 (create formula)	**World Model:** Houston is charming. Melvin is charming. Gwendolyn is not charming. **Key mapping:** 'G': 'charming', 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn' **Satisfied?:** unsatisfied	**Formula:** -exists x.G(x)
Task 2 (create world model)	**Formula:** -exists x.G(x) **Satisfied?:** unsatisfied	**World Model:** Houston is charming. Melvin is charming. Gwendolyn is not charming. **Key mapping:** 'G': 'charming', 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn'
Task 3 (deduce sat)	**Key mapping:** 'G': 'charming', 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn' **Key mapping:** 'G': 'charming', 'b': 'Houston', 'd': 'Melvin', 'e': 'Gwendolyn' **Formula:** -exists x.G(x)	**Satisfied?:** satisfied

4.4 Evaluation

4.4.1 Baselines

For any new dataset we need to establish baselines. Baselines are references that we compare our models' performances to. In chapter 5, we perform experiments to generate such random baselines. In the following, we explain how we establish them.

We build random but syntactically correct baselines for each of the tasks. Such a baseline supposes that the model is capable of forming syntactically correct outputs (including the correct amount of predicates for task one (create formula) and task two (create world model)), but all other variation in the generation is random. That is, for instance, in task one (create formula), the model produces a syntactically correct formula, but randomly chooses e.g. which operators and how many to use, which quantifiers to use, etc.

In order to achieve this, the following sampling algorithms are proposed: for task one (create formula), we sample two datapoints from our dataset, such that both datapoints make use of the same predicates in their formulas. This ensures that we have a syntactically correct comparison. Besides, since our dataset contains all possible formulas given our constraints on the number of variables, quantifiers, predicates, etc., we explore the full variety of different logical formulas that are syntactically correct. Then we use the model checker with the formula from the first datapoint, and the world model and key mapping as well as whether the formula should be satisfied from the second datapoint. Finally, we check if the formula is actually satisfied given the world model and key mapping (using a model checker). We repeat this sampling procedure one million times.

For task two (create world model), we perform this procedure analogous to task one (create formula). That is, we sample two datapoints from our dataset, such that both datapoints make use of the same predicates in their formulas. This ensures that we have a syntactically correct comparison. Then we use the model checker with the formula and whether it should be satisfied from the second datapoint, and the world model and key mapping from the second datapoint. Finally, we check if the formula is actually satisfied given the world model and key mapping (using a model checker). We repeat this sampling procedure one million times.

Task three (deduce sat) has a 50% chance, if we assume that the output is syntactically correct, as for this task, we only expect a binary output (satisfied or unsatisfied). Therefore, for task three (deduce sat), no algorithm for creating a random but syntactically correct baseline is needed.

4.4.2 Zeroshot & Fewshot learning, Finetuning and Generalization

We evaluate how well our models perform at the proposed tasks via zeroshot learning, fewshot learning, finetuning (on a single task and on multiple tasks) and via two different kinds of generalization tasks. Zeroshot learning refers to evaluating (language) models on a task without them having seen a single example of the task before (e.g. during pre-training). Fewshot learning, on the other hand, adds to the prompt a certain number of examples (in our case, between one and four), including the correct answer. This means that we do not perform any weight updates of the model, which is what happens during finetuning (or further training). It has been shown, that certain large pre-trained language models can already solve certain tasks in a zero-shot (Wei et al., 2021) and few-shot setting (Brown et al., 2020). We inquire whether this applies to our tasks as well.

We already defined finetuning in chapter 2, and the way we finetune is two-fold: first, we finetune the language models on every task individually (single task), then we finetune the language models on all the tasks (multi task). With regards to single task finetuning, we aim to find out how much this can improve the performance over the zeroshot and fewshot setting, on the one hand, and whether it helps to generalize to the other two tasks on the other hand. As per multi task finetuning, we would hypothesize that an emulation of an algorithm for logical reasoning would transfer between tasks, and therefore, training on all tasks should lead to better results overall.

For instance, if a human being performed task two (create world model), typically after coming up with a world model and key mapping, he/she would need to check whether the generation actually leads to a satisfied or unsatisfied formula. Similarly for task one (create formula), when a formula is to be created. That is, a human being who performs well on task one (create formula) and/or task two (create world model) would typically also perform well on task three (deduce sat), which is about deducing whether a formula is satisfied or not given a world model and key mapping.

Besides, we also evaluate our models on the task of generalizing to a harder dataset, as we defined it and gave an example in section 4.2. That is, in general, we expect it to be a harder dataset, the more variables, quantifiers and predicates we include, as well as how big the world model is. We expect this for two reasons: first, from cognitive psychology, we know that human beings can only keep track of a certain number (seven plus/minus two) of items in their mind at one time (Miller, 1956). Second, in their article (Richardson and Sabharwal, 2022), they also show that in logical reasoning in propositional logic, adding

more variables to the task decreases the performance of language models - suggesting that this makes it harder for them. We also test this in our quantitative analyses.

All of our models are finetuned using supervised finetuning on the next token prediction task. However, they are evaluated on accuracy, that is, the correct percentage after postprocessing and parsing their outputs and then passing them through the nltk model checker.

4.4.3 Dataset Sizes

Our synthetic (base) dataset has 100000 datapoints, an arbitrary number that we can increase or decrease. However, since we decided to train some very large language models, and only had restricted time and computational resources, the subsets used for evaluation and training were much smaller. Our evaluation datasets (for zeroshot, fewshot, finetuning and generalization) are of size 1000 datapoints (both for evaluating on the task with the same difficulty and with a harder difficulty). The fewshot evaluation datapoints differ in their token length, as they include a certain number of fewshot examples (one, two or four) with the input and the expected answer as part of the prompt.

For example, an evaluation datapoint for the zeroshot setting in task one (create formula) is an input-target pair, where we plug-in world model, key mapping, label (satisfied or unsatisfied) and formula from above in the respective places in the following prompt template:

Input:

```
Here is a world model:

{{ world model }}

Let us interpret predicates and names as follows::

{{ key mapping }}

Provide a logical formula in predicate logic that is {{ satisfied }} given
the above situation and interpretation (keys)!
```

Target:

{{ formula }}

So, the model is expected to produce the target (in this task, the formula).

Compared to this, an evaluation datapoint for the fewshot setting adds a certain number of randomly chosen such examples, plus the target (in order to teach the model via those examples), plus the last example input that does not appear together with the target, which the model is supposed to produce.

The training dataset for training on a single task first takes a subset of 10000 datapoints from the base dataset, performs a train-eval-test split of 60/20/20 %, such that we end up with 6000 training datapoints for each model and each task. When we train on all tasks simultaneously, however, we train on 2400 examples per task (or 7200 examples in total). We analyze and present descriptive statistics of these different training and evaluation datasets in section 5.2. In table 4.2, we summarize the number of datapoints used in finetuning and evaluation.

Table 4.2: Number of datapoints for training and evaluation datasets.

Dataset	Number of datapoints
Zeroshot Evaluation	1000
Fewshot evaluation	1000
Training single task	6000
Training Evaluation	1000
Training multi task	2400 (per task)
Hard dataset evaluation	1000

4.4.4 Parsers

In order to evaluate our models, for each of the three tasks, we built or modified a separate parser. The parsers convert the natural language outputs of the language models into a logical form (that can subsequently be used be the nltk model checker) (see figure 4.1). For task one (create formula), we modified the nltk parser to search the first and longest substring it can parse into a syntactically correct formula (see figure 4.3). This is because, as we stated before, a language model may often output more than the desired output, and not necessarily in the desired order either. For the task two (create world model), our custom parser searches for a world model and key mappings in the answer. Finally, all parsers map this natural language output to logical form. For task three (deduce sat), we implemented a parser that simply searches for the respective keywords (satisfied or unsatisfied) in the outputs.

4.4.5 Prompts

With regards to fewshot learning, we evaluate the models using two different prompts. A prompt describes the task to the language model. In section 4.4.3, we already showed the first kind of prompt. This prompt optimizes for the number of tokens. In other words, in this prompt, we try to describe the task as shortly as possible to the language model.

An example of the second prompt for task one (create formula) is as follows:

Input:

```
Context:

You will receive a problem in first-order predicate logic to solve.  This
problem contains a set of statements about the world (let us call it a 'world
model') and a mapping from things in this world to variables representing
these things (we call that mapping the 'keys').  You are then asked to provide
a formula in first-order predicate logic that is either satisfied or unsatisfied
given the world model and keys.  Please return only the formula, written
in the format of the python library nltk.

Question:

Here is a world model:

{{ world model }}

Here are the keys:

{{ key mapping }}

Please write down only one formula in first-order predicate logic that is
{{ satisfied }} given the above world model and keys.

Answer:
```

Target:

{{ formula }}

While the prompts have been designed by trial and error, the rationale behind the second prompt is that it optimizes for explaining the task well, including specifying the exact output it expects (potentially helping the language model what it should generate, and thereby potentially helping with parsing the outputs). Both prompts could be beneficial for generation: optimizing for the number of tokens could make it easier for the language model to attend to the relevant parts of the prompt, while explaining the task well could help the model to better understand what is required of it. Please refer to section 6.2) for the prompts for task two (create world model) and task three (deduce sat). Nevertheless, it remains unclear what an optimal choice of prompt would be in order to elicit correct outputs.

4.5 Models

Our investigation involves four pre-trained causal language models, that is Llama-2 (the 13 billion parameter variant) (Touvron et al., 2023), Falcon (the 7 billion parameter variant) (Penedo et al., 2023), an Orca model (with 13 billion parameters) (Mathur, 2023) and WizardCoder (in the 15 billion parameter variant) (Luo et al., 2023), as well as one sequence-to-sequence language model: Flan-Ul2 (which has 20 billion parameters) (Tay et al., 2022a). All the models are based on the original transformer architecture. Furthermore, all of them have been instruction finetuned after pre-training. Instruction finetuning is a kind of supervised finetuning that trains a pretrained model on an instruction-following task. In other words, while pre-training has as its goal to teach a language model a general understanding of building coherent and well-formed sentences, instruction finetuning aims to achieve that a language model can follow instructions and thereby complete tasks that we present to it. Hence why we also experiment with different prompts that explain our tasks. In the following, we discuss the five language models used and in table 4.3, we present a short overview of their most important aspects.

Table 4.3: Overview of the large language models used.

Language Model	Number of parameters	Architecture	Notable Aspect
Flan-UL2 (20B)	20 billion	seq-to-seq	Largest of the compared models.
Falcon-7B	7 billion	causal	Refined Dataset for Pre-training.
Llama-13B-Chat-hf	13 billion	causal	High score on BIG Bench hard.
Orca	13 billion	causal	High score on ARC.
WizardCoder	15 billion	causal	High score on coding benchmark HumanEval.

Flan-Ul2 (Tay et al., 2022a) is our only sequence-to-sequence model, i.e. the only model that uses both an encoder and a decoder from the transformer architecture. It is based on the T5 architecture (Raffel et al., 2020). We use the 20 billion parameter version, which is also the model with the most parameters that we compare. Its pre-training objective differs from the other models used in this thesis in that, instead of predicting next words, it is trained on multiple different pre-training paradigms (called denoising tasks). Denoising tasks, in essence, mask different words in different places of an input sequence, which the language model shall predict. After that, it was instruction finetuned using the FLAN collection of datasets (a wide variety of over 1800 instruction finetuning tasks) (Chung et al., 2022).

Falcon-7b-instruct is the smallest causal language model we employ (Penedo et al., 2023), with only seven billion parameters. It only uses a decoder from the transformer architecture and was pre-trained on the causal language modeling task (i.e. to predict next words). Other than this, its architecture is similar to GPT-3 (Brown et al., 2020) and uses multi-query attention (MQA) (Shazeer, 2019), compared to multi-head attention as described in chapter 2. Multi-query attention adapts multi-head attention by letting different attention heads share the same values and keys, so as to get faster decoding (i.e. generation) speed. Plus, it uses FlashAttention (Dao et al., 2022), which makes the self-attention calculations faster. Importantly, Falcon is still trained on a dataset that is primarily based on web data from CommonCrawl. However, contrary to the other models we compare, the creators of Falcon paid particular attention to filter out as much low-quality sequences and documents as possible, making the result, the Refined Web dataset, high quality for pre-training. After this step, it was instruction finetuned on instruction and conversational tasks. On benchmarks, this model does relatively badly on code generation tasks, but quite well on BIG Bench hard (which is a narrowed down subset of hard tasks from the BIG Bench dataset).

A little less is known about the LLama-2 model that we use in the 13 billion parameter variant. Their paper (Touvron et al., 2023) does not mention any data sources for pre-training or finetuning, but we note that it has been pre-trained from publicly available resources on 2 trillion tokens, as well as (instruction) fine-tuned also on publicly available data. We employ Llama in the chat variant, that is, it has been finetuned for a conversational setting. Importantly, the Llama-2 model is further trained using RLHF (reinforcement learning from human feedback) so to conform to human preferences. Besides, it uses a modified transformer (decoder-only) architecture. Notably, it does better than Falcon on the BIG Bench hard datasets, and much better than Falcon on coding tasks.

Our Orca model is the Llama-2-13b base model which was further finetuned (not on chat data but) on datasets that were built in the style of the original Orca model (Mukherjee et al., 2023). These datasets are used for imitation learning, where a smaller model is supervised by a larger model and learns from its demonstrations. In this case, it refers to datasets that are constructed by passing inputs to a larger language model (in this case 1 million datapoints from GPT-4 and 5 million datapoints from ChatGPT) and gathering their generations. These input-output sequences are then used as training data for the Orca model that is supposed to imitate the bigger model. In their paper (Mukherjee et al., 2023), the authors have shown that their Orca model scores as high on reasoning capabilities on BBH (BIG Bench hard) as ChatGPT, which is roughly 13 times the size of the orca model. In other words, the Orca model manages to imitate not only the style but also the reasoning capabilities of the larger model. Particular emphasis in these kinds of datasets has been on asking the teachers (GPT-4 and ChatGPT) to e.g. "explain like I'm five", or "step by step" to get much longer reasoning outputs, which suggests that step-by-step explanations can help models improve their learning outcomes. Notably, not only has Orca been trained on logical reasoning tasks (in natural language), but, at the time of writing, this model is one of the highest scoring large language models of its size on the hugginface leaderboard (Beeching et al., 2023) and it scores particularly well on reasoning benchmarks, such as the AI2 Reasoning Challenge (Clark et al., 2018).

WizardCoder (Luo et al., 2023), while being only a 13 billion parameter model, is one of the leading open-source models on the HumanEval benchmark, a benchmark for code generation (Chen et al., 2021). It's an instruction-finetuned Starcoder (Li et al., 2023). The Starcoder model is not instruction-finetuned itself, but uses the same architecture as GPT-2 (Radford et al., 2019), adding to it multi-query attention (MQA) (Shazeer, 2019), FlashAttention (Dao et al., 2022) and the Fill-in-the-Middle objective (FIM) (Bavarian et al., 2022). FIM asks language models to fill in what is missing between two sequences, as opposed to predicting next words (common left-to-right generation). It has been instruction finetuned similarly to Orca, i.e. via imitation learning using ChatGPT as the teacher, but with fewer datapoints, and a form of finetuning they call evol-instruct adapted to the task of code synthesis. Evol-instruct is a form of instruction finetuning, where an LLM creates more diverse and more difficult instructions from a set of instructions.

For each of our models, we make use of the implementation from the huggingface transformers library (Wolf et al., 2020). We use standard supervised finetuning, optimizing the cross-entropy loss. In chapter 5, we detail the kinds of experiments performed, and in

section 6.2, we specify the hyperparameters used for generation as well as for fine-tuning using QLoRA.

