POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Natural Language Reasoning with Transformer Language Models

NICOLAS ANGELARD-GONTIER
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Mai 2023

© Nicolas Angelard-Gontier, 2023.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Natural Language Reasoning with Transformer Language Models

présentée par **Nicolas ANGELARD-GONTIER**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Amal ZOUAQ, présidente
Christopher J. PAL, membre et directeur de recherche
Laurent CHARLIN, membre
Pasquale MINERVINI, membre externe

DEDICATION

To my parents,
for their support...

ACKNOWLEDGEMENTS

First and foremost, I would like to express my most sincere appreciation to my research supervisor, Christopher Pal, for his continuous support, motivation, and guidance throughout my Ph.D. journey. He gave me the opportunity to pursue this degree and provided me with invaluable industry research opportunities.

My sincere thanks also go to my friends and collaborators Koustuv Sinha, Siva Reddy, Pau Rodriguez, and Issam Laradji for their support, essential feedback, and fruitful collaborations throughout the projects included in this thesis.

During my Ph.D., I was fortunate enough to have multiple internships at Element AI and ServiceNow Research during which I learned from and collaborated with incredible people. I would like to thank all my friends and colleagues who made those days enjoyable and fruitful: Raymond Li, Quentin Capart, Cyril Ibrahim, Alexandre Piche, Rafael Pardinas, Gabriel Huang, Torsten Scholak, Dzmitry Bahdanau, Sebastien Paquet, Valérie Bécaert, and Nicolas Chapados. A special thanks to my collaborator and internship manager David Vazquez who motivated me to start writing this thesis early, and trusted me by recommending my full-time application at ServiceNow Research. I am looking forward to continuing to collaborate with all of you.

I would also like to thank my friends from Polytechnique Montréal and Mila for always motivating me with their insightful feedback and stimulating discussions on various research topics: Prasanna Parthasarathi, Quentin Fournier, Jonathan Pilault, and Sandeep Subramanian.

This thesis consists of experiments requiring a significant amount of computing resources. Thus I would like to thank the Mila IDT team and the Toolkit support team at ServiceNow for their dedicated support and technical assistance without which none of the results presented here would exist.

I would also like to extend my sincere gratitude to Professor Joelle Pineau who sparked my interest in Artificial Intelligence during my Bachelor's degree and introduced me to scientific research within the Reasoning and Learning Lab during my Masters at McGill. Furthermore, I would like to thank the great professors I had the privilege to learn from who have helped me foster my research interest in natural language processing and reinforcement learning: Doina Precup, Jackie Cheung, and Timothy O'Donnell. I also want to thank my dear friends from McGill who taught me a lot about research during the time of my Masters: Ryan Razani,

Ryan Lowe, Iulian Serban, and Michael Noseworthy.

Finally, I am grateful to my family for always being there. I thank my parents Christine Angelard and Michel Gontier for supporting me from the very beginning. A very special thanks to my life partner Zoé for her love and companionship. Life is always more fun when we are living it together. I am also grateful for the enthusiasm and curiosity of my family-in-law who always enjoy discussing scientific research from all domains with me: Andréa, Éric, and Téo.

RÉSUMÉ

En raison de la popularité croissante des modèles de langage à base de *Transformers* (TLMs), il est de plus en plus nécessaire de mieux comprendre leurs forces et leurs limites s'ils doivent être utilisés pour aider les humains à résoudre des tâches complexes avec des implications réelles. Cette thèse se concentre particulièrement sur leurs capacités de raisonnement à plusieurs étapes, car il s'agit à la fois d'une faiblesse des modèles de langage et d'une direction de recherche potentiellement impactante.

Tout d'abord, la généralisation compositionnelle des TLMs est évaluée sur une tâche de raisonnement logique en langage naturelle. Des modèles de *Transformers* décodeurs sont entraînés à répondre à des questions de prédiction de lien entre des personnes en raisonnant sur leurs relations intermédiaires. En particulier, pour mieux comprendre comment les TLMs raisonnent, les modèles sont entraînés à générer différents types d'explications en langage naturel (preuves) avant de générer leur réponse finale. L'exactitude des réponses et des preuves sont évaluées sur des problèmes nécessitant un nombre spécifique d'étapes de raisonnement qui ne sont pas vues pendant l'entraînement. Cette première contribution confirme que les TLMs souffrent de problèmes de généralisation lorsqu'ils sont testés sur des problèmes plus longs que ceux pour lesquels ils ont été entraînés. De plus, elle révèle que les TLMs généralisent mieux lorsqu'ils sont entraînés sur des preuves exhaustives et longues que sur des preuves courtes. Les résultats montrent également que les TLMs généralisent mieux lorsqu'ils sont entraînés à générer des chaines de preuves inverse ("*backward-chaining*") plutôt que des chaînes directes ("*forward-chaining*"). Cependant, on observe également que les modèles entraînés à prédire directement la réponse finale sans générer d'explication logique généralisent mieux aux problèmes plus complexes. Cela suggère que les TLMs ont des stratégies de raisonnement interne difficiles à interpréter, et que bénéficier d'énoncés de preuves logiques naturelles nécessite des représentations internes plus complexes. Des expériences additionelles ont d'ailleurs montré que les modèles pré-entraînés ont de meilleures capacités de raisonnement bien qu'ils n'aient pas été explicitement entraînés à résoudre de telles tâches. Cette première contribution est publiée dans les "*Advances in Neural Information Processing Systems (NeurIPS)*" 2020.

La prochaine contribution introduit un biais inductif d'abstraction dans les TLMs pré-entraînés et démontre ses avantages sur des tâches de raisonnement symbolique. Étant donné que la manipulation de concepts génériques simplifie les processus de raisonnement et permet aux humains de généraliser leurs connaissances entre domaines, cette contribution utilise la

reconnaissance de type d'entitées pour augmenter l'information donnée en entrée au modèle. Cinq stratégies sont proposées pour incorporer ces connaissances supplémentaires dans un TLM encodeur-décodeur: deux méthodes basées sur l'*embedding*, deux méthodes basées sur l'encodage et une méthode basée sur une tâche auxiliaire. Les modèles sont évalués sur divers ensembles de données, allant du raisonnement compositionnel, au raisonnement abductif, aux questions-réponses multi-sauts ("*multi-hop*") et aux questions-réponses conversationnelles. Les résultats expérimentaux indiquent que les meilleurs modèles conscients des types d'entitées améliorent les performances des TLMs jusqu'à 20% sur les tâches exigeant explicitement un raisonnement symbolique, confirmant ainsi les avantages de ce biais inductif. Cependant, la méthode d'abstraction proposée n'est pas aussi efficace sur les tâches plus axées sur le langage naturel. Une analyse plus approfondie suggère que l'abstraction des types d'entitées n'est bénéfique que pour les tâches avec (1) des d'abstraction de bonne qualité et (2) une répartition des données d'entraînement/test en fonction de la complexité de raisonnement de chaque exemple. Cette deuxième contribution est publiée dans les "*Transactions on Machine Learning Research*" (TMLR).

Enfin, avec la croissance des interfaces de discussions, la troisième contribution se tourne vers les environnements textuels interactifs. Ces environnements nécessitent que le modèle effectue un raisonnement à plusieurs étapes, car leur tâche consiste à d'atteindre un objectif final en générant des commandes textuelles pour interagir avec l'environnement et évoluer petit à petit vers cet objectif final. Dans le but de mieux contrôler le comportement des TLMs dans ces environnements, la dernière contribution propose une méthode d'apprentissage par renforcement qui tire parti des TLMs pré-entraînés et les conditionne à un résultat souhaité. Les expériences sur certains des jeux Jericho les plus difficiles montrent que les TLMs peuvent apprendre une correspondance entre une condition de résultat et une action, et confirment ainsi l'avantage significatif de l'utilisation de l'inclinaison exponentielle ("*exponential tilt*") lorsque le modèle génère sa propre condition de résultat. De plus, plusieurs méthodes de conditionnement sont proposées et comparées les unes aux autres. Les résultats montrent que les méthodes proposées peuvent améliorer les performances moyennes jusqu'à 10% par rapport aux méthodes précédentes. Finalement, en exploitant l'utilisation des TLMs dans les environnements textuels, des expériences supplémentaires montrent que les modèles entraînés à prédire les conséquences de leurs actions améliorent également les performances moyennes de 10%.

En résumé, cette thèse tente d'éclairer les capacités de raisonnement à plusieurs étapes des modèles de langage à base de *Transformers* et présente de nouveaux mécanismes pour construire des modèles de langage plus logiques et plus contrôlables.

ABSTRACT

Due to the growing popularity of Transformer Language Models (TLMs), there is an increasing need to better understand their strengths and limitations if they are to be widely used to help humans solve complex tasks with real-world implications. This thesis is particularly centered around their multi-step reasoning capabilities as it is both a weakness of language models and a potentially impactful research direction.

First, the compositional generalization of TLMs is evaluated on a logical reasoning task in natural language. Transformer decoder models are trained to answer link-prediction questions by reasoning over relationships between entities. In particular, to better understand how TLMs reason, models are trained to generate various types of natural language explanations (*proofs*) before generating their final answer. Both the models' answer accuracy and proof accuracy are evaluated on problems requiring specific numbers of reasoning steps that are not seen during training. This first contribution confirms that TLMs suffer from length-generalization issues when tested on longer-than-trained problems. Additionally, it reveals that TLMs generalize better when trained on longer, exhaustive proofs than with shorter ones. Results also show that TLMs generalize better when trained to generate backward-chaining rather than forward-chaining proofs. However, it is also observed that models trained to predict the answer directly without generating a logical explanation generalize better to more complex problems. This suggests that TLMs have internal reasoning strategies that are hard to interpret and that benefiting from naturally stated logical proof statements requires more complex internal representations. Additional experiments showed for instance that pre-trained models have better reasoning capacities although not explicitly trained to solve such tasks. This first contribution is published as a conference paper in the Advances in Neural Information Processing Systems (NeurIPS) 2020.

The next contribution introduces an abstraction inductive bias into pre-trained TLMs and demonstrates its benefits on symbolic reasoning tasks. As manipulating generic concepts simplifies reasoning processes and allows humans to generalize knowledge across domains, this contribution makes use of named entity recognition to label entity types in input sequences. Five strategies are proposed to incorporate this additional knowledge into an encoder-decoder TLM: two embedding-based methods, two encoding-based methods, and one auxiliary-loss-based method. Models are evaluated on various reasoning datasets ranging from compositional reasoning, abductive reasoning, multi-hop question answering, and conversational question answering. Experimental results indicate that the best entity-type

abstraction-aware models improve the performance of TLMs by up to 20% on tasks explicitly requiring symbolic reasoning thus confirming the advantages of this inductive bias. However, the proposed abstraction method is not as effective on more natural language tasks. Further analysis suggests that entity-type abstraction is only beneficial in tasks with (1) good quality abstraction labels and (2) with train/test data split according to the reasoning complexity of each example. This second contribution is published as a journal paper in the Transactions on Machine Learning Research (TMLR).

Finally, with the increasing prevalence of chat interfaces, the third contribution moves away from single-turn question-answering tasks and towards interactive text environments. These environments require the model to perform multi-step reasoning by design as the goal is to reach a final objective by generating text commands to interact with the environment and evolve toward that final goal step by step. In an effort to better control the behavior of TLMs in those environments, the last contribution proposes an offline reinforcement learning method that leverages pre-trained TLMs and condition them on a desired outcome. Experimental results on some of the most challenging Jericho text games show that TLMs can learn a mapping from goal condition to action, and thus confirm the significant advantage of using exponential tilt when the model is generating its own outcome condition. Furthermore, multiple conditioning methods are proposed and compared against each other. Results show that the proposed methods can improve average performance by up to 10% over previous baselines. Eventually, taking advantage of the use of TLMs in text environments, additional experiments demonstrate that models trained to predict the consequences of their actions also improve the averaged normalized performance by 10%.

In summary, this thesis attempts to shed light on the multi-step reasoning abilities of Transformer language models and introduces novel mechanisms to build more logical and controllable language models.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ vi

ABSTRACT viii

TABLE OF CONTENTS x

LIST OF TABLES xiv

LIST OF FIGURES xvii

LIST OF SYMBOLS AND ACRONYMS xx

LIST OF APPENDICES xxii

CHAPTER 1 INTRODUCTION 1

1.1 The Field of Research 1

1.2 Motivation 2

1.3 Research Objectives 3

1.4 Methodology 5

1.5 Thesis Outline 7

CHAPTER 2 LITERATURE REVIEW 8

2.1 Technical Background 8

2.1.1 Large Language Models (LLMs) 13

2.1.2 Reinforcement Learning 14

2.2 Systematic Generalisation 15

2.2.1 Datasets 15

2.2.2 Case Studies in NLI and Semantic Parsing 18

2.3 Neural Theorem Provers 19

2.4 Graph Based Methods To Reason Over Text 23

2.5 Knowledge Augmented Language Models 24

2.5.1 Knowledge-Graph Augmented Language Models 24

2.5.2 Memory Based Networks 26
2.5.3 Input/Output Augmented Language Models 30
2.6 Text-Interactive Environments & Corresponding Methods 32
2.6.1 Recent Online RL Models 36
2.7 Reinforcement Learning via Supervised Learning 38
2.7.1 Decision Transformers 40
2.8 Relevant Work after publications 42
2.8.1 On Prompting-Based Methods 42
2.8.2 On Proof Generation 44

CHAPTER 3 ORGANIZATION OF THE THESIS 46

CHAPTER 4 ARTICLE 1: MEASURING SYSTEMATIC GENERALIZATION IN NEURAL PROOF GENERATION WITH TRANSFORMERS 48
4.1 Introduction 48
4.2 Related Work 50
4.3 Evaluating systematic generalization through interpretable reasoning 51
4.3.1 The task 51
4.3.2 Proof resolution strategies 53
4.3.3 Systematic generalization in proof generation 54
4.4 Experiments and Analysis 56
4.4.1 Answer Accuracy 57
4.4.2 Proof Validity 58
4.4.3 Proof is given 60
4.5 Conclusion 61
4.A Original CLUTRR evaluation 63
4.B Experiments parameter settings 65
4.C More parameters 66
4.D Fine-tuning GPT2 67
4.E Encoder-Decoder Network 68

CHAPTER 5 ARTICLE 2: DOES ENTITY ABSTRACTION HELP GENERATIVE TRANSFORMERS REASON? 69
5.1 Introduction 69
5.2 Related Work 71
5.3 Introducing Abstraction Inductive Biases 73
5.3.1 Abstraction as an additional embedding 74

5.3.2 Abstraction as an additional sequence to encode 75
5.3.3 Abstraction as an auxiliary task 75
5.4 Experiments 76
5.4.1 Compositional generalization with CLUTRR 77
5.4.2 Abductive Reasoning with ProofWriter 79
5.4.3 Multi-hop Question Answering with HotpotQA 81
5.4.4 Conversational Question Answering with CoQA 82
5.5 Discussion 83
5.6 Conclusion 86
5.A Hyperparameters 88
5.B On the abstraction accuracy of the `dec-loss` model 89
5.C CLUTRR results when `n=1` 89

CHAPTER 6 LONG-CONTEXT LANGUAGE DECISION TRANSFORMERS AND EXPONENTIAL TILT FOR INTERACTIVE TEXT ENVIRONMENTS 91
6.1 Introduction 91
6.2 Methodology 94
6.2.1 Problem setup 94
6.2.2 Goal conditioning 95
6.2.3 Next State Prediction 97
6.3 Related Work 98
6.4 Experimental setup 99
6.4.1 Jericho Engine 99
6.4.2 Data Collection 100
6.4.3 Sequence Definition 100
6.5 Experimental Results 101
6.5.1 The Effect of Exponential Tilt 102
6.5.2 Our Goal Conditioning Strategies 103
6.5.3 Predicting the Next Observation 105
6.6 Conclusion 106
6.A Trajectories Statistics 107
6.B Hyperparameters 108

CHAPTER 7 GENERAL DISCUSSION 109
7.1 Summary of Works 109
7.2 Limitations 110

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS 112

REFERENCES 114

APPENDICES 131

LIST OF TABLES

Table 4.1 CLUTRR example of level 3 (ie: 4 entities, 3 relations, 2 proof steps). The proof follows the `short-proof-rev` strategy. We refer the reader to Figure 4.1 to visualize the corresponding graph in which solid lines refer to the facts given in the story and dotted lines refer to the new facts inferred in each proof step. 52

Table 4.2 Proof resolution types for an example of level 3. We refer the reader to Figure 4.1 for the kinship graph corresponding to this example. **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed. 53

Table 4.3 Percentage of the test proof's building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value of 100% to better visualize which building blocks were entirely contained in the training set. 55

Table 4.4 Percentage of the **original test** proof building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value close to 100% to better visualize which building blocks were entirely contained in the training set. 63

Table 4.5 Percentage of the **Named test** proof's building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value of 100% to better visualize which building blocks were entirely contained in the training set 63

Table 4.6 Parameter settings. 65

Table 5.1 Different architectures for different abstraction strategies. X (blue) is the original sequence embedding, X_s (green) is the embedding of the simplified sequence with entities replaced by their entity type tags, "*ENC*" is the T5 encoder, H (blue) is the contextualized representation of sequence X, H_s (green) is the contextualized representation of sequence X_s, "*DEC*" is the T5 decoder, and Y is the target sequence to predict. 74

Table 5.2 Prediction accuracy on CLUTRR test set for all difficulty levels. Models have been trained on levels 2, 4, 6 with only 9.58% of all the $(e1, rel, e2)$ triples present in the test set. Per-level performance is colored in shades of green for better visualisation. The red boxed area indicates test problems at depths unseen during training. 78

Table 5.3 Prediction accuracy on different slices of the ProofWriter D5 test set for all our models and the originally reported numbers by Clark et al. [1]. Models have been trained on depth D0, D1, D2. Models are trained in the "open-world" assumption (OWA), except for the original Clark et al. [1] model which was trained in the "closed-world" assumption (CWA). Per-depth performance is colored in shades of green for better visualisation. The red boxed area indicates test problems at depths unseen during training. 80

Table 5.4 Average test performance for all models on HotpotQA. Average and standard deviation computed with 3 random seeds. 82

Table 5.5 Average test performance for all models on CoQA. Average and standard deviation computed with 3 random seeds. 83

Table 5.6 Percentage of tokens being tagged as entities and estimated F1 score of the tagger on each dataset. 84

Table 5.7 Library version and model hyper-parameters. 88

Table 5.8 Average CLUTRR answer accuracy for models trained with $n = 20$ and $n = 1$ token per entity tag. 89

Table 6.1 Comparison of different policy training and action selection techniques (adapted from Piche et al. [2]). We compare our approach with Decision Transformers (DTs) [3], Reward Weighted Regression (RWR) [4,5], Reward-Conditioned Policies (RCP) [6] (also used by Multi-Game Decision Transformers [7]), Reweighted Behavior Cloning (RBC) [8] (also used by Trajectory Transformer (TT) [9]), and Implicit RL via supervised learning (IRvS) [2]. Where $\mathbf{s}$ represents the state as encoded by the model and depends on the architecture and inputs used. 97

Table 6.2 Average and the maximum score for each game across 5 random seeds for each goal condition (GC) variation. On the bottom line, scores are normalized according to the maximum human score (enchanter: 400; sorcerer: 400; spellbrkr: 280; spirit: 250; ztuu: 100) and averaged across games. 104

Table 6.3 Average and the maximum score for each game across 5 random seeds and 4 goal conditioning methods, with ($\lambda = 0.5$) and without ($\lambda = 0.0$) the auxiliary loss on the prediction of the next observation o_{t+1}. Scores are then normalized according to the maximum human score ('*max=#*') and averaged across games on the bottom line. 105

Table C.1 CLUTRR test set level 3 output examples. Correct answers are in green and wrong answers are in red for better visibility. 136

LIST OF FIGURES

Figure 2.1 Computational graph of a 1-layer RNN; the unrolled RNN is shown on the right. Matrices **U**, **W**, and **V** are the parameters of this network. 9

Figure 2.2 Computational graph of a 1-layer RNN encoder-decoder. 10

Figure 2.3 Computational graph of the attention mechanism for one decoding step. 11

Figure 2.4 Computational graph of a L-layer Transformer network [10]. 12

Figure 2.5 Reinforcement Learning loop between an agent and an environment . 14

Figure 4.1 Example of a CLUTRR graph with known facts (solid lines) and unknown facts to infer (dotted lines). 49

Figure 4.2 Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline. 57

Figure 4.3 Proof validity for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. 59

Figure 4.4 Answer accuracy for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>" and they generate the answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline. 60