Chapter 5

Experiments and Results

In this chapter, we introduce the results of our more than 150 experiments and present qualitative and quantitative analyses to contextualize the results. The first section of this chapter establishes baselines for the respective tasks. In the second section, we describe the zeroshot learning results, and after that, the fewshot learning experiments performed and the results pertaining to it. Next, we analyze and discuss the finetuning experiments, including how well our models can generalize to the other proposed tasks as well as to a harder dataset. For the qualitative analyses, we inspect 30 random examples for each case.

The authors acknowledge support by the state of Baden-Württemberg through bwHPC. All of our experiments were performed on NVidia A100 80GB GPUs.

5.1 Baselines

We establish baselines for both the base and the hard dataset. Our first experiments are to establish random but syntactically correct baselines for each of the tasks as per the algorithms described in section 4.4. The random, but syntactically correct baseline for task one (create formula) turns out as 0.500493, or roughly 0.5. The random, but syntactically correct baseline for task two (create world model) turns out as 0.827161, or roughly 0.827. A summary of this is depicted in 5.1.

Table 5.1: Baselines for the three tasks of the base dataset.

Task	Baseline
Task 1 (create formula)	0.5
Task 2 (create world model)	0.827
Task 3 (deduce sat)	0.5

In principle, one would also expect a baseline of 0.5 for task two (create world model). So why is this not the case here? The reason for the high baseline for task two (create world model) lies in a bias in the dataset (that we only found out about after performing the following experiments). The bias is that we created certain formula-world model combinations, but not others, overlooking the fact that the size of the world models (i.e. the number of constants) correlates with the probability of either satisfying or dissatisfying a formula.

An inspection of some example formulas from table 5.2 makes this clearer. For instance, in the first row of this table, if we aim to build a world model that satisfies the formula:

$$exists\, x.F(x)$$

the higher the number of constants, the higher the probability that the world model satisfies the formula (the assumption (or design decision) being that every constant is mapped to every predicate randomly, either fulfilling or not fulfilling that predicate; in other words, our world models do not contain any distractors, or constants or predicates that are irrelevant to the respective formula). With one constant, we therefore have a 50% chance to satisfy that formula, with two constants, it's a 75% chance and so on, converging to 100% with more constants. And the symmetric case is true for not satisfying that formula, where the probability distribution converges to 0% as the number of constants increases (see row two in 5.2). However, since we add a datapoint only when it fulfills the label, we oversample large world models (with e.g. two or more constants) for satisfying that formula and we oversample small world models (with e.g. just one constant) for unsatisfying it.

This is not just the case for this one formula, but for others as well, as is shown in the table 5.2. In general, we oversample the formulas in the green color. This is because in our dataset construction algorithm (see algorithm 1), we choose the world model size randomly between one constant and the parameter "max world model size" (which refers to the maximum number of constants that we allow to appear in our world models). Therefore, we tend to have world models with sizes bigger than two.

Table 5.2: A list of select formulas from the base dataset, their possible target labels, and the probability that a certain number of constants (that are either mapped to the predicate or not with probability 0.5) fulfill the target label.

Formula	Target label	Number of constants	Probability that correct
exists x. F(x)	Satisfied	1 2 ...	0.5 0.75 ...
exists x. F(x)	Unsatisfied	1 2 ...	0.5 0.25 ...
-exists x. F(x)	Satisfied	1 2 ...	0.5 0.25 ...
-exists x. F(x)	Unsatisfied	1 2 ...	0.5 0.75 ...
all x. F(x)	Satisfied	1 2 ...	0.5 0.25 ...
all x. F(x)	Unsatisfied	1 2 ...	0.5 0.75 ...
-all x. F(x)	Satisfied	1 2 ...	0.5 0.75 ...
-all x. F(x)	Unsatisfied	1 2 ...	0.5 0.25 ...
all x. (F(x) → G(x))	Satisfied	1 2 3 ... 10	0.75 0.56 0.42 ... 0.056
all x. (F(x) → G(x))	Unsatisfied	1 2 3 ... 10	0.25 0.44 0.58 ... 0.944

Note also, that this probability distribution changes (gets more complex) for more complex formulas, such as

$$all\ x.(F(x) \rightarrow G(x)).$$

This is because when using one constant c in this case and map it with probability 0.5 to the predicates, we have four possible world model combinations: (Fc, Gc), (-Fc, -Gc), (-Fc, Gc) and (Fc, -Gc). Only the last of these combinations makes the above formula false, i.e. there is a 75% chance of satisfying the formula. The probability also converges to 0 the bigger the world model (the more number of constants) we use. So, we oversample the case where we do not satisfy this formula, as we tend to use world model sizes bigger than three (the optimal threshhold (or critical region) is between two or three constants, as can be seen from the table 5.2).

This is once more depicted in figure 5.1, where we inspect how the probability that a datapoint fulfills the target label changes depending on the world model size (i.e. the number of constants). For this, we assume that we have a complete but random mapping from constants to predicates. As a result, we oversample datapoints from above the critical region.

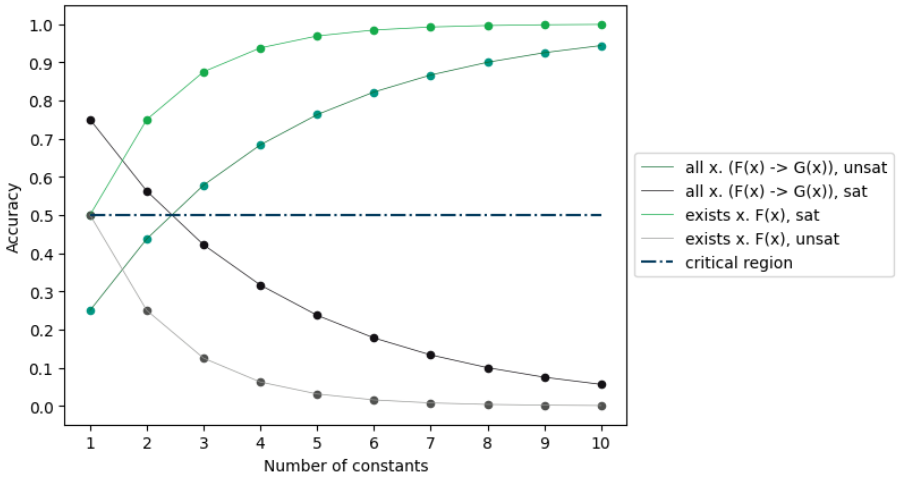


Figure 5.1: An analysis of how the number of constants (in the world model) affect the probability that a datapoint fulfills the label (assuming that we have a complete but random mapping from constants to predicates). We oversample datapoints from above the critical region.

So, in general, we have a bias in our dataset, where certain formula-target combinations appear much more often (the ones from above the critical region), and at the same time,

for these combinations, we tend to oversample large world models (to some extent as well). It turns out that because of this, most datapoints have large world models, which are once again very likely to lead to the expected label for another datapoint (which is how we construct the experiments for our random baseline). Hence, why we have such a high baseline for task two (create world model). A remedy is discussed in the section 6.2 on future work, but we can see from figure 5.1 that in order to avoid a "shortcut" for solving this task, one should sample from (close to) the "critical region", i.e. build world models close to probability of 0.5. But one should be aware that where this critical region lies changes depending on the formula (see 5.1).

Another baseline is established regarding the hard dataset. A summary is depicted in 5.3. For task one (create formula), it is 0.500888 or roughly 0.5. For task two (create world model), it is 0.953444. This is much higher than 0.5, which is due to the same reason explained for the baseline for task two above (see 5.1). Task three (deduce sat) remains at 0.5 as before.

Table 5.3: Baselines for the three tasks of the hard dataset.

Task	Baseline
Task 1 (create formula)	0.5
Task 2 (create world model)	0.953
Task 3 (deduce sat)	0.5

5.2 Dataset statistics

In this section, we present a summary of the descriptive statistics of the different training and evaluation datasets. Since the following is a summary, the full results can be found in the appendix (section 6.2). This includes the distribution of quantifiers, keys, predicates, constants, sentences in the world model, operators and labels in the respective datasets.

In all the evaluation and training datasets, we find that we have a balanced dataset regarding the labels, quantifiers, which of two binary operators is used and whether there is a negation operator in front. In other words, we usually have about 50/50% satisfied/unsatisfied labels, exists/all quantifiers, an and or an implication operator and whether a negation operator is placed in front. The only exception is the hard dataset, where we always use both quantifiers. About half of the formulas use a single predicate (either F or G), and about the other half uses two predicates. A balanced dataset ensures that our models do not learn to create only certain kinds of formulas, world models or key mappings.

Furthermore, we tested whether there are tautologies and contradictions in the datasets. There is about a 12.5% chance that a formula is a tautology and another 12.5% chance that a formula is a contradiction, across all of the datasets. This can be helpful in our analyses further down in order to determine, when a model fails or succeeds.

In summary, all of these statistics do not differ significantly across the sampled different and differently sized datasets for each of the evaluation and training settings. The only exception is the hard evaluation dataset, which was constructed differently on purpose.

The hard evaluation dataset is harder than the training dataset, as we use more predicates, more quantifiers and more operators. That is, we have both one "exists" and one "all" quantifier in each formula. We have less formulas with only one predicate, more formulas with two predicates and we introduce formulas with three predicates (F, G and H). There are less formulas that have no implication, a similar amount of formulas with one implication and we introduce formulas with two implications. Similarly, there are less formulas that have no "and" operator, a similar amount of formulas with one "and" operator and we introduce formulas with two "and" operators. The consequence of this is that we also have no formulas with just unary operators (negations), a negligible amount of formulas with only binary operators (implications or "and" operators, but no negations), but almost all formulas have both unary and binary operators. As a consequence, there appear more negations and a higher number of operators in total per datapoint. As a consequence of using more predicates, world models are typically longer as compared to the training dataset. It also means that the number of keys rises slightly whenever we use more operators, otherwise it stays comparable to the base dataset. At the same time, we keep the number of constants similar to the training dataset.

5.3 Zeroshot learning

We first perform a zeroshot evaluation of our five models on 1000 datapoints on each of the three tasks. We use the first (of the two different proposed) prompts (see section 6.2 for the exact wording of the prompts). None of the models surpasses the baselines in any of the tasks (see table 5.4). In task three (deduce sat), they perform according to the baseline, but in task two (create world model) and task one (create formula), they do not get a single example correct.

Qualitative Analysis: When we analyze a subset of the outputs of the language models, all of them struggle to provide formulas (in task one - create formula) that even produce correct syntax. The Llama and Orca models come the closest to a correct syntax, but they fail to include quantifiers or operators. One plausible reason for them doing at least somewhat better is that they have been trained on logical reasoning tasks and already do

Table 5.4: Zeroshot accuracies for the three tasks.

Model	Task 1 (create formula)	Task 2 (create world model)	Task 3 (deduce sat)
Flan-UL2 (20B)	0.0	0.0	0.513
Falcon-7B	0.0	0.0	0.027
Llama-13B-Chat-hf	0.0	0.0	0.401
Wizard (15B)	0.0	0.0	0.490
Orca (13B)	0.0	0.0	0.474

quite well on them (as we described in section 4.5). With regards to task two (create world model), the models do not seem to understand what is required of them, so no syntactically correct output is produced.

In summary, the general difficulty in the zeroshot setting seems to lie in understanding what is required of the language models for task two (create world model), and generally, in being able to produce outputs in the new language of logic. Although we do not have access to the underlying pretraining dataset, this is probably due to not being pretrained on logical languages generally, and particularly on the nltk representation of first-order logic (which has its own specific syntax and semantics). Furthermore, this is probably due to another problem which is inherent in such a logical language, in general, and that is, that is uses a lot of characters as tokens, but most large language models (probably) do not receive much training data containing characters as tokens. This seems to make handling the language of predicate logic particularly difficult.

5.4 Fewshot learning

Next, we aim to find out whether the models reach better results when performing fewshot learning. The fewshot task was performed with two different prompts (see section 6.2), as motivated in section 4.4.5. However, before we compare the results with respect to the two prompts, let us first investigate the results regarding prompt one and compare it to the zeroshot learning setting.

Our first result concerns comparing task one (create formula) in the zeroshot and the fewshot setting (using prompt one in both settings). Here, all models but the Flan-Ul2 model significantly improve their results (see table 5.5), yet they remain below the random but syntactically correct baselines.

The same happens for task two (create world model): also here, all models but the Flan-Ul2 model significantly improve their results (see table 5.6), yet again, they remain below the random but syntactically correct baselines.

Table 5.5: Zeroshot and (best) fewshot accuracies for task 1 (create formula). Prompt 1.

Model	Zeroshot	(Best) Fewshot
Flan-UL2 (20B)	0.0	0.002
Falcon-7B	0.0	0.372
Llama-13B-Chat-hf	0.0	0.397
Orca (13B)	0.0	0.357
Wizard (15B)	0.0	0.468

Table 5.6: Zeroshot and (best) fewshot accuracies for task 2 (create world model). Prompt 1.

Model	Zeroshot	(Best) Fewshot
Flan-UL2 (20B)	0.0	0.000
Falcon-7B	0.0	0.243
Llama-13B-Chat-hf	0.0	0.205
Orca (13B)	0.0	0.696
Wizard (15B)	0.0	0.591

At the same time, for task three (deduce sat), all models stay at a random baseline (see table 5.7).

Table 5.7: Fewshot accuracies for task 3 (deduce sat). One, two or four examples. Prompt 1.

Model	1 example	2 examples	4 examples
Flan-UL2 (20B)	0.488 (0.488)	0.481 (0.481)	0.478 (0.478)
Falcon-7B	0.493 (0.493)	0.493 (0.493)	0.493 (0.493)
Llama-13B-Chat-hf	0.510 (0.510)	0.504 (0.504)	0.503 (0.503)
Orca (13B)	0.501 (0.501)	0.494 (0.494)	0.497 (0.497)
Wizard (15B)	0.486 (0.486)	0.511 (0.511)	0.506 (0.506)

Note, how in table 5.7, we start to include two types of accuracies: the accuracy as we know it (percentage of syntactically and logically correctly produced outputs), plus a corrected accuracy: one that discards all the generations, where the respective parser does not parse anything that is seen as syntactically meaningful. This could be the case, if the language model produces some output that does not contain a parseable formula for task one (create formula), a world model and a key mapping for task two (create world model), or satisfied / unsatisfied for task three (deduce sat). The second kind of accuracy (we call it "corrected accuracy") is therefore at least as high as the usual accuracy. In table 5.7,

(and often throughout our tasks), these two kinds of accuracy do not differ, which means that our parsers do detect syntactically correct (but not necessarily logically correct), and therefore relevant answers. Besides, it also shows how the models, in the fewshot setting, produce a lot of syntactically incorrect outputs (due to a big difference between the two accuracies).

Next, we are interested in finding out whether more fewshot examples lead to higher accuracies. In task three (deduce sat), this is not the case, as we can see from the following figures in 5.2. The accuracies tend to stay the same.

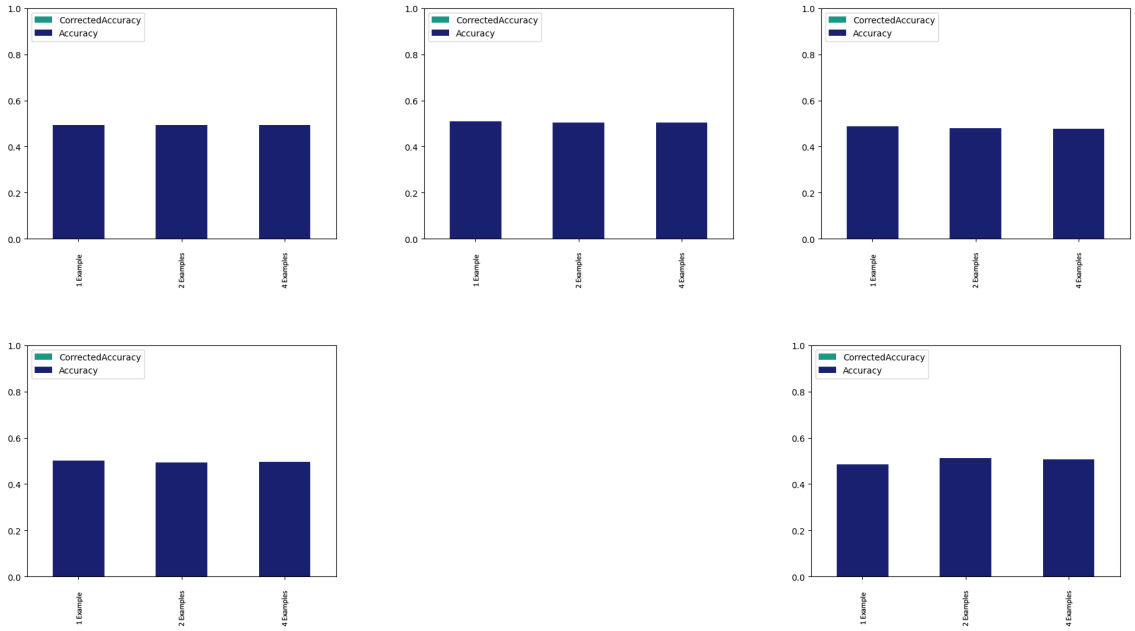


Figure 5.2: Fewshot performance on task 3 (deduce sat).
Top: Left: Falcon-7b; Middle: Llama-13b; Right: Flan-Ul2.
Bottom: Left: Orca-13b; Right: Wizard-15b.

Qualitative Analysis: In task three (deduce sat), all models perform close to random chance in all the tested fewshot settings. The typical outputs of all models is either satisfied or unsatisfied (possibly followed by some stop token, or more explanations behind their decision), so they are considered syntactically correct, but logically, they represent a random guess.

Regarding task two (create world model), receiving more fewshot examples seems to help for the Llama model, but not for the other models (see figure 5.3):

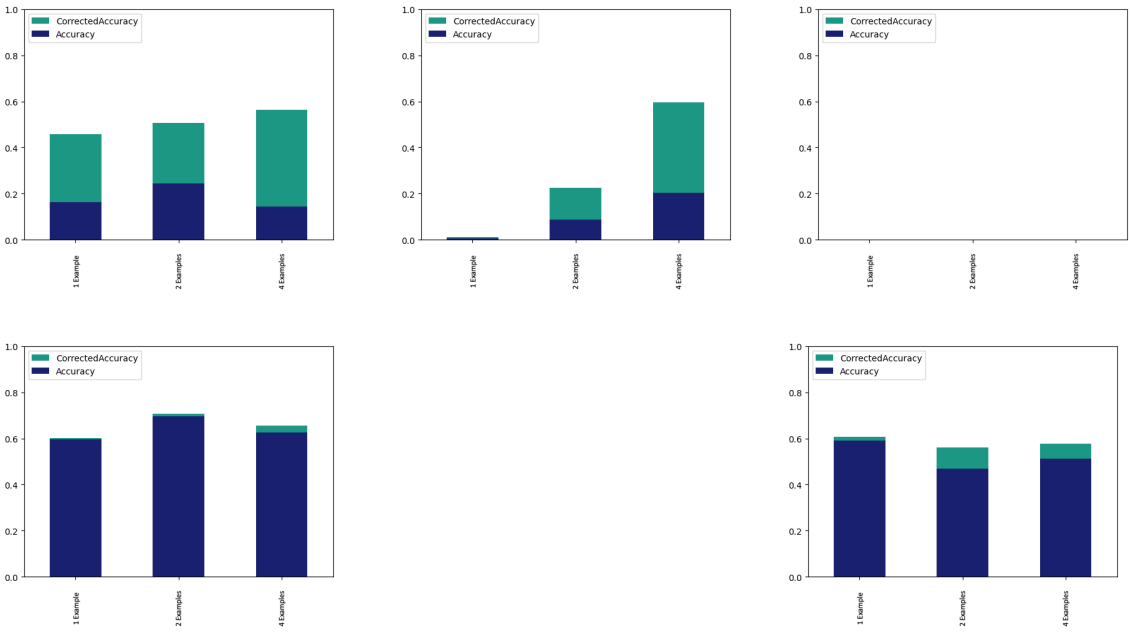


Figure 5.3: Fewshot performance on task 2 (create world model).
Top: Left: Falcon-7b; Middle: LLama-13b; Right: Flan-Ul2.
Bottom: Left: Orca-13b; Right: Wizard-15b.

For task one (create formula), it seems to help the Llama model, the Orca model and the Wizard model to some extent to receive more fewshot examples, but not the other models (see figure 5.4).

However, we note that some language models perform significantly better than others: Flan-Ul2 performs much worse than the other models on task one (create formula) and task two (create world model) - no matter the amount of examples we give it - as we can see in the two tables 5.8 and 5.9).

Table 5.8: Fewshot accuracies for task 1 (create formula). One, two or four examples. Prompt 1.

Model	1 example	2 examples	4 examples
Flan-UL2 (20B)	0.0 (0.0)	0.0 (0.0)	0.002 (0.057)
Falcon-7B	0.300 (0.380)	0.372 (0.419)	0.345 (0.362)
Llama-13B-Chat-hf	0.151 (0.344)	0.256 (0.360)	0.397 (0.428)
Orca-13B	0.057 (0.313)	0.190 (0.424)	0.357 (0.496)
Wizard-15B	0.409 (0.433)	0.432 (0.442)	0.468 (0.477)

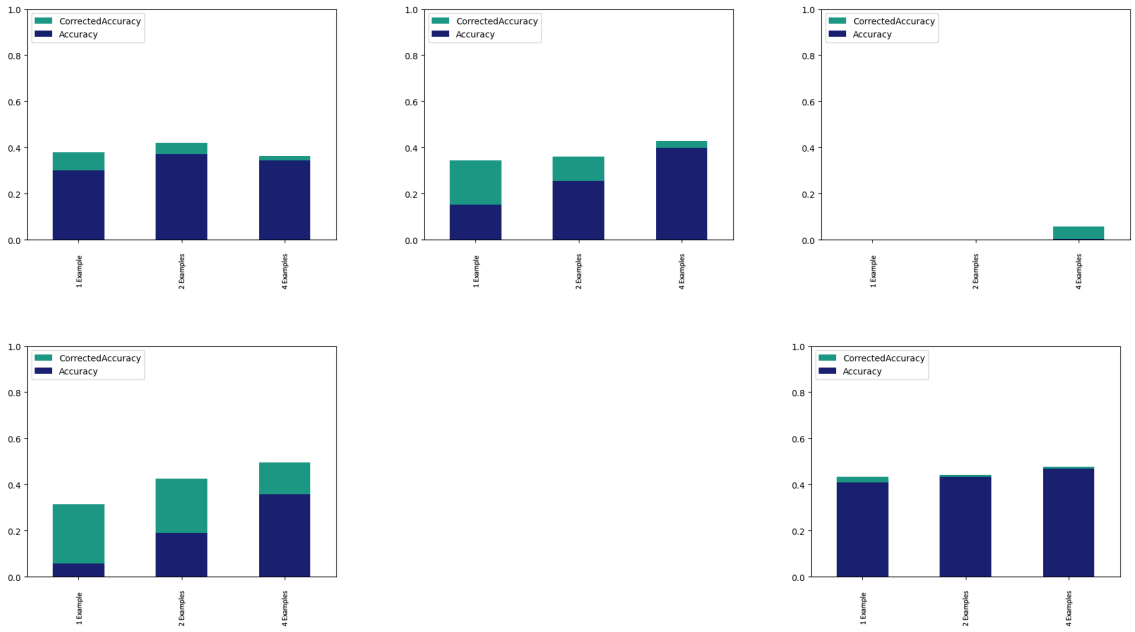


Figure 5.4: Fewshot performance on task 1 (create formula).
Top: Left: Falcon-7b; Middle: LLama-13b; Right: Flan-Ul2.
Bottom: Left: Orca-13b; Right: Wizard-15b.

Table 5.9: Fewshot accuracies for task 2 (create world model). One, two or four examples. Prompt 1.

Model	1 example	2 examples	4 examples
Flan-UL2 (20B)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
Falcon-7B	0.164 (0.457)	0.243 (0.507)	0.144 (0.563)
Llama-13B-Chat-hf	0.007 (0.013)	0.088 (0.226)	0.205 (0.596)
Orca-13B	0.597 (0.601)	0.696 (0.708)	0.626 (0.655)
Wizard-15B	0.591 (0.606)	0.468 (0.560)	0.511 (0.577)

Qualitative Analysis: Regarding task one (create formula), Flan-Ul2 only got a few answers correct in total (over all fewshot settings and all of them were in the task with the most fewshot examples). Typical correct example outputs are "-exists x.F(x)", but the vast majority of the time, it does not produce syntactically correct formulas.

Falcon produces syntactically correct outputs many times, but in this group of syntactically fine formulas, such as "exists x.-(F(x) & G(x))" or "exists x.(G(x) -> -F(x))", it tends to include unnecessarily long formulas, such as "all x.((G(x) & F(x)) & (G(x) & F(x))). In general, it also still produces some syntactically incorrect outputs.

Similarly, the Llama model produces syntactically incorrect outputs still, but a few times,

they end up to be not just syntactically but also logically correct. This is especially the case, if we look at the corrected accuracy (green bar in the figures), where we exclude syntactically incorrect generations. In that case, it compares to the falcon model.

Above, we argued that the general difficulty in generating syntactically correct output seems to lie in understanding what is the required output of the language models (e.g. for task two (create world model)), and generally, in understanding how to produce the new language of logic (e.g. for task one (create formula)). If understanding the new language of logic and being able to produce results in it would be a problem, none of the models would get an accuracy much higher than 0.0. This, however is not the case in the fewshot setting anymore for most of the models, only for Flan-Ul2. This suggests that Flan-Ul2 might not tokenize individual "characters" very well, compared to the other models, that now certainly understand how to form syntactically correct formulas in predicate logic. Another possible explanation is that the Flan-Ul2 model might have weak character embeddings as compared to the other models. Due to this issue with this model, we do not attempt to finetune the Flan-Ul2 model going further.

In order to get another datapoint, that could indicate an answer towards these two lines of arguments, we proposed a second prompt in section 4.4.5 (an example of which is shown for each task in section 6.2). The second prompt aims to be as precise as possible in telling the model what is required of it by stating to write the formula in the "format of the python library nltk" (regarding task one (create formula)), and by explaining what a key mapping and a world model is, as well as in what format those should be returned in for task two (create world model).

We find that Flan-Ul2 neither improves nor gets worse when using prompt two, indepedent of the task, as we can see from the following figures 5.5, 5.6 and 5.7:

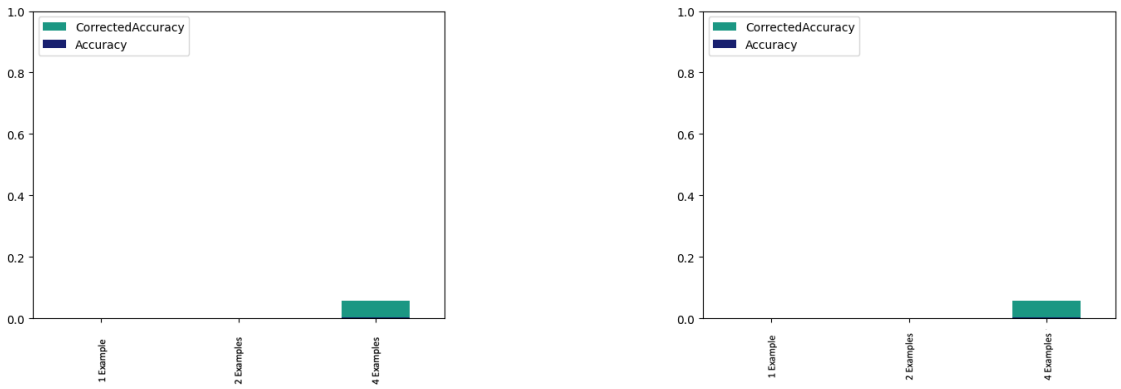


Figure 5.5: Fewshot performance of Flan-Ul2 on task 1 (create formula). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

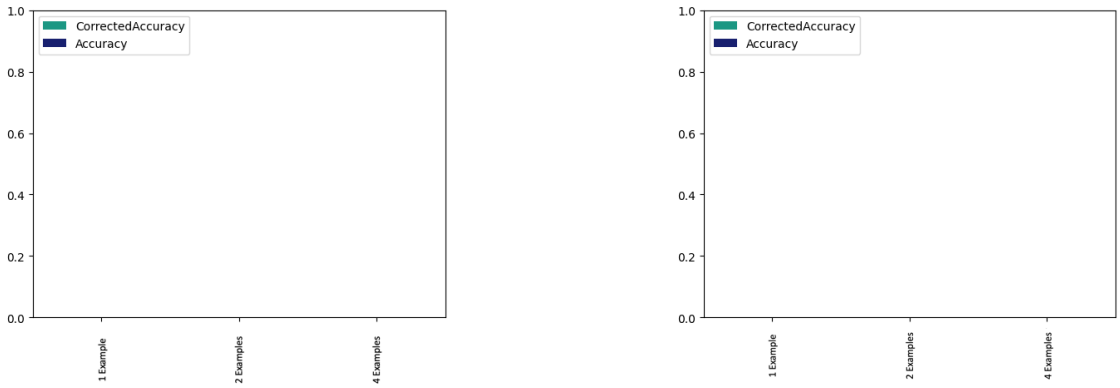


Figure 5.6: Fewshot performance of Flan-Ul2 on task 2 (create world model). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

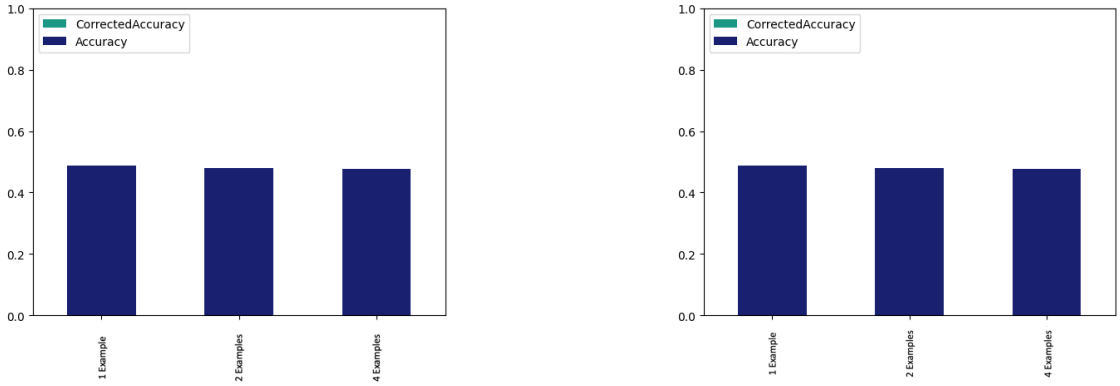


Figure 5.7: Fewshot performance of Flan-Ul2 on task 3 (deduce sat). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

The results of the other models regarding task three (deduce sat) also do not change substantially given prompt two, as compared to prompt one (please refer to section 6.2 and the beginning of this section for comparison of the exact accuracies). They still produce results with roughly 50% accuracy.