Figure 4.5 Answer accuracy on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and asked to generate the proof and answer. Models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **lp** = long-proof, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline. 64

Figure 4.6 Evaluation of models trained on levels 2, 4, 6 only. 65

Figure 4.7 Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. Models are trained on levels 2, 4, 6 only. Stories are expressed with the `facts` template. Different proof settings are evaluated: **np** = no-proof and **spr** = short-proof-reversed. We also report the naive most-frequent-relation (**mfr**) baseline. Results on other proof settings with the 2.5M parameter network can be found in Figure 4.2a. 66

Figure 4.8 Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. Models are fine-tuned on levels 2, 4, 6 only. Stories are expressed with the `amt` template. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **np** = no-proof. We compare the performance of models trained from scratch (dotted lines; `gtp2FS-`) and of fine-tuned models (solid lines; `gpt2FT-`). 67

Figure 4.9 Answer accuracy for all test levels from 2 to 10. The models encodes the input "<STORY> [story] <QUERY> [query]" and they decode the proof and answer. Models are trained on levels 2, 4, 6 only. Stories are expressed with the `facts` template. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline. 68

Figure 5.1 Average answer accuracy on extrapolation reasoning levels for models trained with 0%, 25%, 50% and 75% entity tagger noise on CLUTRR and ProofWriter. The baseline model (no abstraction) is represented by a black dotted line. Previous work on ProofWriter is represented by a black solid line. 85

Figure 6.1 Overview of our approach: Noisy trajectories are generated from a high quality game walkthrough by taking 100 random steps at each 5% of the trajectory. The collection of trajectories on multiple games is used to train our LLDT model offline to predict a goal condition, next action, and next observation. The LLDT is then evaluated in each game environment, initialized with 5 random seeds. 92

Figure 6.2 Our Long-Context Language Decision Transformer framework. A trajectory of length T is split at a random index $t \in [0, T-1]$. The model encodes the sequence of observations (o), goal conditions (g), and actions (a) up to time step t. The first o_1 and last o_t observations are fully written, but to shorten the input sequence, the other intermediate observations are replaced by a special "< STATE >" token. The decoder predicts the goal condition g_t, action to take a_t, and next observation o_{t+1}. 93

Figure 6.3 Average normalized score across different Jericho games (enchanter, sorcerer, spellbrkr, spirit, ztuu) with various amounts of exponential tilt ("*Predicted GC*" lines). We also report the performance of a model being conditioned on the optimal goal according to each game's maximum score ("*Optimal GC*" line). The average normalized score of various baselines trained on each game is depicted with dotted lines. . 103

Figure 6.4 Proportion of trajectory normalized scores for a selection of games. In each sub-figure title, n is the number of trajectories and ms is the maximum score. The X-axis is the normalized score the trajectory achieves. The Y-axis is the proportion of trajectories finishing with that score. 107

Figure 6.5 Proportion of trajectory lengths for a selection of games. In each sub-figure title, n is the number of trajectories. The X-axis is the number of steps in a trajectory. The Y-axis is the proportion of trajectories of that length. 108

LIST OF SYMBOLS AND ACRONYMS

AI	Artificial Intelligence
AMT	Amazon Mechanical Turk
AvgRTG	Average Return-to-go
CFQ	Compositional Freebase Questions
CNN	Convolutional Neural Network
CoT	Chain of Thought
CTP	Conditional Theorem Prover
DBCA	Distribution-Based Compositionality Assessment
DT	Decision Transformer
EMAT	Efficient Memory-Augmented Transformer
FinS	Final Score
GC	Goal Condition
GAT	Graph Attention Network
GECA	Good-Enough Compositional data Augmentation
GNN	Graph Neural Network
GNTP	Greedy Neural Theorem Prover
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HCP	Hypernym Class Prediction
HGN	Hierarchical Graph Network
IF	Interactive Fiction
ILP	Inductive Logic Programming
ImR	Immediate Reward
IRvS	Implicit Reinforcement Learning via Supervised Learning
KAR	Knowledge Attention and Recontextualization
KB	Knowledge Base
KG	Knowledge Graph
LLDT	Long-Context Language Decision Transformer
LLM	Large Language Model
LM	Language Model
LSTM	Long Short-Term Memory
MB-RCP	Model-based Reward-Conditioned Policy
MDP	Markov Decision Process

MLM	Masked Language Modeling
MLP	Multi-Layer Perceptron
MRC	Machine Reading Comprehension
MT	Machine Translation
NER	Named Entity Recognition
NLI	Natural Language Inference
NLP	Natural Language Processing
NLU	Natural Language Understanding
NMN	Neural Module Network
NTP	Neural Theorem Prover
POMDP	Partially Observable Markov Decision Process
QA	Question Answering
RBC	Reweighted Behavior Cloning
RCP	Reward-Conditioned Policy
RL	Reinforcement Learning
RRN	Recurrent Neural Network
RTG	Return-to-go
RvS	Reinforcement Learning via Supervised Learning
RWR	Reward Weighted Regression
SGD	Stochastic Gradient Descent
TLM	Transformer Language Model
TT	Trajectory Transformer
UDRL	Upside-down Reinforcement Learning

LIST OF APPENDICES

Appendix A Article 1 / Long Proof pseudo-code 131
Appendix B Article 2 / Input - Output examples 133
Appendix C Article 2 / Prediction examples 136
Appendix D Article 2 / GeoQuery Results 144

CHAPTER 1 INTRODUCTION

Since the advent of modern computers, the frequency of human-to-machine interactions has been rising continuously. This can largely be attributed to machines being capable of performing tasks that were once deemed impossible. With ongoing and accelerating technological advancements, humans have become used to interacting with machines for a range of purposes. However, these interactions are typically done with special commands through multiple clicks and mouse drags, which are often inefficient, tedious, and not flexible enough to express one's true intention. On the other hand, humans communicate with one another efficiently using natural language. The field of natural language processing (NLP) aims to bridge this gap by teaching machines how to understand and generate natural language. Natural language interfaces can potentially increase the efficiency and productivity of workers trying to achieve a task by offering a chat-based experience on any computer application. As such, advances in the field of NLP research bring us closer to the goal of communicating with machines in natural language.

Having a conversation with computers is not a novel idea. Already in the 1950s Allan Turing was thinking about it and introduced the "Turing test" [11]. Shortly thereafter, Weizenbaum developed the first computer program that could interact with humans in natural language (ELIZA [12]) using templates and decision tree rules. A couple of years later, computation with layered networks of artificial neurons was introduced. [13, 14]. However such networks required (and still do) a lot of data to be trained and thus were not used for NLP systems until recently. Recent progress in computer hardware and the availability of significant amounts of data changed our approach to building NLP systems. We now have powerful enough computers and sufficient amounts of public data to train artificial neural networks of millions to billions of parameters and build data-driven NLP systems.

1.1 The Field of Research

Natural Language Processing (NLP) is a field of artificial intelligence (AI) that aims to enable computers to understand and generate natural language, most often in written form rather than spoken (which is the domain of Speech Processing). Some popular tasks to test this ability are: Question Answering (QA), which involves answering a question using a given document, Natural Language Inference (NLI), which involves identifying whether one sentence entails or contradicts another, Machine Translation (MT), which involves generating the equivalent of a sentence in one language from a sentence in another language, Language

Modeling, which involves predicting the subsequent words of a sentence given its beginning, and many other related tasks.

The modern approach to tackle these tasks is to use vectors of hundreds of dimensions to represent words (i.e., word vectors/embeddings) and sentences (i.e., sentence embeddings) [15–17] and then process them through neural networks. Traditionally, sequences of word vectors were processed with neural network architectures such as Recurrent Neural Networks (RNNs) [18], Long Short-Term Memory networks (LSTM) [19] and Gated Recurrent Units (GRU) [20]. However, more recently, a new type of architecture relying heavily on the attention mechanism [21], called Transformers, has been demonstrated to outperform traditional recurrent architectures on multiple tasks [10, 17, 22].

Neural networks are commonly trained to minimize a differentiable loss function between their predicted output and a ground truth target using optimization algorithms such as stochastic gradient descent (SGD) [23]. Recently, a new training paradigm emerged from the literature: by training Transformer networks to perform the language modeling task on large amounts of data (such as the entire web, or collections of books), one can achieve state-of-the-art performance on a variety of NLP tasks by fine-tuning the network on a relatively small amount of data for a relatively small amount of time [24–26]. This suggests that language modeling, when performed on large-scale Transformers with massive datasets, is a task that captures useful features for all other NLP tasks. Transformer networks with a large number of parameters (in the range of billions) pre-trained in this fashion on enormous corpora of text are referred to as Large Language Models (LLMs) or Foundation Models in the literature.

1.2 Motivation

Although significant progress has been made in the field of NLP, it is still uncertain to which extent Transformer Language Models (TLMs) are capable of logical reasoning. For text-generative systems such as TLMs to be widely used as a tool to help humans solve complex tasks, they must be able to decompose problems into smaller ones, propose a solution for each of them individually, and compose these solutions back to accomplish the original task. In addition, generative systems should output truthful sentences that can be verified by logical reasoning steps instead of convincing the end user that they generated correct knowledge by "sounding" confident about what they generate. If not, such systems can have the potential of propagating false information at massive scales. For these reasons, generative TLMs must be able to logically reason from a set of given facts and infer new ones by thinking step by step with truthful inference mechanisms.

In the past, recurrent language models were known to be weak logical reasoners. In particular, these models were found to be better at memorizing data from their training sets rather than learning general compositional rules [27]. Multiple studies suggest that language models can be easily deceived when it comes to tasks requiring compositional generalization [28–31].

With the impressive progress made by Transformers on various NLP tasks, the motivation of this research is thus to *understand if TLMs can be used to perform logical reasoning on natural language tasks*. A challenging task in the NLP community is the "link prediction" problem, which involves inferring new facts that are coherent given a set of true facts. The name link prediction comes from the graph neural network (GNN) community [32] where the task is to discover new relationships (links) between entities (nodes) in a knowledge graph. The core limitation of GNNs is that the data must be represented as a graph structure. However, most human knowledge is not available in graphs but in texts which is difficult to reason over automatically due to the ambiguity of natural language. As a result, the idea of inferring new knowledge by 'reading' existing facts in text form is an attractive and powerful one.