What is noteworthy is that the Llama model improves its accuracy somewhat for task one (create formula), when using prompt two instead of prompt one, while the orca and wizard models tend to produce similar results (see the following figures 5.8, 5.9 and 5.10). Prompt two tells the language model more precisely how the output should look like, and this could be helpful for generating at least a syntactically correct output.

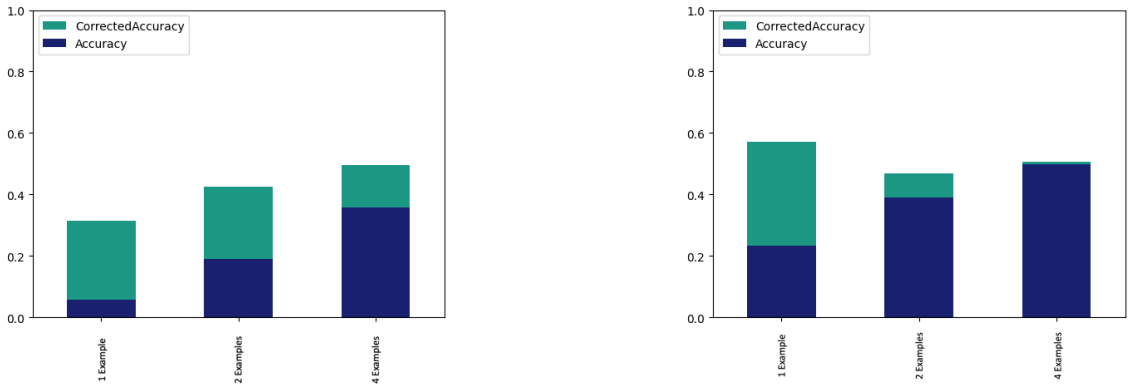


Figure 5.8: Fewshot performance of Orca on task 1 (create formula). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

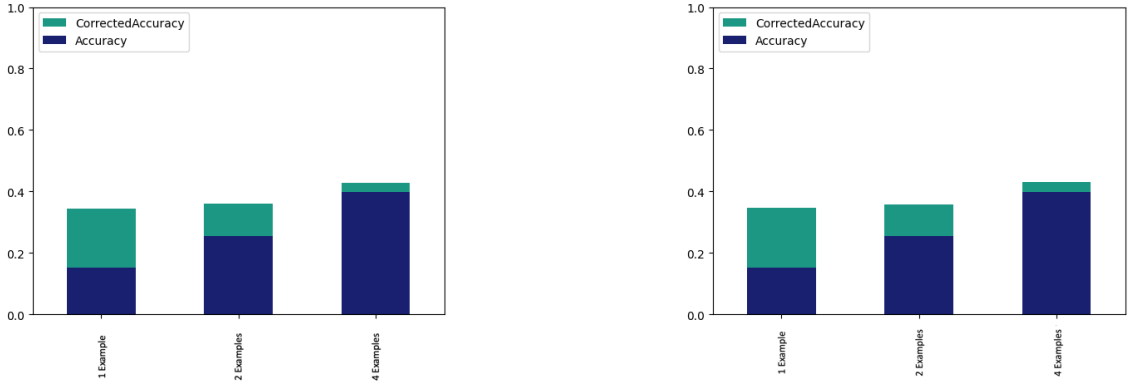


Figure 5.9: Fewshot performance of Llama on task 1 (create formula). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

The differences seem to even out when comparing the corrected accuracies (green bars). So, in general, more syntactically incorrect outputs are produced using prompt one, which seems plausible. Therefore, in order to avoid this when finetuning, we continue using prompt two for that. We also observe here, that the fewshot setting gave examples that led to an understanding of how to generate outputs in the language of predicate logic, but a precise explanation of the inputs and the expected outputs seems to further improve the results.

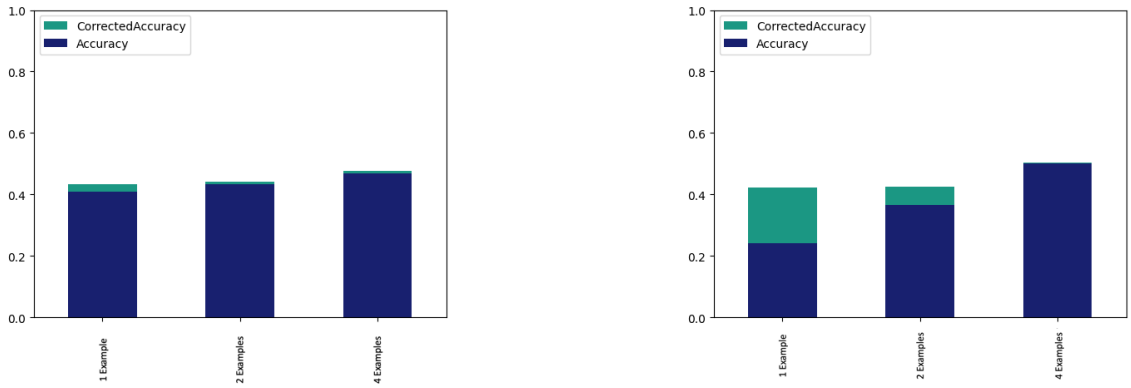


Figure 5.10: Fewshot performance of Wizard on task 1 (create formula). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

However, this does not hold true for the Orca and Wizard models regarding task two (create world model), where both perform better (throughout all tasks) given prompt one than prompt 2 see the following two figures 5.11 and 5.12).

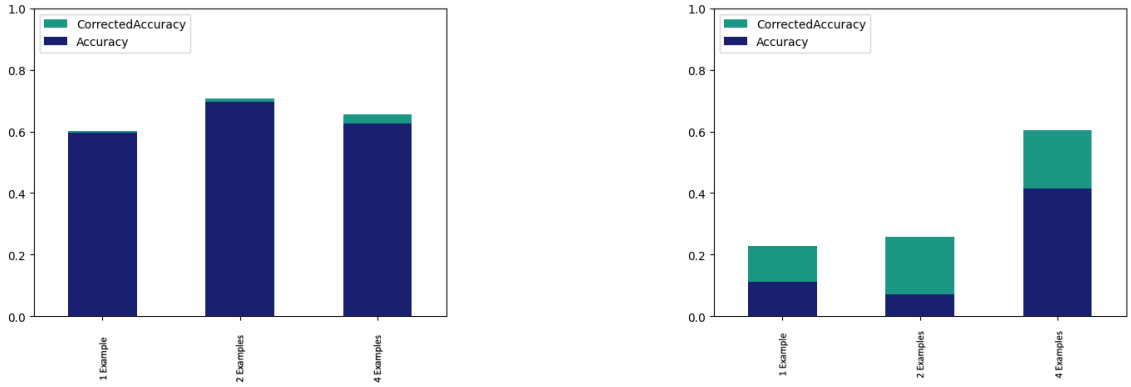


Figure 5.11: Fewshot performance of Orca on task 2 (create world model). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

One possible explanation why prompt two leads to much worse results for task two (create world model) is that the models get much longer input sequences when using prompt two. This might make it more difficult to correctly attend to the relevant relationships between all of the different variables, predicates and constants in the key mapping (and over to the formula and world model).

At the same time, since (most of) the green bars of both models using prompt two get close to (or even surpass) their respective green bars using prompt one, this suggests, that when they receive prompt two, they produce many syntactically incorrect outputs, but also that those outputs that are syntactically correct, are fairly often also logically correct.

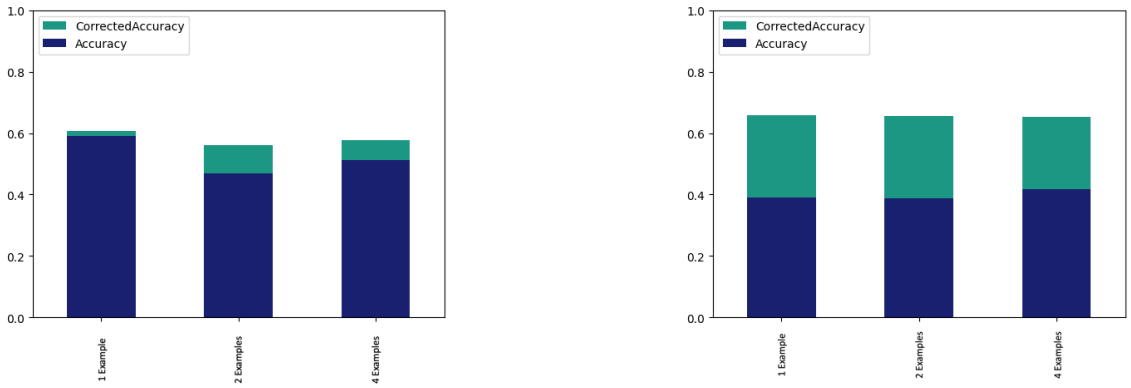


Figure 5.12: Fewshot performance of Wizard on task 2 (create world model). Prompt comparison. Left: Prompt 1. Right: Prompt 2.

In other words, they are in principle capable of producing both syntactically and logically correct outputs just as when using prompt one. Therefore, for all three tasks, we keep prompt two when performing finetuning.

Qualitative Analysis: For both prompts, both the Llama and Falcon models produce the same mistakes in their results regarding task two (create world model) in the fewshot setting. For instance, when Llama is wrong, it might miss predicates in the key mapping. It tends to provide an exhaustive world model (just like we give it to it via the one, two and four examples), that is: it maps every predicate to every constant, either with or without a negation. The Falcon model, on the other hand, produces a lot of repetitions, such as "World model: G: Safe F: Safe a: Safe b: Safe c: Safe..." etc., where "..." refers to further repetitions of the key mapping (from any lowercase letter of the alphabet to "Safe"). So qualitative inspection does not give us any further hints as to why these models differ here.

In summary, fewshot shows that most models get syntactically better and even produce logically correct outputs, but unfortunately, still not above the respective baselines (so still not logically correct more than random chance). Therefore, pre-trained models cannot immediately emulate reasoning in predicate logic when receiving some fewshot examples and therefore, finetuning is suggested to improve this.

5.5 Finetuning

5.5.1 Finetuning on one Task

We finetuned four of our previously inspected models: Falcon-7B, Llama-13B (in the finetuned variant for chat), Orca-13B and Wizard-15B. Please find the hyperparameters in the

appendix section D. We can show that LLMs are capable of emulating logical reasoning in predicate logic in all three tasks: three models, Falcon, Llama and Orca perform above baseline on the inductive task (task 1 - create formula), and most models perform above baseline on task 2 (create world model) and task 3 (deduce sat).

On task 1 (create formula) (see table 5.10), all models besides Wizard surpass the baseline. A possible explanation why the wizard model does not perform above baseline may be due to the hyperparameter choice, as it differs from that of the multitask setting, where it achieves an accuracy above the baseline for this task (see table 5.16).

Task 1 (create formula):

Table 5.10: Finetuning accuracies for task 1 (create formula).

Model	Accuracy
Falcon-7B	0.909 (0.909)
Llama-13B-Chat-hf	0.985 (0.985)
Orca-13B	0.699 (0.699)
Wizard-15B	0.017 (0.017)

Task 2 (create world model): All four models beat the baseline for this task. The best models, Llama and Wizard, achieve 89.6% accuracy in task two (create world model) compared to a baseline of 82.7%.

Table 5.11: Finetuning accuracies for task 2 (create world model).

Model	Accuracy
Falcon-7B	0.889 (0.889)
Llama-13B-Chat-hf	0.896 (0.896)
Orca-13B	0.848 (0.877)
Wizard-15B	0.881 (0.881)

Task 3 (deduce sat): All models also beat the baseline for task three (deduce sat), which is 50%. However, the Falcon model only surpasses the baseline slightly. The best model (Orca-13B) in task 3 (deduce sat) achieves 90.1% accuracy.

Table 5.12: Finetuning accuracies for task 3 (deduce sat).

Model	Accuracy
Falcon-7B	0.561 (0.561)
Llama-13B-Chat-hf	0.867 (0.867)
Orca-13B	0.901 (0.901)
Wizard-15B	0.783 (0.783)

It should be noted that the models improve over the fewshot setting in producing syntactically correct outputs for all three tasks, and surpassing the baselines on all three tasks as well, meaning they learned some logical reasoning in addition to producing syntactically correct outputs. This is especially noteworthy regarding task one (create formula) as the artificial language of predicate logic was a new language to the models (as far as we know from the pre-training process).

Besides these successes, two things are striking, however: first, the wizard model performs badly on task one (create formula). Second, the Falcon model performs much worse than the other models on task three (deduce sat). In the following qualitative and quantitative analyses, we aim to inquire why this is the case. From these analyses, we find that both Wizard and Falcon seem to only understand syntax of predicate logic but not the semantics, in these two cases.

The Falcon model was trained on significantly fewer epochs for task three (deduce sat) than the other models (see D). This might explain its worse performance. However, it was also trained on somewhat fewer epochs regarding the other tasks and there it does fine. Similarly, llama was also trained on somewhat fewer epochs regarding task three (deduce sat) and does quite well here (, refer to D for the exact number of training epochs).

Qualitative Analysis: One thing that seems problematic, regarding task one (create formula), is that Llama and Orca give multiple different formulas within one answer. For evaluation, we always choose the first of these formulas.

While the Falcon model does overwhelmingly good and beats the baseline, it seems that it gets an overwhelming amount of implications (using the implication operator) wrong. This can be investigated further during quantitative analysis.

The Wizard model generally produces syntactically correct formulas, but they tend to be relatively long, often repeating predicates such as "G(x)" multiple times, which would not be necessary. This suggests, that Wizard has not understood how to form formulas that are logically correct.

With regards to task two (create world model), all four models produce very similar outputs. They are typically mostly syntactically correct, but they often add a lot of extras to the key mapping.

As a side note, Falcon may produce the (syntactically and logically) correct formula (for task one (create formula)), but give a wrong explanation for it. This can certainly only be an anecdote, as we did not train it on providing explanations nor did we ask it to provide them. Similarly, regarding task three (deduce sat), as an anecdote, Llama, Wizard and Orca often give a correct solution, but when adding an explanation, it may be correct or incorrect.

Quantitative Analysis: In this quantitative analysis, we look at select models regarding certain tasks and inquire what kinds of variables correlate with their performance. With respect to models that perform well, we look at Falcon and Llama for task one (create formula), Wizard for task two (create world model) and Orca for task three (deduce sat). With respect to models that do not perform well, we look at Falcon for task three (deduce sat). For this, we use the descriptive statistics introduced in section 5.2, that is whether it is harder for a model to get a good accuracy, if there are e.g. more predicates in the formula.

One aspect that we also analyze here, that was part of the descriptive statistics, is whether the models can work with tautologies / contradictions (when given to them), and whether they produce tautologies / contradictions for task one (create formula). Producing tautologies or contradictions for task one (create formula) would be the trivial solution, as tautologies are formulas that are always satisfied and contradictions are formulas that are never satisfied. Even though, we show that this is not the case, this still means that task one (create formula) should be changed to ask to create not just any formula, but one that is neither a contradiction nor a tautology in a further iteration of working with these tasks.