Furthermore, when inferring new knowledge from existing facts, an important concept arises: systematic generalization. When a model is expected to perform on unseen combinations of knowledge, the task is said to be evaluating the generalization capacity of the model. Systematic generalization has been characterized as the capacity to understand and produce a potentially infinite number of novel combinations from known components [33, 34]. More recently, systematic generalization has been seen as "*the ability to manipulate concepts in new combinations after being trained on all possible concepts, but only a subset of all their possible combinations*" [35]. For instance, if a model is trained to identify blue squares, blue triangles and green triangles, it should also be able to identify green squares. In this toy example, two concepts (shape and colour) are mixed together. More broadly, if a system has been trained on a wide range of factual information, it should be able to integrate and combine them in novel ways to deduce new facts. This perception of systematic generalization encapsulates the reasoning aspect within the learning paradigm.

Research Question Overall, this thesis asks the following question: "*are Transformer language models able to reason logically across multiple steps and what is required for them to be better at it?*"

1.3 Research Objectives

The above question is further divided into three research objectives.

1. The primary objective of this thesis is to investigate the factors that influence the ability of vanilla Transformer models to capture logical rules. Specifically, one objective is to evaluate if training a model to generate intermediate logical steps in a question-answering setting helps to generate the final answer. This will help identify whether the model is using the intermediate reasoning steps like humans would, or if it is actually better to simply decode the answer without generating intermediate steps. The model will be tested against its ability to generalize to increasingly more complex questions requiring longer reasoning chains but, that rely on the same logical rules present in the training set. This idea of systematic generalization is a core evaluation metric that allows seeing if the model has indeed captured the latent rules of the environment and is able to recursively apply them, or if it is instead trying to memorize solutions and rely on non-logical factors such as syntax and memorization. The first work in this thesis presented in Chapter 4 addresses this first objective by measuring systematic generalization through the ability of a model to reason about unseen *combinations* of inference rules despite being trained on all individual inference rules.

2. The second objective of this research is to enhance the reasoning capacity of TLMs by introducing some form of inductive bias into the model and exploring new mechanisms that can be added to the training procedure of the network. Focusing on the inductive bias of *abstraction*, the objective is for the model to reason about abstract entity types rather than grounded tokens. This objective is motivated by the fact that humans abstract to simplify reasoning and become very efficient. This is true for instance in mathematics when manipulating variables instead of raw numbers. The introduction of generic variables allows progress in a logical reasoning process without keeping track of every grounded atomic entity. Manipulating abstract concepts allows humans to generalize knowledge across domains. The hypothesis is that introducing some abstraction to the network will help the model generalize to more complex tasks. The second work of this thesis presented in Chapter 5 undertakes this second objective by asking the question "does entity abstraction help generative transformers reason?"

3. Eventually, the final objective of this thesis is to gain more control over the behavior of TLMs by conditioning their generation on some desired objective. As people spend a significant fraction of their lives performing activities closely linked with natural languages, such as having conversations, writing e-mails, filling out forms, reading and writing documents, having an intelligent assistant able to perform complex tasks in a natural way would increase one's productivity. Interactive text-based games allow for experimental research as a good proxy for such settings without having to allocate any

human resources. Each environment (or game) has a main objective with some side quests (or sub-objectives), all expressed in text. The goal of the agent is to accomplish as much as possible in this virtual world by communicating actions to take at each step. The interactive nature of these environments also makes them an ideal framework to evaluate the multi-step reasoning abilities of TLMs. The goal of this research objective is to leverage pre-trained TLMs to solve long text-based interactive tasks and propose improvements over previous methods. Chapter 6 of this thesis accomplishes such an objective by introducing "long-context language decision transformers and exponential tilt for interactive text environments".

1.4 Methodology

To answer the research question, let's first define what is meant by "logically reasoning across multiple steps". There exist multiple forms of reasoning (deductive, inductive, abductive, mathematical, and more) but this thesis does not focus on any specific form. Here, reasoning is defined as being able to decompose a potentially complex task into simpler subtasks, solve each subtask, and compose their answers to eventually solve the original task. One desideratum towards achieving this goal is to have language models capable of producing outputs that are logically coherent with their contextual input, regardless of the reasoning form at hand. For instance, if the model is given the facts that "all humans are mortal and Alice is a human", it should *know* that "Alice is mortal" (deductive reasoning). Similarly, if the model is given the facts that "Alice picks 3 apples per day and gives one to Bob every day", it should *know* that "after 3 days, Alice has 6 apples" (mathematical reasoning).

The main focus of the thesis resides in the aspect of *multi-step* reasoning. As mentioned above, this refers to situations when a model needs to decompose a complex task, solve intermediate tasks, and compose intermediate solutions. This research investigates this aspect of reasoning by setting language models in situations in which they need to either (i) combine multiple sources of information in their context to generate an answer, or (ii) generate multiple answers each logically following each other, or (iii) both. The multi-step reasoning aspect will be evaluated throughout all contributions of the thesis by using tasks requiring the model to 'think' step by step. In particular, this methodology will be applied to the three objectives listed above like so:

1. First, the capacity of vanilla transformer models to capture logical rules will be evaluated with a question-answering task (CLUTRR [36]) in which the answer to the question is not explicitly stated in the input. To correctly answer, the model will be required to

do multi-hop inference: that is (1) extracting relevant information from its input, (2) combining the information in order to infer new knowledge, (3) repeating the procedure until it can correctly answer the question. This type of question-answering problem is appealing not only for its difficulty but also for its explicit requirement to perform reasoning steps over multiple sentences. The model will be trained on a collection of examples varying from different difficulty levels. The model will then be triggered to not only answer the question but also explain itself by generating the intermediate reasoning steps required to go from the question to the final answer. The work presented in Chapter 4 will evaluate generative TLMs on both the correctness aspect (did the model generate the correct answer) and the logical reasoning aspect (was the model able to identify the relevant intermediate steps). In addition, the capacity of the model to generalize to unseen difficulty levels of questions will be evaluated.

2. Second, the abstraction inductive bias will be added to vanilla TLMs by leveraging existing NLP libraries such as spacy[1] to annotate training data. In particular, a named entity recognition (NER) model will be used to extract entity types from the dataset. These automatically retrieved labels will then be used as additional inputs to the model and as an additional auxiliary prediction objective. Different abstraction strategies will be compared against each other. Model variants will be tested based on their ability to solve different reasoning tasks. Multiple datasets will be used to better understand when such abstraction is beneficial. In particular, both controlled setups such as CLUTRR [36] and ProofWritter [37], and more natural datasets such as HotpotQA [38] and CoQA [39] will be explored. The first two tasks are algorithmically designed to evaluate multi-step reasoning while the latter two are designed to be more 'realistic' with human written text which is more natural but harder to control for multi-step reasoning. The results of this work are detailed in Chapter 5.

3. Eventually, in order to explore the reasoning capacity of TLMs in interactive text environments, the third contribution of this thesis will leverage text-based games such as Jericho [40]. These artificial environments are chosen because of their analogy to intelligent text assistants helping people with various tasks. At the same time, interactive environments also exhibit multi-step reasoning requirements yet are under-explored in the NLP community. Decision transformers (DTs) with exponential tilt will be used to frame the reinforcement learning (RL) task as an offline supervised learning procedure by sequencing the series of observations, actions, and rewards experienced while interacting with an environment [3]. Different conditioning methods of DTs will be

[1] https://spacy.io

introduced and compared. In addition, the proposed model will be trained with an additional auxiliary loss to perform better in text-based games. The aforementioned improvements will result in a novel model: "long-context language decision transformers" (LLDTs) and be further presented in Chapter 6.

1.5 Thesis Outline

The chapters of this thesis are organized as follows. Chapter 2 provides some technical background and a critical literature review of the field of NLP specifically centred around multi-step reasoning. This chapter also introduces Reinforcement Learning (RL) in the context of interactive text environments. Chapter 3 explains the organization of this thesis and the relation of the following three chapters (4, 5 and 6) to the research objectives described above. Chapter 4 measures the systematic generalization of language models in a proof generation framework. This chapter is the result of a paper published in Advances in Neural Information Processing Systems 2020. Chapter 5 introduces entity abstraction techniques to improve the generalization capacity of language models. This chapter is the result of a paper published in the Transactions on Machine Learning Research journal in 2022. Chapter 6 proposes novel methods to improve the performance of language models in complex text-interactive environments. Chapter 7 present a summary of the works and examines some of its limitations. Finally, Chapter 8 concludes the thesis by proposing interesting future directions to solve some of the aforementioned limitations.

CHAPTER 2 LITERATURE REVIEW

This chapter provides a critical literature review of the field of NLP specifically centred around multi-step reasoning. Reinforcement Learning (RL) is also introduced in the context of interactive text environments. First of all, some technical background on various neural network architectures most commonly used to process text is introduced in Section 2.1. Additionally, Section 2.2 briefly summarizes some influential work on Systematic Generalization. The chapter then covers some methods for reasoning on text such as neural theorem provers (Section 2.3), graph-based methods (Section 2.4), and knowledge-augmented language models (Section 2.5), with a focus on knowledge-graph, memory, and input/output-based methods. The chapter then continues by covering previous work on text-based environments and online reinforcement learning methods to solve them in Section 2.6 before focusing on offline RL methods and return conditioned supervised learning in Section 2.7. Eventually, this chapter closes in Section 2.8 with some of the most recent progress in the field of NLP (in the context of reasoning) that happened after the first contribution presented in this thesis, as the field has seen significant progress in 2020-2023.

2.1 Technical Background

This section provides some technical background on neural network architectures that are widely used for reading, representing and generating text. Moreover, a brief overview of the Reinforcement Learning (RL) field will be given to better contextualize the last contribution of the thesis.

Recurrent Neural Networks In order to encode variable-length sequences (i.e. sentences made of tokens) into fixed representations (i.e. vectors), the traditional neural architectures were Recurrent Neural Networks (RNNs) [18, 41–43] until Transformers [10] were invented (more about this architecture below). Each token is represented by a vector (or embedding) and given as input to RNNs. These networks are flexible enough to encode data that is correlated in time by introducing a feedback loop on each artificial neuron. This can be represented as a directed cyclic graph as illustrated in Figure 2.1. In theory, by unfolding this graph, the network can encode sequences from any length.

Unfortunately, in practice, it has been observed that it is difficult to train RNNs to capture long-term dependencies because the learning gradients tend to either vanish (most of the time) or explode (rarely, but with severe effects) [44, 45]. Long short-term memory networks

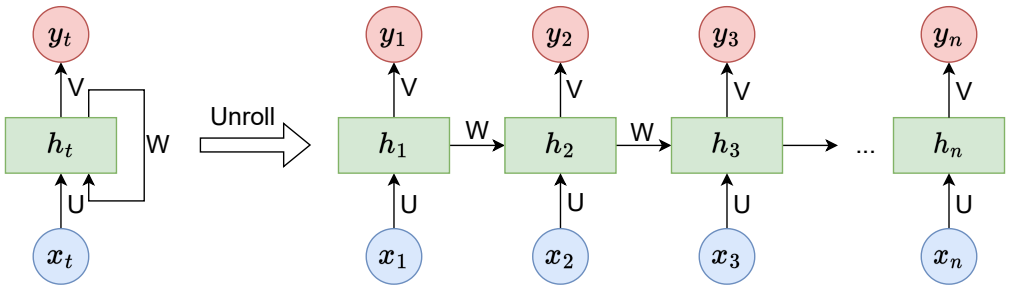


Figure 2.1 Computational graph of a 1-layer RNN; the unrolled RNN is shown on the right. Matrices $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{V}$ are the parameters of this network.

(LSTMs) are a special case of RNNs, introduced primarily to overcome the vanishing gradient problem [19]. The key idea behind LSTMs is that they add an additional state (memory state) that gets only slightly modified throughout the input sequence. It is thus easy for information to flow as the input gets longer. The amount of information that can be added to (or removed from) the memory state is carefully regulated by multiple gating mechanisms. A few years later, Gated Recurrent Unit networks (GRUs) [20] were introduced to solve the same issue as LSTMs but with fewer parameters, resulting in simpler implementation and easier training.

Formally, given an input sequence $\mathbf{X}$ made of n token embeddings $\{\mathbf{x}_t\}_{t=1}^{n}$, RRNs iteratively construct a sequence of hidden states $\mathbf{h}_t$ and produce a sequence of outputs $\mathbf{y}_t$. RNNs are defined by the following update equations at each time step t:

$$\mathbf{h}_t = tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}), \tag{2.1}$$

$$\mathbf{y}_t = softmax(\mathbf{V}\mathbf{h}_t + \mathbf{c}), \tag{2.2}$$

with $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}$ and $\mathbf{c}$ being parameters of the network that do not depend on the time step t. To learn the optimal value of each parameter, the network is trained with an optimization algorithm such as stochastic gradient descent (SGD) [23]. Overall, RNNs, LSTMs and GRUs all share a common structure of repeating neural network modules when unrolled as seen in Figure 2.1. The main difference between RNNs, LSTMs and GRUs resides in the implementation of the repeating module itself (Equation 2.1).

Encoder-Decoder Conditional text generation tasks such as question answering require the input and output sequences to be of different lengths. In that case, sequential models such as RNNs (or Transformers as discussed below) can be combined in an encoder-decoder fashion [46, 47] where one *encoder* model transforms input text $\mathbf{X}_{1..n}$ into a dense vector

$\mathbf{h}_n$ and another *decoder* model transforms $\mathbf{h}_n$ into an output sequence $\mathbf{Y}_{1..m}$. The unrolled computational graph of such architecture is depicted in Figure 2.2. Note that the input

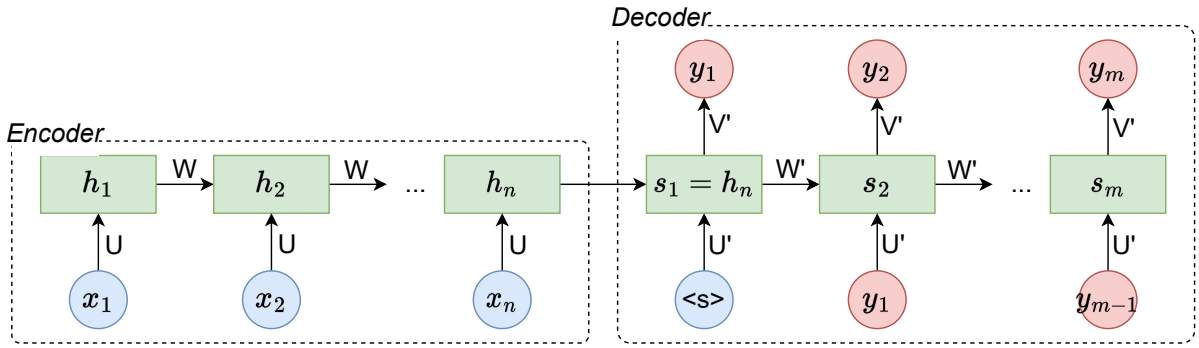


Figure 2.2 Computational graph of a 1-layer RNN encoder-decoder.

and output of the decoder network is the same sequence, but with the input being shifted to the right by one token. This corresponds to the language modeling objective mentioned in Section 1.1, which involves predicting the next token given the previous ones [48]. This objective is commonly used on decoder architectures as we will see multiple times during the thesis. In addition, it is important to note that when decoding, the generation of the target sequence may not only depend on the last hidden representation of the encoder $\mathbf{h}_n$ even though it is supposed to capture all the input sequence information. This motivated the development of the attention mechanism [21].

Attention Mechanism The idea of the attention mechanism [21] is to take a weighted sum of all the encoder hidden states with attention weights depending on the current generation step: $\sum_{j=1}^{n} \alpha_{i,j}\mathbf{h}_j$. The attention weights $\alpha_{i,j}$ are dynamically learned along with the rest of the network parameters via an optimization algorithm such as SGD. The computational graph of this procedure is depicted in Figure 2.3. Since introduced, having an attention mechanism in RNNs has been the norm. However, some problems still remain with RNNs: although LSTMs and GRUs allow for information to 'flow' better from distant tokens than original RNNs, the iterative nature of the architecture still greatly limits their long-term representation ability in practice [25, 49]. In addition, their cyclic behavior also limits their parallelization on graphical processing units (GPUs) [10].

Transformer To mitigate the above limitations, Vaswani et al. [10] proposed a new architecture called *Transformer* without any cycles and only relying on attention, thus greatly improving the parallelization power of their RNN predecessors. The Transformer network

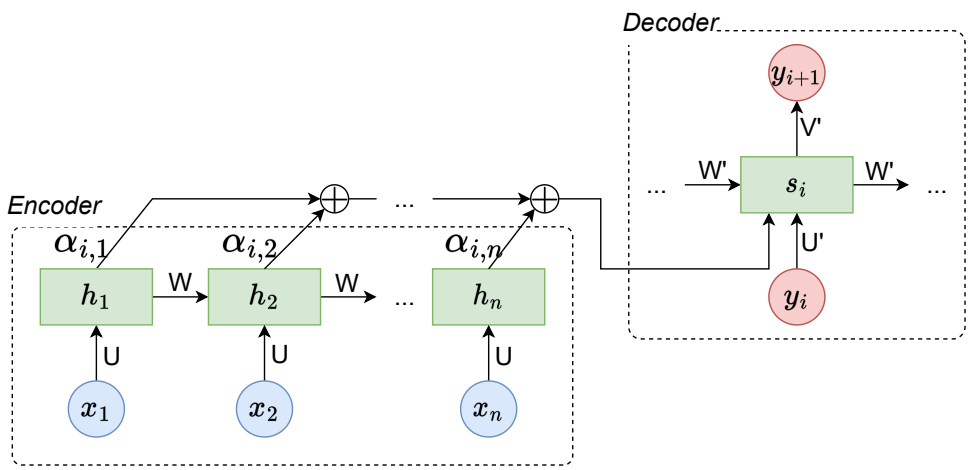


Figure 2.3 Computational graph of the attention mechanism for one decoding step.

is an encoder-decoder architecture composed of multiple encoder and decoder blocks. Each block's output is fed to the next as input.

Attention mechanisms within each block vary slightly between the ones used in encoder blocks and the ones used in decoder blocks, but they all rely on the same principle. Attention can be defined as a function taking as input a set of n query vectors ($Q \in R^{n \times d}$), m key vectors ($K \in R^{m \times d}$) and m value vectors ($V \in R^{m \times d}$), each of dimension d. For each query vector, the mechanism computes an attention distribution α over the key vectors: $\alpha = \text{softmax}(\frac{Q.K^T}{\sqrt{d}})$. The function then returns a weighted combination of the value vectors: $attn(Q, K, V) = \alpha V$.

In encoder-decoder Transformers, the "regular" attention mechanism has the query vectors (Q) representing the current decoder time-step, while the key (K) and value (V) vectors represent all the information previously encoded by the encoder module. When the query (Q), key (K), and value (V) vectors are all the same, the attention mechanism is called "self-attention". This type of attention is used in encoder Transformers to capture relationships between tokens belonging to the same sequence. In generative models such as decoder Transformers, the model should not consider future information when computing the present token representation. In particular, tokens that will be predicted at later time steps should not influence the current token. As such, the self-attention in decoder Transformers is often modified to mask future tokens. This type of attention mechanism is called "causal attention".

Transformer encoder blocks are defined by two consecutive operations: (1) a self-attention mechanism on the output of the previous block, and (2) a feed-forward network. Each operation is followed by a layer normalization step to facilitate training. Decoder blocks are

defined by three consecutive operations: (1) a causal attention mechanism on the output of the previous block, (2) a regular attention between the last encoder block and the current decoder block, and (3) a feed-forward network. Each operation is again followed by a layer normalization step to facilitate training. The entire Transformer computational graph is pictured in Figure 2.4.