Llama and Falcon score the highest on task one (create formula). It does not seem that the kind of quantifier, the number of keys, the number of constants, the number of sentences in the world model (i.e. the world model size), nor the number of operators or negations, nor whether there is a negation in front of the formula correlate with either true or false answers for Llama. Besides, Llama does not manage to produce a single correct contradiction and it produces tautologies, but not consistently so that they are are always correct. This last point means that the Llama model does not find the trivial solution to task one (create formula). Only the following aspects are striking: first, one predicate is slightly easier than two predicates; second, when it produces an implication it tends to be correct more often than when it does not and third, analogously so for the "and" operator. So in summary, we can distinguish what the model has difficulty with (when it generates something), and what creates a difficulty for the model (which inputs make the task harder): the model has a slight difficulty when creating formulas that have neither an "and" operator nor an implication operator. But more predicates make task one (create

formula) more difficult for Llama (see figure 5.13).

For Falcon, just as with Llama, it does not seem that the kind of quantifier, the number of keys, the number of constants, the number of sentences in the world model (i.e. the world model size), nor the number of operators or negations, nor whether there is a negation in front of the formula correlate with either true or false answers. Besides, Falcon produces both contradictions and tautologies, but not consistently correctly, i.e. Falcon also does not use the trivial solution to task one (create formula). Only the following aspects are striking: first, one predicate is slightly easier than two predicates; second, and contrary to Llama, when it produces an implication it tends to be correct slightly less often than when it does not and third, analogously so for the "and" operator. So in summary, we can distinguish what the model has difficulty with (when it generates something), and what creates a difficulty for the model (which inputs make the task harder): the model has a slight difficulty when creating formulas that have either an "and" operator or an implication operator as opposed to no binary operator. But more predicates make task one (create formula) more difficult for Falcon.

For the Wizard model trained on task two (create world model) and evaluated on it, it does not seem that the kind of quantifier, nor the number of operators or negations, nor whether there is a negation in front of the formula correlate with either true or false answers. Only the following aspects are striking: First, wizard makes no errors when only 2 or 3 keys are produced in the key mapping. The more keys it produces, the less correct it is. Similarly, for the number of constants and the world model size (although these correlate with the number of keys, so this is not surprising). Second, Wizard does not make any errors, when there is only one predicate in the formula, but only when there are two predicates, so the more predicates that appear in the formula, the less accurate the model is (see figure 5.14). Third, a negation in front of the formula leads to a little bit less accuracy compared to the case when there is no negation in front.

With respect to Orca finetuned on task three (deduce sat), it does not seem that the kind of quantifier, the number of constants, nor the number of operators or negations, nor whether there is a negation in front of the formula correlate with either true or false answers. Rather, this models produces better outcomes, the less keys there are (e.g. it makes no errors with only two keys), the less predicates there are (it makes almost no error with one predicate, see figure 5.15), the smaller the world model is. Besides, it makes slightly more errors if there is an "and" operator and slightly less, if there is an implication, interestingly. Also, it fares much better with contradictions and tautologies, as compared to formulas that are neither!

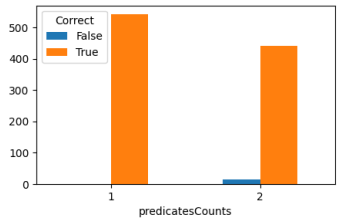


Figure 5.13: Llama-13b (finetuned on task 1 - create formula).

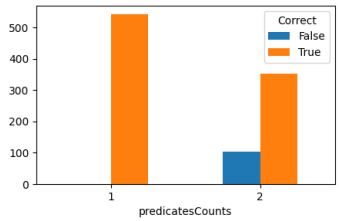


Figure 5.14: Wizard-15b (finetuned on task 2 - create world model).

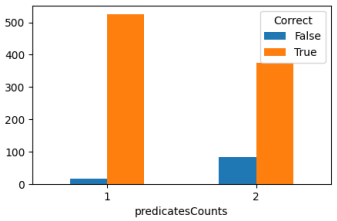


Figure 5.15: Orca-13b (finetuned on task 3 - deduce sat).

Figure 5.16: Finetuned performance: Correct and Incorrect answers on the respective task depending on whether one or two predicates appear in the formula. More predicates means lower accuracy across tasks and models.

Finally, Falcon performs worse than the other models on task three (deduce sat), but still above the baseline. For this model, it does not seem that more predicates or a bigger world model, nor the kind of binary operator (implication or and operator), nor whether there is a negation in front, affect this. It gets most tautologies wrong, but most contradictions right. And it gets somewhat better results with more keys, constants, operators and negations, something which seems counterintuitive. So we speculate that it might just have learned an incorrect way to solve the task, as it does not produce an accuracy that is much over the baseline.

In summary, with regards to task one (create formula), two (create world model) and three (deduce sat), quantitative analysis suggests that it gets harder the more predicates are used. For task three (deduce sat), adding not only to the predicates, but also to the world model and keys seems to make it a harder task. Nevertheless, we can finetune models such that they perform reasonably above baselines on all three tasks.

5.5.2 Generalization to other Tasks

We tested how well these models generalize to the respective other tasks. If a large language model were to emulate reasoning in first order predicate logic, one would expect that a model that does well on one of the tasks, would also do reasonably well on the other tasks. On the one hand, this is because during supervised finetuning, they learn the syntax of the respective other tasks. On the other hand, generalization would typically be the case for humans learning to solve these tasks, as for task one (create formula) and two (create world model), we would need to check if a generation actually satisfies the formula or not, so, we would need to perform well on task three (deduce sat) as well. However, we do not observe generalization to the other tasks, as can be seen in tables 5.13, 5.14 and 5.15.

Table 5.13: Generalization from language models trained on Task 1 (create formula) to other tasks.

Model	Task 2 (create world model)	Task 3 (deduce sat)
Falcon-7B	0.0 (0.0)	0.302 (0.489)
Llama-13B-Chat-hf	0.004 (0.023)	0.455 (0.482)
Orca-13B	0.0 (0.0)	0.495 (0.495)
Wizard-15B	0.232 (0.244)	0.499 (0.499)

Table 5.14: Generalization from language models trained on Task 2 (create world model) to other tasks.

Model	Task 1 (create formula)	Task 3 (deduce sat)
Falcon-7B	0.0 (0.0)	0.006 (0.222)
Llama-13B-Chat-hf	0.0 (0.0)	0.289 (0.426)
Orca-13B	0.0 (0.0)	0.665 (0.665)
Wizard-15B	0.001 (1.0)	0.007 (0.412)

Table 5.15: Generalization from language models trained on Task 3 (deduce sat) to other tasks.

Model	Task 1 (create formula)	Task 2 (create world model)
Falcon-7B	0.0 (0.0)	0.0 (0.0)
Llama-13B-Chat-hf	0.0 (0.0)	0.099 (0.274)
Orca-13B	0.0 (0.0)	0.021 (0.072)
Wizard-15B	0.281 (0.281)	0.881 (0.881)

What surprises is that Orca that was trained on task 2 (create world model) gets a higher accuracy than baseline on task 3 (deduce sat) (from the models that were not specifically trained on task 3 (deduce sat)), namely 66.5%. This may have been due to Orca being explicitly finetuned on logical reasoning tasks (before our finetuning process on our tasks).

Qualitative Analysis: Qualitative analysis suggests that the models do not necessarily produce the correct syntax for the respective other tasks, and even if they do, are logically incorrect.

5.5.3 Finetuning on all tasks

As a next step, and since many models tend to perform much better when finetuning them, but do not generalize well, we finetune the base models on all three tasks at the same time. Finetuning on all tasks (after shuffling the respective datapoints) tends to lead to a medium-high accuracy across tasks for most models (see table 5.16). That is, for task one, all of the models perform higher than baseline, but not as high as in single task finetuning. For task two (create world model) only the orca model performs above baseline. and for task three (deduce sat), the accuracies are only slightly above the baseline.

Table 5.16: Finetuning accuracies for all tasks after training on all tasks.

Model	Task 1 (create formula)	Task 2 (create world model)	Task 3 (deduce sat)
Falcon-7B	0.754 (0.754)	0.728 (0.747)	0.565 (0.565)
Llama-13B-Chat-hf	0.736 (0.736)	0.548 (0.873)	0.631 (0.631)
Orca-13B	0.720 (0.720)	0.896 (0.896)	0.562 (0.562)
Wizard-15B	0.691 (0.691)	0.692 (0.743)	0.584 (0.584)

One possible reason for this, is that training on multiple tasks involved 7200 datapoints over all tasks, but only 2400 datapoints for each single task, as compared to 6000 datapoints when training on just a single task. So, one could continue to train these models on the same amount of data for each task in the future and compare them. What is more, for the multitask finetuning setting, we only train for three instead of ten epochs (which we do for single task finetuning) (see section D). Since more training and more datapoints tend to help with the performance, this might explain it.

For the Llama model, regarding task two (create world model), there is a big difference between the accuracy and the corrected accuracy. That is, the Llama model does not seem to generate syntactically correct (parseable) output most of the time, only that when it does, it is doing quite well.

Qualitative Analysis: In general, regarding task one (create formula), when Falcon makes errors, it makes quite a few syntactic errors in the formulas, for instance: "all x.-(F(x) -> F(x))exists x.F(x)". The Llama, Wizard and Orca models typically produce syntactically correct formulas in task one (create formula), but are often logically wrong. Regarding task two (create world model), once again, the models add unnecessary parts to the key mappings. All four models perform only somewhat above the baseline on task three (deduce sat), but produce syntactically correct "satisfied" or "unsatisfied" answers.

Quantitative Analysis: It is noteworthy that the Wizard model is better on task one (create formula), after training it on all three tasks as compared to only being trained on task one (create formula). Quantitative analysis shows that when it is trained on all tasks, its accuracy tends to be worse, the more predicates are in the formula. Also, it seems like it can handle contradictions quite well (not perfectly, though), but not tautologies. So, it seems that here, the model simply may have started down the "correct gradient" that makes it learn what it needs to for task one (create formula).

In summary, both qualitative and quantitative analyses did not indicate anything other than the following: training on all tasks at the same time mimics the individual finetuning outcomes closely, albeit with a typically worse accuracy (with the exception of one case), which is probably due to training on less datapoints and for fewer epochs.

5.5.4 Generalization to a harder Task

In order to test generalization further, we also defined a harder dataset. We have already described that dataset and given an example in section 4.2. As a reminder, we augment the difficulty of the "SPLC", by introducing the following into the logic formulas (and consequently also into the world model, and key mapping): one more variable (that sums up to two variables in total), two quantifiers in one formula (instead of one), three predicates (instead of two). We call the resulting dataset the "hard SPLC" (or the hard dataset). Consequently, the hard dataset is a disjunct dataset to the base dataset from before. This is the advantage of our method: we can, in principle, create infinitely complex logic formulas.

With regards to task one (create formula), we took the three best models and in table 5.17, we observe that they generalize exceptionally well.

Table 5.17: How well do the models generalize to a harder dataset w.r.t. task one (create formula)?

Finetuned Model	Finedtuned on ...	Accuracy on finetuned task	Evaluation on hard task	Accuracy on hard task
Falcon-7B	Task 1 (create formula)	0.909 (0.909)	Task 1 (create formula)	0.869 (0.869)
Falcon-7B	All tasks	0.754 (0.754)	Task 1 (create formula)	0.770 (0.770)
Llama-2-13B	Task 1 (create formula)	0.985 (0.985)	Task 1 (create formula)	0.939 (0.939)

Qualitative analysis: Qualitative analysis shows that they produce syntactically correct formulas, however, they do not make use of all the predicates. Both Falcon and Llama, for instance, manage to make use of a new predicate token (correctly abstracting its use),

but they use only a subset of the predicates to build the formula (i.e. e.g. only using predicate F and G and one operator, not three predicates and two operators). An even harder generalization could ask to use all predicates in a future iteration of this work.

Quantitative analysis: Quantitative analysis shows that part of the formulas are tautologies and contradictions, but e.g. Llama still makes mistakes in this regard, and the percentage of tautologies and contradictions closely matches those of the training dataset.

With regards to task two (create world model), we took several of the best models and in table 5.18, we observe that they generalize less well than the models for task one (create formula). That is, they do not match the baseline. The baseline is quite high in this case, and this is because the baseline assumes that the models produce not just syntactically correct outputs, but also outputs that use all of the predicates. This is something that could reasonably be asked of the LLMs, however.

Table 5.18: How well do the models generalize to a harder dataset w.r.t. task two (create world model)?

Finetuned Model	Finedtuned on ...	Accuracy on finetuned task	Evaluation on hard task	Accuracy on hard task
Falcon-7B	All tasks	0.728 (0.747)	Task 2 (create world model)	0.544 (0.554)
Falcon-7B	Task 2 (create world model)	0.889 (0.889)	Task 2 (create world model)	0.668 (0.671)
Llama-2-13B	Task 2 (create world model)	0.896 (0.896)	Task 2 (create world model)	0.602 (0.626)
Orca-13B	All tasks	0.896 (0.896)	Task 2 (create world model)	0.788 (0.792)
Orca-13B	Task 2 (create world model)	0.848 (0.877)	Task 2 (create world model)	0.478 (0.527)
Wizard-15B	Task 2 (create world model)	0.896 (0.896)	Task 2 (create world model)	0.674 (0.674)
Wizard-15B	Task 3 (deduce sat)	0.881 (0.881)	Task 2 (create world model)	0.789 (0.828)

Qualitative analysis: Qualitative analysis of a subset of the models that were evaluated on task two (create world model) (i.e. Falcon trained on task two (create world model), Orca trained on all tasks and Wizard trained on task three (deduce sat)) shows that, the Wizard model sometimes does not map some of the predicates, thereby producing a smaller world model. The falcon model, on the other hand, produces an overly long key mapping, ensuring to map everything but at the expense of accuracy. Orca does not suffer from this problem and it even maps all of the predicates neatly. For this model, we not only see a high generalization accuracy, but also qualitatively the best generalization (even if not beating the baseline). It is, after all, a harder task to produce mappings for more predicates, so a drop in performance is expected.

With regards to task three (deduce sat), we took several of the best models and in table 5.19, we observe that they do not generalize, as they merely produce an accuracy close to the baseline. This stark drop in performance seems to indicate that we have created a hard task for language models.

Table 5.19: How well do the models generalize to a harder dataset w.r.t. task three (deduce sat)?

Finetuned Model	Finedtuned on ...	Accuracy on finetuned task	Evaluation on hard task	Accuracy on hard task
Wizard-15B	Task 3 (deduce sat)	0.783 (0.783)	Task 3 (deduce sat)	0.497 (0.497)
Orca-13B	Task 3 (deduce sat)	0.901 (0.901)	Task 3 (deduce sat)	0.554 (0.554)
Llama-2-13B	Task 3 (deduce sat)	0.867 (0.867)	Task 3 (deduce sat)	0.501 (0.501)
Falcon-7B	All tasks	0.565 (0.565)	Task 3 (deduce sat)	0.495 (0.496)

In summary, and in general, our results indicate that while models being finetuned on a task can produce outcomes above the respective baselines and thereby emulate reasoning in predicate logic to some extent, they also fail to generalize to similar tasks and to harder tasks (of the same type of problem). This seems to indicate that we have created a hard task for language models.

Chapter 6

Conclusion

This section concludes by first answering the design and research questions from chapter 1, and second discussing possibilities for future work.

6.1 Answers to Research Questions

In the following, we give answers to the design and research questions from the introduction. While carrying out more than 165 experiments, we found out the following:

Design Question: "How should a dataset look like that teaches large language models predicate logic?"