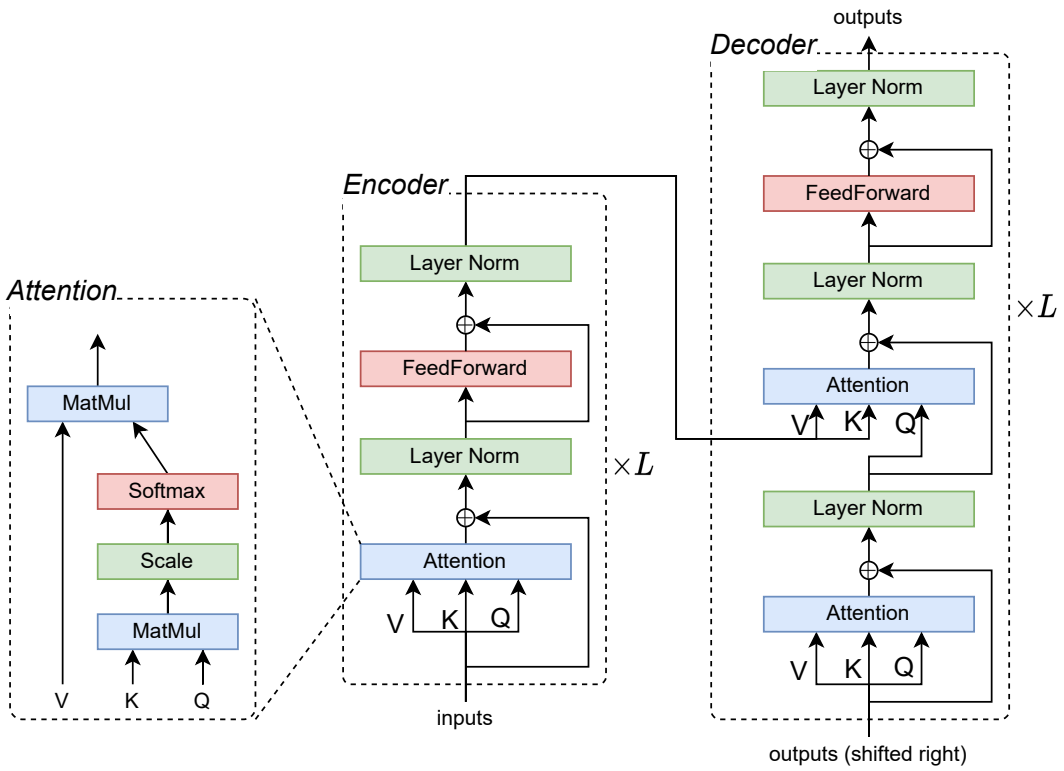


Figure 2.4 Computational graph of a L-layer Transformer network [10].

Note that this entire procedure does not encode any information about the original order of tokens in the input and output sequences. This makes the current model behaves like a bag-of-word model [10]. In order to solve this issue, the authors of the original paper summed the token embeddings with sinusoidal positional embeddings representing the relative position of each token in the source and target sequences respectively.

The significant advantage of Transformers comes from the fact that their attention mechanism is done over every token simultaneously in a sequence. This allows the information between two tokens to be directly encoded even if they are relatively far from each other. As a result, Transformers initiated a wave of innovation in the NLP research community. With their advantage of greater parallelization, these networks can be extended to billions of parameters and trained on terabytes of text to learn useful representations for many down-

stream tasks [17, 22, 24–26, 50]. Pre-training on large datasets and fine-tunning on specific tasks became a new training paradigm that achieves state-of-art performances on many NLP tasks such as Natural Language Inference (NLI) [51], Question Answering (QA) [51, 52], Machine Translation (MT) [52], and many others [50].

2.1.1 Large Language Models (LLMs)

Since the introduction of the Transformer network, multiple variations have been developed, in particular with increasing model size. Transformer networks with a large number of parameters (in the range of hundred of millions to billions for 2022 standards) trained to predict the next word (language modeling objective) on massive corpora of text are referred to as Large Language Models (LLMs) or Foundation Models [53].

Some of these models such as BERT (bidirectional encoder representations from Transformers) [17] and RoBERTa (a robustly optimized BERT pretraining approach) [51] only use Transformer encoder blocks and are trained with a de-noising objective such as Masked Language Modeling (MLM), which consists of predicting masked words in the input sequence. BERT and RoBERTa are trained on 16GB and 160GB of English text and their largest model size have 340 and 355 million parameters respectively. These types of models are often used as a starting point for building text encoders, classifiers or retrievers [17, 51, 54–56].

Other models such as T5 (text-to-text transfer Transformer) [52] and BART (a denoising autoencoder for pretraining sequence-to-sequence models) [57] are encoder-decoder models similar to the original Transformer architecture. T5 is trained with the language modeling objective on 750GB of English text and its model size varies from 60 million to 11 billion parameters. These models are used for conditional generation tasks such as summarization, machine translation or question answering [52, 57, 58].

Eventually, other models such as GPT-1,2,3 (generative pre-trained Transformers) [22, 24, 50] and PaLM (pathways language model) [59] are fully auto-regressive, decoder-only models. After removing the inter-attention module between the encoder and decoder of the traditional Transformer architecture, these models use decoder blocks with causal attention only. As such these types of models are also called "causal Transformers". GPT-3 and PaLM are trained on 45TB and 70TB of text (ranging from multi-lingual sources to source code) with the language modeling objective and their largest model size have 175 and 540 billion parameters respectively. These models are often used for tasks such as summarization, machine translation or question answering. Additionally, at this scale of parameters, these models have been shown to perform a multitude of tasks by simply giving a few examples of input/output pairs in their context [50, 59]. This is referred to as "in-context learning" and

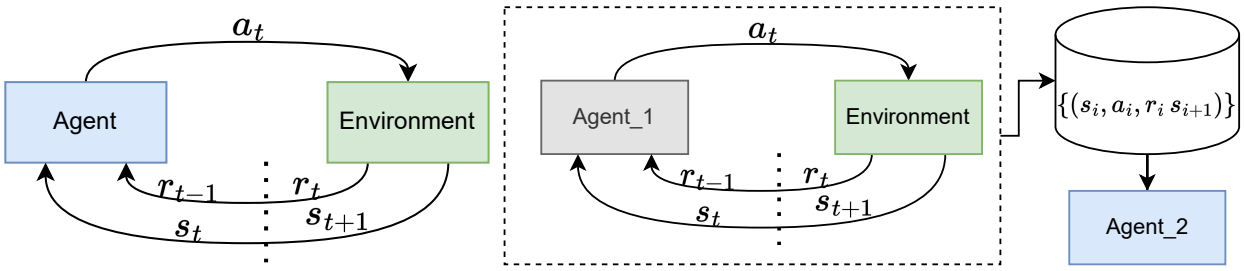
"prompting". These methods yielded powerful applications in the context of multi-step reasoning that will be mentioned in Section 2.8 as they were discovered after the publication of the first work presented in this thesis.

Overall, pre-trained LLMs like BERT, T5, or GPT are often used as building blocks or as initialization of model weights. This is because they encode world knowledge and natural language processing inductive biases in their parameters which can be transferred to any task at hand [50, 60–62]. The work presented in this thesis will focus on encoder-decoder and decoder-only Transformers since they are generative models capable of producing text as output.

2.1.2 Reinforcement Learning

This subsection briefly introduces some concepts of Reinforcement Learning (RL) to better understand where the last contribution of this thesis is in context with the field.

To use optimization algorithms such as SGD, the model's objective function must be differentiable [23]. For instance, the language modeling objective is equivalent to the maximum likelihood function [48], which is differentiable. When the objective of the model is not a differentiable function, Reinforcement Learning (RL) is used. This is particularly useful in environments such as games where the goal of the model is to 'win' the game, or achieve a high score as we will see in Chapter 6. In such settings, the classical learning procedure is to let the system self-improve by interacting with a given environment (simulators or game engines) during multiple episodes. In practice, given a current 'state' s_t the model provides



(a) Online Reinforcement Learning. (b) Offline Reinforcement Learning.

Figure 2.5 Reinforcement Learning loop between an agent and an environment

'action' a_t to the environment, and the environment returns the resulting reward r_t and the next state s_{t+1}. This loop is repeated until a final state is reached. The goal of the agent is to learn an optimal policy $\pi(s_t) = a_t$ that maps states to actions. A policy is considered optimal if it maximizes the expected sum of discounted future rewards (also known as the expected

return): $E[G_t] = E[\sum_{k=0} \gamma^k r_{t+k}]$, with γ being the discount factor [63]. This interactive setting is referred to as "online" reinforcement learning since the model has direct access to the environment and can observe the consequences of its evolving behavior. Another paradigm called "offline" reinforcement learning is to learn an optimal policy from a fixed collection of previous interactions. The dataset of interactions can come from expert policies if available (Imitation Learning, Behavior Cloning) or from non-optimal policies (Reinforcement Learning via Supervised Learning - RvS [64]). Offline RL is often used in real-life settings where exploration is risky or costly due to humans involved in the loop [65]. It has the advantage of learning from data rather than from interactions when the said interactions are impractical or dangerous. However, unlike online RL, the agent cannot 'test' its behavior against the environment while learning, which brings multiple challenges such as distribution mismatch where the policy learned from the data is not optimal for the desired environment [65]. In this setting, the goal of the agent is to learn a behavior that will maximize its expected return once deployed in the environment. See Figure 2.5 for a pictorial illustration of the two main approaches.

The last contribution of the thesis leverages the advancement in the representation power of Transformer language models to propose an offline RL approach to text-based environments.

2.2 Systematic Generalisation

This section defines what is meant by systematically generalizing and presents some influential work on the topic. Note that 'systematic generalization' and 'compositional generalization' will be used interchangeably in this thesis.

In natural language, systematic compositionality is defined as the ability to comprehend and create an unlimited number of new combinations using known elements [33, 34, 66]. Natural language offers a good framework for this as language can be broken down into small reusable units (words) that can be combined into complex sentences expressing an infinite range of meanings. One of the simplest examples is if you understand the sentence "*Alice loves Bob*", you can also understand the sentence "*Bob loves Alice*".

2.2.1 Datasets

SCAN Systematic generalization has gained attention over the years for its significance in assessing the capabilities and limitations of neural networks. Lake & Baroni [67] investigated the systematicity of language models by proposing a synthetic instruction-following task named SCAN. The goal of the task is for a model to be able to interpret instructions that

were not seen during training but with words that have been seen during training. For example, a model should understand commands such as "jump twice" after being trained on "jump", "run twice", and "walk twice". This view of systematic generalization puts the emphasis on neural network learning and the design of train/test dataset splits. In their investigation, Lake & Baroni [67] show that RNN language models [21, 47] do not learn general rules on how to compose words and fail to generalize in a compositional manner.

Inspired by neuroscience [68], Russin et al. [69] propose a novel model called "Syntactic Attention" that separates systems for syntactic and semantic processing. Given a sentence, a semantic module processes words with a simple linear attention, while a syntactic module processes words with an RNN, allowing to capture temporal information. The syntactic module then determines the attention it requires over the word representations from the semantic module at each step during decoding. The resulting model achieves better results on SCAN [67] than models without any hand-engineered features.