We designed a dataset that has several advantages: first, it is automatically labelled and can automatically deduce, by use of a model checker and several parsers, whether the produced outputs by the language models are correct. Second, it combines both the natural language humans use everyday as well as the artificial language of predicate logic, that we require the language models to learn and map to natural language. Third, it includes multiple tasks that each depend on each other in that they stem from the same base dataset. This way, we can test generalization, which we can use as an indication to whether the language models learn the same abstract rules behind manipulating natural language and the language of predicate logic. Fourth, the difficulty of the dataset can be adjusted along several dimensions (e.g. number of predicates, quantifiers, operators, world model size, etc.) in order to test generalization and track progress of future language models that might be even better in handling predicate logic.

Research Question 1: What is a suitable **baseline** for the three tasks?

Our experiments form a random and syntactically correct baseline that also asks the models to use the correct number of predicates. This is a high bar, but a minimum bar, and one that does not require to also produce logically correct formulas.

Research Question 2: How high do current pre-trained large language models score

in these tasks in a **zeroshot evaluation**?

They score the baseline for task three (deduce sat), but obtain an accuracy of 0.0 for the other two tasks (create formula; create world model). This is probably due to not being pretrained on logical languages generally, and particularly on the nltk representation of first-order logic (which has its own specific syntax and semantics), plus to not fully understanding what outputs are required of them.

Research Question 3: How high do current pre-trained large language models score in these tasks in a **fewshot evaluation**?

They improve their syntax with respect to what is required of them as pertains to task one (create formula) and two (create world model), while no improvements are observed for task three (deduce sat). Again, this is probably due to not being pretrained on logical languages generally, and particularly on the nltk representation of first-order logic (which has its own specific syntax and semantics), but now they start to understand what outputs are required of them and so they produce (close to) randomly correct answers.

Research Question 4: Does different **prompting** change the fewshot results?

Different prompting does not produce any significant differences in the results.

Research Question 5: Can current SOTA large language models emulate reasoning in first-order predicate logic?

We can show that they can, as they perform reasonably above the baselines on all three tasks. However, other than in one instance, they do not come close to 100% accuracy, indicating some scope for future work on these tasks.

Research Question 6: Given a finetuned model that does well on one task, does this generalize to one of the other tasks?

We observe at most weak generalization to the other tasks, indicating some scope for future work.

Research Question 7: Does multitask learning (i.e. finetuning a large language model on all three tasks) lead to higher accuracy in the respective tasks compared to the singletask setting?

Unfortunately, it does not. However, in this setting, we finetuned the models on less datapoints than in the single task setting, and the accuracies are close to that setting and tend to be above the respective baselines, suggesting that a similar amount of datapoints and training could lead to similar levels of performance.

Research Question 8: Given a finetuned model that does well on one task, does this generalize to a harder version of the same task?

We do observe generalization to a harder dataset on task one (create formula), however, the models use various techniques to perform well in this setting that the authors did not foresee, such as using less predicates in the formulas than expected (which is a shortcut solution to this task and this indicates that task one should be adapted to avoid this).

For task two (create world model), a drop in performance is observed, and for task three (deduce sat), the drop is so stark that the models only perform according to the random baseline. This indicates much scope for future work.

6.2 Future Research

Our work and the obtained results suggest different directions for future work in order to further improve the logical reasoning skills of large language models. We can adapt aspects of the dataset / tasks, create another baseline, change aspects of finetuning, inquire further into generalization, or even use new training methods such as RLMF. In more detail:

First, some changes to our approach are recommended. Two changes to the tasks are necessary: on the one hand, since we learned that there exists a trivial solution to task one (create formula), it is advised to adapt task one (create formula) to ask for formulas that are neither tautologies nor contradictions. Furthermore, task one (create formula) should be reformulated to include that the model needs to use all predicates from the world model (else an easy solution is possible on this task). On the other hand, the training dataset should exclude tautologies and contradictions as well, so that we do not ask any trivial questions for task two (create world model), and so that we do not train on tautologies and contradictions for task one (create formula).

Also, the bias with regards to the creation of label-formula pairs, that we discussed in section 5.1, should be eliminated so to avoid that language models learn a potential shortcut. A solution would be to create hard instances, analogous to (Richardson and Sabharwal, 2022). Their paper (Richardson and Sabharwal, 2022) bases their selection of formulas in propositional logic on findings by Selman et. al (Selman et al., 1996), who find that the probability for satisfying such a formula is either very high or very low given a certain number of variables per clause. This is how they eliminate easy instances and only include hard instances in their dataset. For our task, this means to only include certain formula-world-model-size-combinations, where the probability that the target label is satisfied is close to (the critical region of) 0.5 (compare to 5.1). So for example, in table 5.2, the world model size would change according to the type of formula (smaller for less complex formulas, and larger for more complex formulas). For example, for the formula

$$allx.(F(x) \rightarrow G(x)).$$

this would mean to create a world model with either two or three constants (but not more or less). Note that hard instances could not include world models larger (or smaller than that, as those would be cues that the models could use for prediction (possibly not learning to emulate logic). However, we should be aware that for task two (create world model), there will always be a shortcut of creating a random world model with a certain

size that will lead to a high accuracy (but not necessarily demonstrate an ability to emulate logical reasoning)! Therefore, we could also adapt task two by asking for a certain world model size (or number of constants) in the output.

Other tasks are also conceivable using our approach and could be investigated: for example, we could ask the model to create one formula that is correct and another formula that is incorrect for a given world model (an adaptation to task one - create formula), or to create a world model that satisfies the formula and another one such that the formula is not satisfied (an adaptation to task two - create world model). We are proposing here to create more tasks that are very closely related as they all probe logical reasoning skills in predicate logic. This could be called task-coherent datasets.

Besides, we could even add datapoints to our dataset that have multiple targets for one input or multiple inputs for one target: this is because the order of sentences in the world model and the key mapping is irrelevant for the respective tasks, and this is something, the large language models would need to learn also, as for them, order is generally relevant.

Second, it would also be interesting to compare the language models against the human performance on these tasks, thereby creating another baseline.

Third, further finetuning could help in many ways. One way is to perform a hyperparameter search for the LoRA adapters and see how good our individual models can eventually get during training (as there is still room for improvement on our proposed tasks). Other ways would be to finetune without LoRA, or to finetune for more training epochs. Moreover, we opted for doing many experiments with fewer datapoints over fewer experiments with more datapoints. But the language models could be finetuned on a much bigger dataset, that can be automatically created. Besides, another way is to finetune our models on all three tasks (multitask setting) but with as many datapoints as in the singletask setting, and see whether the performance compares; or, the other way around, i.e. training for the singletask setting but with as few datapoints as for the multitask setting.

Fourth, there are more avenues to be explored regarding generalization. First, in order to attain an even better generalization to a harder dataset, we could try curriculum learning (Bengio et al., 2009). In curriculum learning, a model is trained on an easy task first and then a harder task afterwards, and as a result, it generalizes much better. Another avenue for further testing generalization would be to systematically construct harder and harder datasets in a principled manner. We only created one such harder dataset, but many such datasets can be created along many different dimensions (e.g. more predicates, more quantifiers, more operators, larger world models, etc.). What is more, we could change the grammar of our tasks to see how well the models generalize to a new grammar setting. And probably a most interesting test would be to inquire whether training on our synthetic dataset transfers to higher performance on real world data (or other reasoning tasks, generally).

Fifth, as we already devised in chapter 4, our dataset is suitable for finetuning using reinforcement learning from machine feedback (RLMF; instead of from human feedback), so we could use it with this methodology in the future. For instance, it has been shown in training code generating language models, that RLMF works (Le et al., 2022).

In summary, there remains significant scope for future research regarding teaching large language models logical reasoning in predicate logic. While we have shown that language models are capable of emulating reasoning in predicate logic above random baselines, robust scaling to harder tasks remains to be solved. That is why we may have possibly designed some hard logical reasoning tasks here.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *CoRR*, abs/2012.13255, 2020. URL https://arxiv.org/abs/2012.13255. 22

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 14

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022. 54

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard [Accessed: (08th September 2023)], 2023. 1, 54

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*, pages 610–623, 2021. 2

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, mar 2003. ISSN 1532-4435. 17, 19, 37

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 86

Gregor Betz, Christian Voigt, and Kyle Richardson. Critical thinking for language models. *arXiv preprint arXiv:2009.07185*, 2020. xi, 2, 30, 32, 33

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009. 24, 39

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 20, 48, 53

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter,

REFERENCES

Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374. 54

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022. 53

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018. URL http://arxiv.org/abs/1803.05457. 54

Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*, 2020. xi, 29, 30, 31

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(76): 2493–2537, 2011. URL http://jmlr.org/papers/v12/collobert11a.html. 16

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359, 2022. 53, 54

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023. 23, 112

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 17

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018. 19

Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. 23

L.T.F. Gamut. *Logic, Language, and Meaning, Volume 1: Introduction to Logic*. Logic, Language, and Meaning. University of Chicago Press, 1991. ISBN 9780226280851. 23, 27

Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2018.12.010. URL https://www.sciencedirect.com/science/article/pii/S2352154618301943. Artificial Intelligence. 7

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016. 9

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013. 21

Eric M Hammer. Semantics for existential graphs. *Journal of Philosophical Logic*, 27:489–503, 1998. 23

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954. 16

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 14

Chadi Helwe, Chloé Clavel, and Fabian Suchanek. Reasoning with transformer-based models: Deep learning, but shallow reasoning. In *International Conference on Automated Knowledge Base Construction (AKBC)*, 2021. 29

Chadi Helwe, Chloé Clavel, and Fabian Suchanek. Logitorch: A pytorch-based library for logical reasoning on natural language. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 250–257, 2022. 29

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 10

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 8

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019. 19

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019. 20

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685. xi, 21, 22, 112

Zhijing Jin, Abhinav Lalwani, Tejas Vaidhya, Xiaoyu Shen, Yiwen Ding, Zhiheng Lyu, Mrinmaya Sachan, Rada Mihalcea, and Bernhard Schölkopf. Logical fallacy detection. *arXiv preprint arXiv:2202.13758*, 2022. 29

Dan Jurafsky and James H Martin. Speech and language processing (3rd (draft) ed.), 2022. xi, 8, 9, 10, 11, 12, 13, 16, 17, 18, 19

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011. ISBN 978-0-374-27563-1. 2

Aishwarya Kamath and Rajarshi Das. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978*, 2018. 34

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 8

REFERENCES

Nora Kassner and Hinrich Schütze. Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7811–7818, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.698. URL https://aclanthology.org/2020.acl-main.698. 1

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022. 35, 87

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019. URL http://arxiv.org/abs/1910.13461. 18

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015. 16

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023. 54

Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. Isarstep: a benchmark for high-level mathematical reasoning. *arXiv preprint arXiv:2006.09265*, 2020. 29

Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*, pages 6565–6576. PMLR, 2021. 37

Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. Generated knowledge prompting for commonsense reasoning. *arXiv preprint arXiv:2110.08387*, 2021. 29

Nicholas Lourie, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Unicorn on rainbow: A universal commonsense reasoning model on a new multitask benchmark. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13480–13488, 2021. 1

Donald W Loveland. Automated theorem proving: mapping logic into ai. In *Proceedings of the ACM SIGART international symposium on Methodologies for intelligent systems*, pages 214–229, 1986. 28

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023. 52, 54

Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and Evelina Fedorenko. Dissociating language and thought in large language models: a cognitive perspective. *arXiv preprint arXiv:2301.06627*, 2023. 1

Minsky Marvin and A Papert Seymour. Perceptrons. *Cambridge, MA: MIT Press*, 6:318–362, 1969. 7

Pankaj Mathur. Orca mini v3 13b: An orca style llama2-70b model, 2023. URL https://https://huggingface.co/psmathur/orca_mini_v3_13b. 52

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943. 7

Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Javad Hosseini, Mark Johnson, and Mark Steedman. Sources of hallucination by large language models on inference tasks. *arXiv preprint arXiv:2305.14552*, 2023. 1, 37

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010. 10

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011. 17

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013. 16

George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956. 48

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023. 54

Niels Mündler, Jingxuan He, Slobodan Jenko, and Martin Vechev. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation. *arXiv preprint arXiv:2305.15852*, 2023. 1, 37

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022. 37

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023. 52, 53

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 1, 18, 54

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020. 53

Inioluwa Deborah Raji, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. AI and the everything in the whole wide world benchmark. *CoRR*, abs/2111.15366, 2021. URL `https://arxiv.org/abs/2111.15366`. 1

Kyle Richardson and Ashish Sabharwal. Pushing the limits of rule reasoning in transformers through natural language satisfiability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11209–11219, 2022. xi, 30, 31, 32, 48, 85

REFERENCES

Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020. 29

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 8, 19

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010. 7

Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. Prover: Proof generation for interpretable reasoning over rules. *arXiv preprint arXiv:2010.02830*, 2020. 29, 34

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019. 29

Abigail See, Stephen Roller, Douwe Kiela, and Jason Weston. What makes a good conversation? how controllable attributes affect human judgments. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1702–1723, 2019. 17

Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1):17–29, 1996. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(95)00045-3. URL https://www.sciencedirect.com/science/article/pii/0004370295000453. Frontiers in Problem Solving: Phase Transitions and Complexity. 85

Louis Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope, and Ray Kurzweil. Generating high-quality and informative conversation responses with sequence-to-sequence models. *arXiv preprint arXiv:1701.03185*, 2017. 19

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150, 2019. URL http://arxiv.org/abs/1911.02150. 53, 54

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022. 2, 33

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. *arXiv preprint arXiv:2012.13048*, 2020. 29

Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, et al. Ul2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2022a. 52, 53

Yi Tay, Jason Wei, Hyung Won Chung, Vinh Q Tran, David R So, Siamak Shakeri, Xavier Garcia, Huaixiu Steven Zheng, Jinfeng Rao, Aakanksha Chowdhery, et al. Transcending scaling laws with 0.1% extra compute. *arXiv preprint arXiv:2210.11399*, 2022b. 19, 37

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 37, 52, 53

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423, 14602113. URL http://www.jstor.org/stable/2251299. 1

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. xi, 11, 15

Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015. doi: 10.1109/JPROC.2015.2455034. 32

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021. 20, 48

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022. 2

Ludwig Wittgenstein. *Philosophical Investigations*. Basil Blackwell, Oxford, 1953. ISBN 0631119000. 16

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6. 54, 111

Yotam Wolf, Noam Wies, Yoav Levine, and Amnon Shashua. Fundamental limitations of alignment in large language models. *arXiv preprint arXiv:2304.11082*, 2023. 37

Nathan Young, Qiming Bao, Joshua Bensemann, and Michael Witbrock. Abductionrules: Training transformers to explain unexpected inputs. *arXiv preprint arXiv:2203.12186*, 2022. 29

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. Ar-lsat: Investigating analytical reasoning of text. *arXiv preprint arXiv:2104.06598*, 2021. 29

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul F. Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *CoRR*, abs/1909.08593, 2019. URL http://arxiv.org/abs/1909.08593. 37

Appendix

A Illustrative items of the synthetic base dataset

For each task, we tested two different prompts. The following illustrative items of the proposed synthetic dataset follow templates, that are built with placeholders (curly braces) for input and output.

One example for placeholders could be:

Formula:
$all\ x.F(x) \rightarrow G(x)$

Model:
Peter is hungry.
Peter is tall.
Sue is not hungry.
Sue is tall.

Satisfied?:
satisfied

Keys:
F: hungry
G: tall
a: Peter
b: Sue

A.1 Task 1: Create Formula

Prompt 1:

Input:
"Here is a world model:
{{ world model }}
Let us interpret predicates and names as follows:
{{ key mapping }}
Provide a logical formula in predicate logic that is {{ satisfied }} given the above situation and interpretation (keys)! "

Expected Target:
“ {{ formula }} “

Prompt 2:

Input:
Context:
You will receive a problem in first-order predicate logic to solve. This problem contains a set of statements about the world (let us call it a 'world model') and a mapping from things in this world to variables representing these things (we call that mapping the 'keys'). You are then asked to provide a formula in first-order predicate logic that is either satisfied or unsatisfied given the world model and keys. Please return only the formula, written in the format of the python library nltk.
Question:
Here is a world model:
{{ world model }}
Here are the keys:
{{ key mapping }}
Please write down only one formula in first-order predicate logic that is {{ satisfied }} given the above world model and keys.
Answer:

Expected Target:
“ {{ formula }} “

A.2 Task 2: Create world model

Prompt 1:

Input:
“Consider the following formula:
{{ formula }}
describe a situation in which the formula is {{ satisfied }} and provide the keys! “

Expected Target:
“ Situation:
{{ world model }}
Keys:
{{ key mapping }}
“

Prompt 2:

Input:
Context:
You will receive a problem in first-order predicate logic to solve. This problem contains a formula in first-order predicate logic, written in the format of the python library nltk for which you are asked to create a 'world model' and 'keys' that either satisfy or do not satisfy the formula. A world model is a set of statements about whether one or more predicates apply to things in this world. Keys are a mapping from things to (lower case) variables and predicates to (upper case) variables.
Question:

Consider the following formula:
{{ formula }}
Provide a world model and keys such that the formula is {{ satisfied }}
Please answer by returning the world model starting with 'World model:', followed by the keys starting with 'Keys:'."
Answer:

Expected Target:
“ Situation:
{{ world model }}
Keys:
{{ key mapping }}
“

A.3 Task 3: Deduce sat

Prompt 1:

Input:
“Consider the following formula in first-order predicate logic:
{{ formula }}
Let us interpret predicates and names as follows:
{{ key mapping }}
Also, here is a world model:
{{ world model }}
Is the provided formula satisfied or not given the situation? ('satisfied' or 'unsatisfied?'). “

Expected Target:
“ {{ satisfied }} “

Prompt 2:

Input:
Context:
You will receive a problem in first-order predicate logic to solve. This problem contains a formula in first-order predicate logic, written in the format of the python library nltk a set of statements about a world ('world model') and a mapping ('keys') from things in this world to variables representing these things. Please return in one word whether the formula is satisfied or unsatisfied given the world model and keys.
Question:
Consider the following formula:
{{ formula }}
Here is a world model:
{{ world model }}
Here are the keys:
{{ key mapping }}
Is the provided formula satisfied or unsatisfied given the world model and keys?
Answer:

Expected Target:

“ {{ satisfied }} “

B Dataset statistics

B.1 Fewshot and Zershot evaluation

Regarding the evaluation dataset for zeroshot and fewshot learning, we observe 507 formulas that are satisfied, 493 that are unsatisfied. We have 503 formulas that use one exist quantifier, 497 formulas that use one all quantifier. 552 formulas use a single predicate (either F or G), and 448 formulas use two predicates. 457 formulas use the implication operator, and 470 formulas use the and operator. There were 497 formulas that had the negation operator in front. Hence, we have a very balanced evaluation dataset in these regards.

Furthermore, 56 formulas only use a unary operator (i.e. the negation), while 48 formulas use only a binary operator (either the and operator or the implication operator); the other 896 formulas use both unary and binary operators. There are 123 tautologies and 120 contradictions in this dataset.

Finally, here are the graphs showing counts of (1) how many constants we use, (2) how many keys we use, (3) how big our world models are (in terms of the number of sentences), (4) how many negations there are and (5) how many times we observe a certain total number of operators.

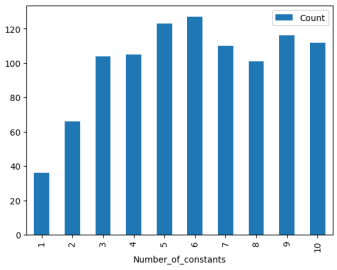


Figure 6.1: Number of datapoints that have a certain number of constants. Fewshot and Zeroshot.

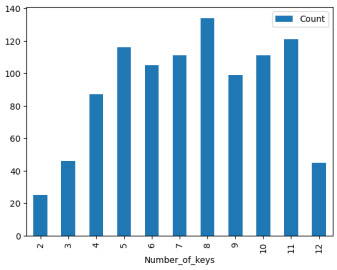


Figure 6.2: Number of datapoints that have a certain number of keys. Fewshot and Zeroshot.

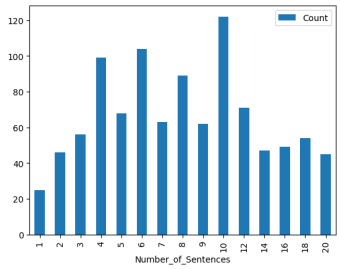


Figure 6.3: Number of datapoints that have a certain number of sentences (world model size). Fewshot and Zeroshot.

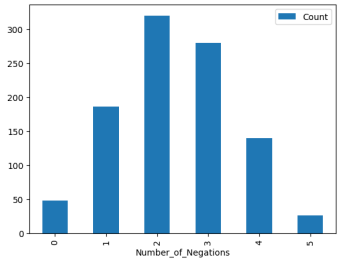


Figure 6.4: Number of datapoints that have a certain number of negations. Fewshot and Zeroshot.

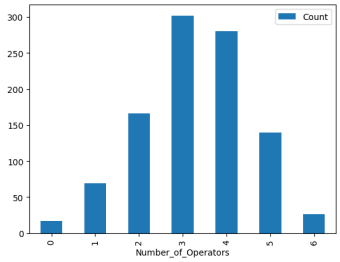


Figure 6.5: Number of datapoints that have a certain total number of operators. Fewshot and Zeroshot.

B.2 Training evaluation

The training evaluation dataset uses a different subset of 1000 datapoints from the base dataset, so we analyze it separately. Note, however that it does not differ in a significant way from the zershot and fewshot evaluation dataset: we observe 505 formulas that are satisfied, 495 that are unsatisfied. We have 503 formulas that use one exist quantifier, 497 formulas that use one all quantifier. 543 formulas use a single predicate (either F or G), and 457 formulas use two predicates. 468 formulas use the implication operator, and 474 formulas use the and operator. There were 494 formulas that had the negation operator in front. Hence, we have a very balanced evaluation dataset in these regards.

Furthermore, 43 formulas only use a unary operator (i.e. the negation), while 45 formulas use only a binary operator (either the and operator or the implication operator); the other 912 formulas use both unary and binary operators. There are 123 tautologies and 118 contradictions in this dataset. So, we

observe that this dataset does not significantly differ from the evaluation dataset in the fewshot and zeroshot setting.

Finally, here are the graphs showing counts of (1) how many constants we use, (2) how many keys we use, (3) how big our world models are (in terms of the number of sentences), (4) how many negations there are and (5) how many times we observe a certain total number of operators.

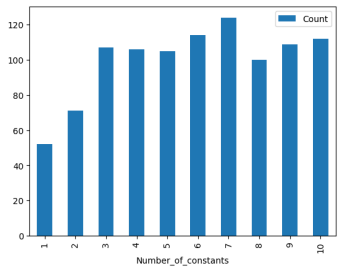


Figure 6.6: Number of datapoints that have a certain number of constants. Training evaluation.

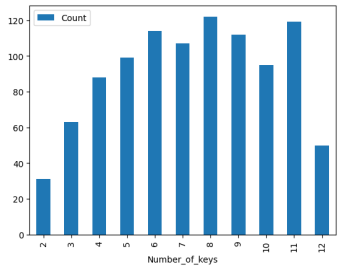


Figure 6.7: Number of datapoints that have a certain number of keys. Training evaluation.

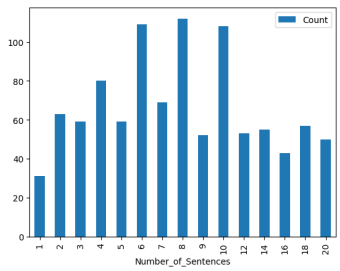


Figure 6.8: Number of datapoints that have a certain number of sentences (world model size). Training evaluation.

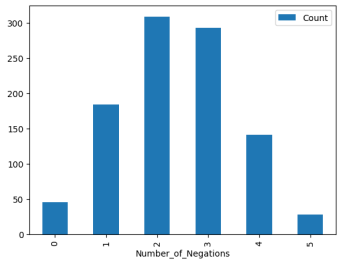


Figure 6.9: Number of datapoints that have a certain number of negations. Training evaluation.

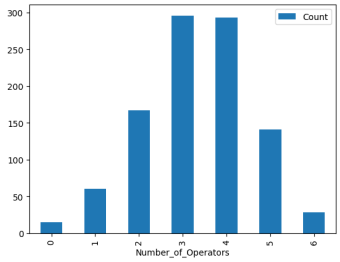


Figure 6.10: Number of datapoints that have a certain total number of operators. Training evaluation.

B.3 Hard evaluation

The hard evaluation dataset is harder than the training evaluation dataset, as we use more predicates, more quantifiers and more operators. That is, we have both one "exists" and one "all" quantifier in each formula. We have 108 formulas with only one predicate, 672 formulas with two predicates and 220 formulas with three predicates (F, G and H). 255 formulas have no implication, 533 formulas have one implication and 212 formulas have two implications. 217 formulas have no and operator, 531 formulas have one and operator and 252 formulas have two and operators. This has as a consequence, that we also have no formulas with just unary operators (negations), only 6 formulas with only binary operators (implications or and operators), but 994 formulas that have both unary and binary operators.

It follows that there appear more negations and more total operators. As a consequence of using more predicates, there are more world models that are longer as compared to the training dataset. It also means that the number of keys rises whenever we use more operators, otherwise it stays comparable to the training dataset. At the same time, we keep the number of constants similar to the training dataset. All of this can be seen from the following figures.

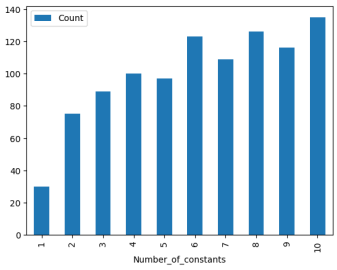


Figure 6.11: Number of datapoints that have a certain number of constants. Hard dataset.

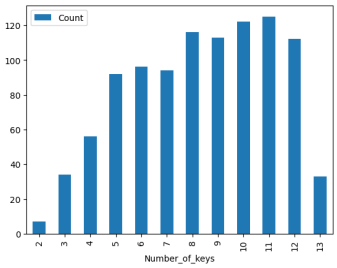


Figure 6.12: Number of datapoints that have a certain number of keys. Hard dataset.

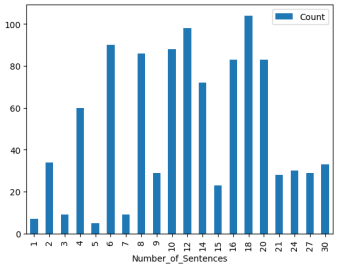


Figure 6.13: Number of datapoints that have a certain number of sentences (world model size). Hard dataset.

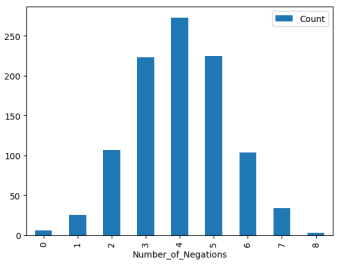


Figure 6.14: Number of datapoints that have a certain number of negations. Hard dataset.

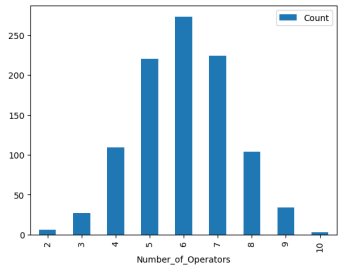


Figure 6.15: Number of datapoints that have a certain total number of operators. Hard dataset.

We maintain balance in this dataset in the labels and the negations in front of a formula: 504 formula-world model combinations are unsatisfied, while 496 are satisfied. 499 formulas have no negation in front, while 501 have a single negation in front. Finally, in order to measure the number of tautologies and contradictions, we had to introduce a timeout (which we set to two seconds), which means that for 67 formulas, it could not be decided whether they were a tautology or a contradiction (or neither). For the remaining formulas, 111 were tautologies and 100 were contradictions.

B.4 Training on a single task

Regarding the training dataset for the single task setting, we observe 3037 formulas that are satisfied, 2963 that are unsatisfied. We have 2954 formulas that use one exist quantifier, 3046 formulas that use one all quantifier. 3266 formulas use a single predicate (either F or G), and 2734 formulas use two predicates. 2769 formulas use the implication operator, and 2860 formulas use the and operator. There were 3009 formulas that had one negation operator in front, while 2991 did not. Hence, we have a very balanced training dataset in these regards.

Furthermore, 278 formulas only use a unary operator (i.e. the negation), while 278 formulas use only a binary operator (either the and operator or the implication operator); the other 5444 formulas use both unary and binary operators. There are 706 tautologies and 742 contradictions in this dataset.

Finally, here are the graphs showing counts of (1) how many constants we use, (2) how many keys we use, (3) how big our world models are (in terms of the number of sentences), (4) how many negations there are and (5) how many times we observe a certain total number of operators.

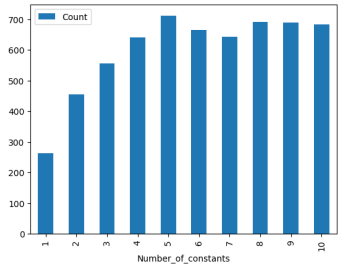


Figure 6.16: Number of datapoints that have a certain number of constants. Finetuning.

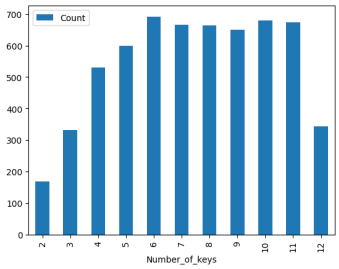


Figure 6.17: Number of datapoints that have a certain number of keys. Finetuning.

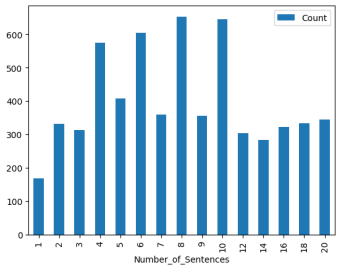


Figure 6.18: Number of datapoints that have a certain number of sentences (world model size). Finetuning.

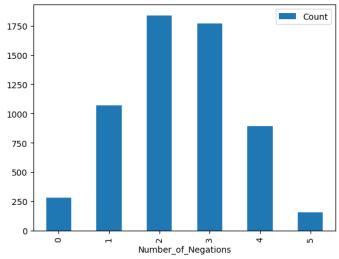


Figure 6.19: Number of datapoints that have a certain number of negations. Finetuning.

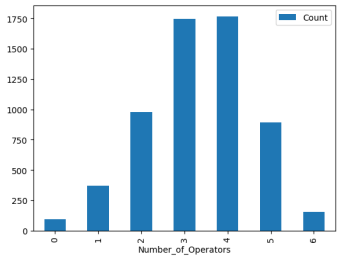


Figure 6.20: Number of datapoints that have a certain total number of operators. Finetuning.