SQOOP Bahdanau et al. [35] investigate the compositional generalization capacity of visual question-answering systems by proposing a synthetic task called "Spacial Queries On Objects Pairs" (SQOOP) which consists in answering yes/no questions about spacial relationships of objects in images. In particular, the SQOOP dataset consists of 64x64 RGB images containing symbols and questions of the form "*is there a letter A left of a letter B?*". In this context, the authors characterize a model that can systematically generalize if it is "able to reason about all possible object combinations despite being trained on a very small subset of them" [35]. Their experimental results show that neural module networks (NMNs) [70] generalize more systematically than traditional multi-modal architecture such as FiLM [71] leveraging RNNs and convolutional neural networks (CNNs). However, NMNs rely strongly on their underlying module layout, which, when learned end-to-end, does not allow for strong compositional generalization.

CFQ One limitation common to most of the previous works presented above is that testing compositional generalization is often done in synthetic scenarios or tasks, which are not present in more realistic data. This is what the Compositional Freebase Questions (CFQ) dataset [72] aims to solve. As mentioned previously, testing systematic generalization puts the emphasis on the design of train/test data splits. As a technique to quantitatively estimate the suitability of a dataset split for measuring compositional generalization, the authors introduce the "distribution-based compositionality assessment" (DBCA) method. This technique can also be used to create new datasets. As an example, they introduce the CFQ dataset consisting of realistic questions that are mapped to their corresponding SPARQL

queries, which can thus be used to train semantic parsing models.

CLUTRR Since then, other work focusing on non-synthetic datasets explored systematic generalization. One dataset that is used multiple times in this thesis is the Compositional Language Understanding with Text-based Relational Reasoning (CLUTRR) dataset [36]. In this work, the authors introduce a benchmark to generate synthetic graphs based on kinship relationships. CLUTRR proposes a graph version and multiple text versions. The text versions are random-order linearizations of (entity, relation, entity) triples based on templates. Both a simple template and a more natural amazon mechanical turk (AMT) template are available. The AMT templates rephrase every relation in the graph with a natural language scenario involving the entities of interest, for instance "*Kristin and her son Justin went to visit her mother Carol on a nice Sunday afternoon. They went out for a movie together and had a good time.*". The task consists of predicting the relation between two entities (in this case persons) given the (family) graph/story they are in. Questions about entities are constructed such that their relationship can only be discovered by traversing the family graph and combining relationships along the way. In the example above, to infer the relationship between *Justin* and *Carol*, one must combine the facts that *Justin is the son of Kristin* and that *Kristin is the daughter of Carol*. The minimum number of combinations of facts required to answer a question is referred to as the number of 'steps' (or 'hops') or the 'depth' of a question. The interesting aspect of CLUTRR is that with a fixed set of entities, relations, and family rules, graphs can be of arbitrary complexity with questions requiring any number of reasoning steps. This allows to test for systematic generalization by training models on some graph sizes and testing on others, after making sure that all entities, relations, and family rules are present in the training set. This is exactly what is first studied in Chapter 4.

RuleTaker / ProofWriter Around the same time as the work presented in Chapter 4, Clark et al. [1] propose a synthetic question-answering dataset called RuleTaker to investigate the capacity of Transformer language models to reason over knowledge formulated as natural language. It was later updated with the ProofWriter paper [37] and renamed accordingly. The goal of this task is for a model to predict if a given fact is true, false, or unanswerable based on a given collection of true facts and logical rules. For instance, a fact can be "*Dave is green*" and a logical rule can be "*All rough people are green*". Multiple logical rules must often be composed together to verify if a query is true or false. Interestingly, the authors released multiple train/test splits of their dataset that require a different number of combinations of rules to verify the query. The number of combinations of rules is often referred to as the

number of 'steps' (or 'hops') in a proof, or the 'depth' of a proof. As such, one can decide to test for compositional generalization by training models on questions requiring $\leq n$ number of hops, and test on questions requiring $> n$ number of hops, similarly to CLUTRR [36].

2.2.2 Case Studies in NLI and Semantic Parsing

NLI In an effort to better understand the limitations of neural networks to systematically generalize, Dasgupta et al. [28] present a natural language inference (NLI) diagnostic test dataset to study the sentence representations of neural networks during training. Their results show that by default neural systems learn heuristic strategies rather than compositional rules. However, the authors were able to improve results by modifying the training distribution with various data augmentation techniques. Nevertheless, they still observed shortcomings in this generalization behavior. Around the same time, Andreas [73] also proposed a "good-enough compositional data augmentation" (GECA) to improve performance on the SCAN dataset [67] and a semantic parsing task.

Goodwin et al. [29] evaluate systematic generalization in an NLI setting with a linguistic perspective. In particular, they design controlled probes, metrics, and test cases to observe the failure cases of neural architectures. Similarly to Lake & Baroni [67], they observe that Natural Language Understanding (NLU) systems can achieve high overall performance without being systematic.

Semantic Parsing Zheng & Lapata [74] propose to improve the generalization of models in the context of semantic parsing, which consists of predicting the database query corresponding to a natural language question. Their approach is to decompose the problem into two easier tasks: first tagging each token in the utterance with its corresponding semantic tag, and then using both the original tokens and their predicted tags to predict the final meaning of the utterance. Evaluated on questions for a flight-booking task (ATIS [75, 76]), questions about US geography (GeoQuery [77]), and more natural questions about Wikipedia tables (WikiSQL [78]), their method seems to perform better than previous baselines and data augmentation approaches such as GECA [73].

Again in the semantic parsing context, Oren et al. [79] analyze the impact of different modeling choices on compositional generalization using multiples datasets such as ATIS [75, 76], GeoQuery [77], Advising [80], Scholar [81] and DROP [82, 83]. Focusing on encoder-decoder architectures, they examine the effect of contextualized representations such as ELMO [84] and BERT [17] on the encoder, grammar-based decoding versus classical left-to-right decoding on the decoder, various attention methods, and downsampling examples from the training

set that have frequent templates. Their results show that many of the techniques introducing inductive biases such as grammar-based decoding, attention mechanisms, and contextual representations improve the generalization performance of their model, but the authors also argue that further innovations are needed for better compositional generalization.

More recently, one such innovation was introduced by Bergen at al., [85] who introduced Edge Transformers as a new model that combines inspiration from Transformers and rule-based symbolic AI. The core idea of Edge Transformers is to learn a vector representation for every edge, that is, for every pair of input nodes (tokens) as opposed to just every node (token), as it is done in traditional Transformer models. This new attention mechanism (called "triangular" attention) objective is to better represent relationships between objects and entities represented as words in sequences of text. For a sequence of length n and vectors of dimension d, traditional Transformers learn contextualized representations for each token, thus the input representation is of shape $n \times d$. Edge Transformers on the other hand learn to represent pairs of tokens, thus their contextualized input representation is of shape $n \times n \times d$. Similarly, the attention scores of shape $n \times n$ in traditional Transformers are augmented to have shape $n \times n \times n$ in order to represent how informative one pair of tokens is for each specific token. Concretely, Edge Transformers are implemented as regular Transformers but with the addition of a new dimension of size n. One obvious drawback with this increased representation power is that the architecture is extremely slow to train. As a result, experimental analysis was done on relatively small datasets compared to any foundational models. Nevertheless, when compared to Transformers of the same parameter size, Edge Transformers perform better on many compositional generalization tasks. The authors evaluated their model on datasets such as COGS [86], CLUTRR [36], and CFQ [72]. In all cases, the Edge Transformer outperforms relation-aware Transformers [87], Universal Transformers [88] and classical Transformer baselines with the same number of parameters.

In the next section, we introduce Neural Theorem Provers (NTPs) [89] as they are efficient and interpretable solutions for systematically composing knowledge.

2.3 Neural Theorem Provers

One initial inspiration for this thesis was the work done by Rocktäschel et al. [89] that seeks to bridge the gap between the neural and symbolic paradigms of artificial intelligence. Symbolic and logical methods can handle complex reasoning under restricted domains while statistical approaches such as neural networks can handle unconstrained data but cannot systematically reason [90].

NTP The Neural Theorem Prover (NTP) is a good example of such a neuro-symbolic system that combines constrained, logical and symbolic solvers with flexible neural systems. [89]. NTPs are end-to-end differentiable deductive reasoners based on Prolog's backward chaining algorithm [91]. They assume a knowledge base (`K`) of facts in the form `[p, A, B]` representing the atom `p(A,B)` with predicate `p` and arguments `A, B`. They also assume a set of logical rules (`R`) of the form `H:- B` such as `p(A,C):- q(A,B),r(B,C)` meaning that the body `q(A,B),r(B,C)` implies the head `p(A,C)` with `p,q,r` being predicates and `A,B,C` being variables. Following the backward-chaining algorithm, NTPs can prove if a goal atom `g(A,B)` is true or false given the current knowledge `K` and rules `R`. The main difference with prolog solvers is that NTPs replace the 'exact match' unification operation with a differentiable Gaussian kernel [89]. Prolog solvers use ('unify') atoms only if their predicate exactly matches the ones present in the set of rules. NTPs are more flexible in that they unify atoms based on their predicate representation similarity as measured by a Gaussian kernel. Predicate representations are learned embeddings. This *soft* unification allows to unify atoms even if their predicates do not exactly match. In addition to this `unify` module, NTPs also define `or` and `and` modules that call themselves recursively to find a valid proof. The `or` module initially takes as input the goal atom to be proven (denoted `g` for simplicity) and applies the `unify` module between `g` and all facts `f` $\in$ `K`. For all successful `unify(g,f)`, the `or` module terminates and returns the proof score as being the minimum similarity measure of all previous unifications. If none of the unifications were successful with the knowledge facts `f` $\in$ `K`, the `or` module calls `unify(g,h)` for all proof head `h` in rules `h:- B` $\in$ `R`. For all successful `unify(g,h)`, the `or` module then calls the `and` module on each body atom `B` of the rule that was unified with `g`. The `and` module takes in a list of atoms, (now considered as sub-goals to be proven), and recursively calls the `or` module on each of them. This process finishes when all possible proofs of a given depth are explored. Each proof is given a score, defined as the minimum of all the substitution scores in the proof. Each substitution score is defined as the similarity between the two predicates. The successful proof with the maximum score is returned as the final answer and the goal atom is considered true. If none of the proofs were successful, the goal atom is considered false.