B.5 Training on multiple tasks

Regarding the training dataset for the multi task setting, we observe 1204 formulas that are satisfied, 1196 that are unsatisfied. We have 1179 formulas that use one exist quantifier, 1221 formulas that use one all quantifier. 1302 formulas use a single predicate (either F or G), and 1098 formulas use two predicates. 1117 formulas use the implication operator, and 1141 formulas use the and operator. There were 1198 formulas that had one negation operator in front, while 1202 did not. Hence, we have a very balanced training dataset in these regards.

Furthermore, 99 formulas only use a unary operator (i.e. the negation), while 119 formulas use only a binary operator (either the and operator or the implication operator); the other 2182 formulas use both unary and binary operators. There are 276 tautologies and 293 contradictions in this dataset.

Finally, here are the graphs showing counts of (1) how many constants we use, (2) how many keys we use, (3) how big our world models are (in terms of the number of sentences), (4) how many negations there are and (5) how many times we observe a certain total number of operators.

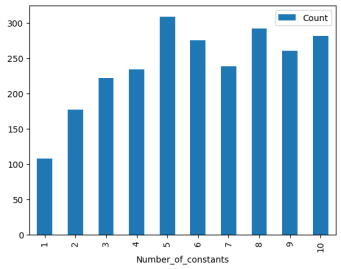


Figure 6.21: Number of datapoints that have a certain number of constants.

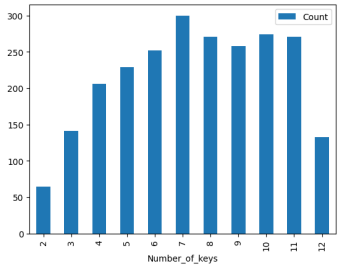


Figure 6.22: Number of datapoints that have a certain number of keys.

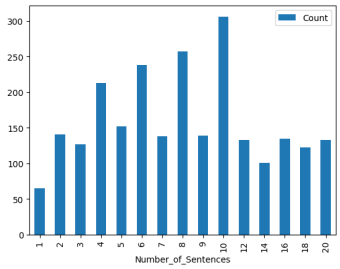


Figure 6.23: Number of datapoints that have a certain number of sentences (world model size).

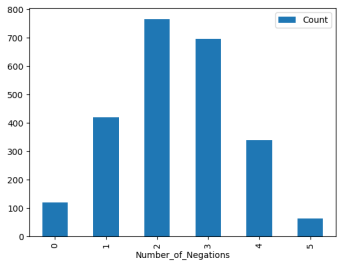


Figure 6.24: Number of datapoints that have a certain number of negations.

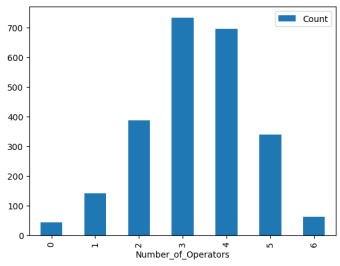


Figure 6.25: Number of datapoints that have a certain total number of operators.

C Results for Prompt 2.

Here we present the results of the fewshot experiments with prompt 2.

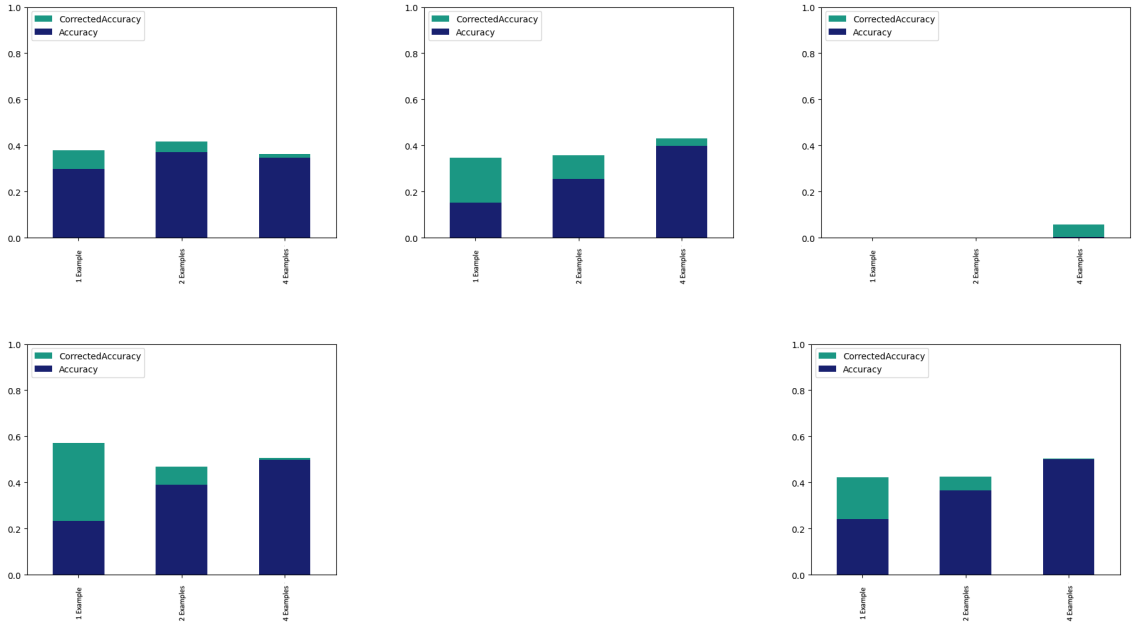


Figure 6.26: Fewshot performance on task 1 (create formula). Prompt 2.
Top: Left: Falcon-7b; Middle: Llama-13b; Right: Flan-Ul2.
Bottom: Left: Orca-13b; Right: Wizard-15b.

Table 6.1: Fewshot accuracies for task 1 (create formula). One, two or four examples. Prompt 2.

Model	1 example	2 examples	4 examples
Flan-UL2 (20B)	0.0 (0.0)	0.0 (0.0)	0.002 (0.057)
Falcon-7B	0.299 (0.379)	0.370 (0.417)	0.346 (0.363)
Llama-13B-Chat-hf	0.152 (0.346)	0.255 (0.359)	0.398 (0.429)
Orca-13B	0.233 (0.571)	0.389 (0.469)	0.499 (0.506)
Wizard-15B	0.240 (0.422)	0.365 (0.426)	0.500 (0.503)

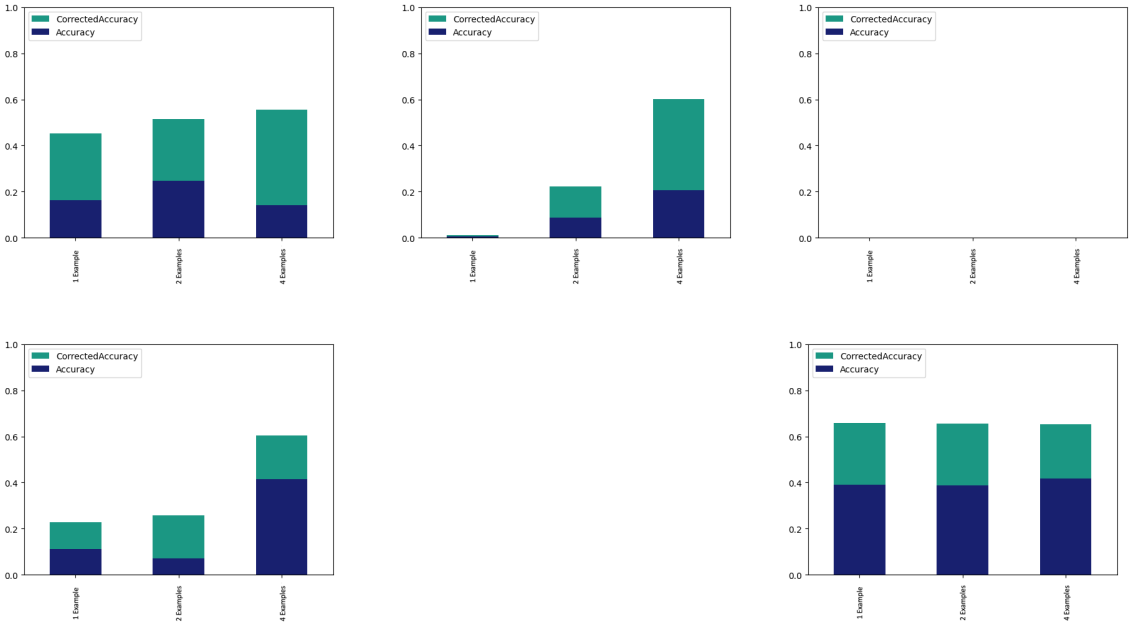


Figure 6.27: Fewshot performance on task 2 (create world model). Prompt 2.
Top: Left: Falcon-7b; Middle: Llama-13b; Right: Flan-Ul2.
Bottom: Left: Orca-13b; Right: Wizard-15b.

Table 6.2: Fewshot accuracies for task 2 (create world model). One, two or four examples. Prompt 2.

Model	1 example	2 examples	4 examples
Flan-UL2 (20B)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
Falcon-7B	0.162 (0.451)	0.247 (0.516)	0.142 (0.555)
Llama-13B-Chat-hf	0.007 (0.013)	0.087 (0.224)	0.207 (0.602)
Orca-13B	0.112 (0.227)	0.071 (0.257)	0.416 (0.605)
Wizard-15B	0.391 (0.658)	0.388 (0.657)	0.418 (0.654)

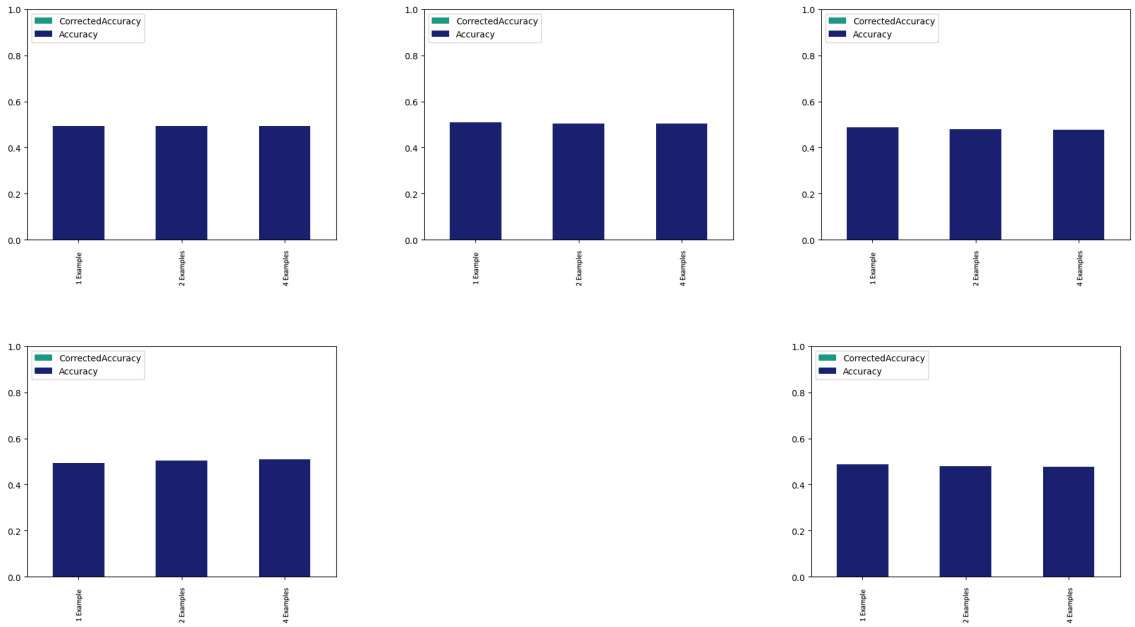


Figure 6.28: Fewshot performance on task 3 (deduce sat). Prompt 2.
Top: Left: Falcon-7b; Middle: Llama-13b; Right: Flan-Ul2.
Bottom: Left: Orca-13b; Right: Wizard-15b.

Table 6.3: Fewshot accuracies for task 3 (deduce sat). One, two or four examples. Prompt 2.

Model	1 example	2 examples	4 examples
Flan-UL2 (20B)	0.488 (0.488)	0.481 (0.481)	0.478 (0.478)
Falcon-7B	0.493 (0.493)	0.493 (0.493)	0.493 (0.493)
Llama-13B-Chat-hf	0.510 (0.510)	0.504 (0.504)	0.503 (0.503)
Orca-13B	0.492 (0.492)	0.504 (0.504)	0.509 (0.509)
Wizard-15B	0.494 (0.494)	0.493 (0.493)	0.495 (0.495)

No figures for task three (deduce sat) are shown, because they are all close to 0.5.

D Training Hyperparameters

Our training hyperparameters for each of the models is depicted in table 6.4. Whatever hyperparameter is not listed there, means that we use the default settings from the huggingface library Wolf et al. (2020). Plus, we use the Autotokenizers and AutoModelForCausalLM classes (or the T5ForConditionalGeneration class in the case of the sequence-to-sequence model). We also generally restrict our models to generate only up to a maximum of 200 tokens (if they do not produce the stop token themselves).

In order to understand the table, here are some explanations of the relevant concepts:

- The number of training epochs refer to the number of times that the finetuning goes through the entire input dataset.

- Weight decay adds a small penalty to the loss function, thereby aiming to prevent overfitting on the training data. Overfitting refers to a model fitted so well on the training data that it does not generalize well to unseen test data.
- The learning rate determines how big of a step we move towards the minimum of the loss function when optimizing.
- A checkpoint is a step during finetuning that is saved (e.g. so that we can continue training at some future moment in time or to compare how training outcomes evolve over time. Therefore, the number of checkpoints saved then refers to the total number of checkpoints that we saved, keeping the best performing ones.
- Quantization, double quantization and LoRA r have been discussed in section 4.
- The LoRA dropout is the dropout applied to the LoRA adapter. Dropout refers to randomly excluding a certain number of nodes while training Hu et al. (2021).
- The LoRA Alpha is a factor that scales how much we update the LoRA matrices Hu et al. (2021).
- The NF4 (Normal Float 4) data type is a special data type to represent data (e.g. language model weights) in 4 bits Dettmers et al. (2023).

Table 6.4: Training hyperparameters for the respective models in the single task setting (for all three individual tasks).

Hyperparameter	Falcon-7B	Llama-13B	Orca-13B	Wizard-15B
Number of Training epochs	10	10	10	10
Weight decay	0.01	0.01	0.01	0.01
Learning rate	$2e^{-5}$	$2e^{-5}$	$2e^{-5}$	$2e^{-5}$
Number of checkpoints saved	3	3	3	3
Generation restriction	max new tokens = 200	max new tokens = 200	max new tokens = 200	max new tokens = 200
4-bit quantization	True	True	True	True
Double Quantization	True	True	True	True
NF4 (Normal Float 4) data type	True	True	True	True
LoRA r	16	8	16	16
LoRA alpha	32	16	32	32
LoRA Dropout	0.05	0.0	0.05	0.05
LoRA applied to matrices:	"query-key-value"	"q-proj", "k-proj", "v-proj", "o-proj"	"q-proj", "k-proj", "v-proj", "o-proj", "gate-proj", "up-proj", "down-proj"	"c-attn", "c-proj", "c-fc", "c-proj"

Some exceptions were made, e.g. for the Llama model regarding task three (deduce sat), a LoRA dropout of 0.1 was applied (instead of 0.0, which was used for task one (create formula) and task two (create world model)). With regards to single task finetuning, some models were not trained for all ten epochs. For instance, Falcon was only trained for a bit more than 5 epochs for task one (create formula)

and task two (create world model), and for two epochs for task three (deduce sat), instead of 10. Llama was only trained for a bit more than 7 epochs on task three (deduce sat). For multitask finetuning, we trained all four models for only three training epochs, as compared to their hyperparameters for singletask finetuning.