Predicate embeddings are learned by minimizing the cross-entropy loss on the proof score of true facts (equivalent to maximizing proof scores of true facts). One interesting aspect of NTPs is that rules can also be learned in this process. That is, instead of relying on hard-coded rules with known predicates such as `p,q,r` in `p(A,C):- q(A,B),r(B,C)`, one can instantiate unknown embeddings for rule predicates (ie: θ_p`(A,C):-` θ_q`(A,B),`θ_r`(B,C)`). This allows learning a representation for the most likely set of predicates that satisfy this type of rule. At inference time the unknown rule predicates can be deduced by taking the

known predicate in the knowledge base with the most similar representation. While this relaxes the burden of having to specify the exact logical rules governing the knowledge base, one must still decide the number of rules to instantiate, and their structure (ie: associative, transitive, etc...).

Originally, NTPs are used to reason on knowledge bases and solve the link prediction task that asks if a link exists between two nodes in a graph. The authors demonstrate that NTPs can perform this task as well as state-of-the-art models on small knowledge bases but they have the advantage of being fully interpretable since they also generate proof traces. One limitation of NTPs is that their procedure is time-consuming and thus does not scale well in applications with a large number of rules and facts.

NLProlog is an extension of NTPs to work with text datasets [92]. Since the number of candidate proofs grows exponentially with the number of rules and the number of hops to perform, NTPs cannot scale to natural language. NLProlog can use facts from databases like NTPs but also facts present in text documents. To do so, Weber et al. [92] extract entities from each sentence using Spacy[1] and consider all the remaining text in one sentence as a single predicate binding the entity tokens present in that sentence. If a sentence has more than two entities, all possible entity pairs are considered and the predicate string includes the other entities as regular text. Predicate strings are encoded by a fixed sentence-to-vector model and a two-layer MLP network that is trained to maximize the likelihood of true facts. The proof search procedure is then identical to NTPs. In order to reduce the run time complexity of the proof search, the authors keep track of the most probable proof and only look for better proofs than the previous best one. This is done by defining a unification threshold that is always set to the most likely previous proof score.

NLProlog is evaluated on the multiple choice WikiHop and MedHop QA datasets [93]. All candidate answers are run through the model and the candidate that has the highest final proof score is returned as the answer. The proposed method achieves competitive results on MedHop and outperforms two baseline models on a subset of WikiHop.

GNTP Greedy Neural Theorem Provers (GNTPs) [94] is an extension of NTPs and NLProlog to work with both text and knowledge graphs and be even more computationally efficient. This allows the authors to experiment on larger knowledge bases by several orders of magnitude. They do that by modifying the `or` module in two ways: (1) by efficiently selecting the top-k_f facts that are most likely to prove a given goal, instead of unifying the

[1] https://spacy.io

goal with all the database facts, and (2) by selecting the top-k_r rules that are most likely to give a high scoring proof, rather than expanding all rules in the database in order to prove a given goal. The authors select the top-k_f facts based on their representation similarity with the goal to be proven based on an efficient GPU implementation of Nearest Neighbour Search [95]. The top-k_r rules are selected similarly based on their head similarity with the goal to be proven.

In addition, when learning rule predicates, the model introduces additional parameters for each predicate as previously explained ($\theta_p \in R^k$). In order to reduce the number of parameters, GNTPs introduce an attention mechanism [21] that represents unknown rule predicates as a linear combination of known predicate embeddings. The known predicates are extracted from facts in the knowledge base and their representation is learned during training. In cases where the predicate embedding dimension k is much larger than the number of known predicates, this method allows learning fewer parameters. GNTPs outperformed NTPs on small databases and produced competitive results on larger knowledge bases such as WordNet [96] and Freebase [97]. To evaluate their method on natural language, the authors manually replaced a few predicates with simple template sentences linking two entities.

CTP One limitation of NTPs [89] (and potentially GNTPs [94]), is that they need to consider all possible proof paths for explaining a goal, thus making them unfit for large-scale applications. In an effort to resolve this limitation, Minervini et al. [98] propose Conditional Theorem Provers (CTPs) as an extension over NTPs and GNTPs. In this work, when proving a goal G, instead of selecting all possible rules in the knowledge base, CTPs rely on a $\texttt{select}_\theta(G)$ module that is responsible for generating the most appropriate rules given the current goal to prove. This is done by reformulating the goal G into a rule of the form `H :- B1,B2` with three differentiable functions (such as linear projections): $f_H(G)$, $f_{B1}(G)$ and $f_{B2}(G)$, each mapping the goal G to the most appropriate head and body predicates. Being differentiable, the `select` module is learned alongside the other parameters of the system via backpropagation. Evaluated on the graph version of CLUTRR [36] as well as the Countries [99], Nations, UMLS, and Kinship datasets [100], CTPs outperform GNTPs, graph attention networks (GATs) [101], and recurrent models such as RNNs [18], LSTMs [19] and GRUs [20]. In addition, CTPs show strong generalization to unseen reasoning depths on CLUTRR, thus highlighting the benefits of neuro-symbolic approaches.

Nevertheless, one limitation of Neural Theorem Provers in their current form is that they are not flexible enough to be integrated into auto-regressive language models, and thus cannot leverage the vast amount of knowledge represented in natural language. NTPs are designed to prove or disprove statements that are either true or false, making their output binary-

like. As a result, if multiple candidate solutions exist to a problem, each must be evaluated individually until a provably true solution is found.

2.4 Graph Based Methods To Reason Over Text

Another inspiration for this thesis was the idea of using graph topology to build better reasoning systems.

KG-MRC Some of the first inspiring work for this thesis was "*Building Dynamic Knowledge Graph from text using Machine Reading Comprehension*" [102]. In this work, the authors leverage a machine reading comprehension (MRC) system [103] based on bidirectional LSTM networks [19] to build a simple bipartite graph tracking entity locations in procedural texts. They evaluated their method on the ProPara dataset [104] which measures the capacity of models to track state changes of entities in procedural text. The entities to track are tokens identified in an input document and the *state* can be any span of text in the document. This task is thus a span-prediction problem. The authors process each sentence in a document incrementally: for each sentence s_t and sentence entity e_i they query their MRC system to predict a span of text $y_{i,t}$ describing the state of e_i at time step t. The MRC system is conditioned both on sentences up until s_t, and the graph representations from the previous time step G_{t-1}. The span predictions $y_{i,t}$ are then encoded for all entities e_i, and used to update the representation of the graph G_t. When reading the last sentence s_T of the document, the predicted spans $y_{i,T}$ are used to predict the final state of entities e_i.

The idea of tracking entity states in a document is one application of multi-step reasoning. For instance in the following story "*The box is in the kitchen. Alice takes the box. Alice goes to the living room. Where is the box?*", the model needs to track the location ('state') of the box ('entity') across multiple sentences. While the above example is a simple toy problem, the same methodology can be applied to more complex situations involving multiple entities and states across longer documents. The first contribution of this work presented in Chapter 4 extends this idea of automatically tracking entities' states but by generating reasoning steps in text rather than a graph, with unlabeled documents, and using multiple types of relations.

HGN Another graph-inspired work applied to multi-step reasoning on text is Hierarchical Graph Networks (HGNs) [105]. HGNs are introduced as a solution to the multi-hop question answering dataset HotpotQA [38] in which the task is to answer questions based on a collection of Wikipedia documents. Questions are designed by humans to require multi-step reasoning, by combining information from usually two documents. HGNs create a graph

for each document with different granularity levels and node types: the top nodes represent paragraphs, each of which is connected to sentence nodes representing sentences in the corresponding paragraphs, and each sentence node is connected to entity nodes representing entities [2] from the corresponding sentences. This collection of trees (each rooted at its paragraph node) is linked together by adding edges between consecutive paragraph nodes and between consecutive sentence nodes. All the node representations are initialized with BERT encodings, and a message-passing algorithm from graph attention networks (GATs) [101] is used to update node representations.

Overall, this work shows how inductive biases like the ones present in graph networks and in hierarchical topologies can help achieve strong performance on multi-hop QA tasks. Motivated by the fact that not all sources of information can be easily represented by graphs, one objective of this thesis is to add inductive biases to language generation systems and observe similar capacities. The second contribution of this thesis presented in Chapter 5 will show some work towards this objective.

Other methods tried to incorporate external knowledge graphs (KGs) as a way to better inform language models and make them better at various tasks. We describe such methods in the next section, along with other ways to augment language models.

2.5 Knowledge Augmented Language Models

This section describes related methods that were used in the literature to augment language models with additional external knowledge. In particular, the focus is on knowledge graph incorporation, memory-based networks, and input/output augmented language models.

2.5.1 Knowledge-Graph Augmented Language Models

Although multi-step reasoning is not the focus of the following approaches, these works inspired this thesis to explore if "entity-aware" models could be better reasoners in Chapter 5.

ERNIE is a model enhancing language representation with informative entities [106]. The goal of this work is to incorporate external knowledge graphs (KGs) into transformer language models. ERNIE is an encoder-based model made of BERT-style encoder blocks [17] encoding tokens into contextual representations ($X = \texttt{T-Enc}(X)$), followed by "knowledge" encoder blocks encoding tokens and entity representations together ($X, E = \texttt{K-Enc}(X, E)$). Wikipedia entities in the input text are automatically identified with TAGME [107] and their

[2] extracted using https://spacy.io

vector representation (embeddings) are computed using TransE: a method which models relationships by interpreting them as translations operating on the low-dimensional embeddings of the entities [108]. Knowledge encoder blocks concatenate entity embeddings E_e to their corresponding token embeddings X_e and feed them to a multi-head attention mechanism [10]. The output is then split back into separate token and entity representations for the next knowledge encoder layer. Experimental results have demonstrated that ERNIE achieves significant improvements on various knowledge-driven tasks such as entity typing, relation classification, and some GLUE tasks [109].

ERNIE assumes that a one-to-one mapping between entities mentioned in the input text and KB entities is available. The next work removes this assumption and instead selects 30 entity candidates per mention.

KnowBERT is a general method to insert knowledge bases (KBs) into large pre-trained models with a Knowledge Attention and Recontextualization (KAR) mechanism [54]. The authors integrate WordNet [96] and a subset of Wikipedia into the BERT model [17] to create a knowledge-enhanced version of BERT. In particular, the KAR mechanism operates between two Transformer layers in the middle of a pre-trained model. First, a set of (30) entity candidates $E_e = [e_1, ..., e_{30}]$ are retrieved from the KB of interest for each entity mention X_e in the text. An entity linker mechanism is trained to map entity mentions (X_e) to the appropriate KG entity embedding (e_i). This linker is used to compute a linear combination of the candidate entity embeddings for each entity mention: $E_e = \texttt{softmax}(W).E_e$ with $w_i = \texttt{linker}(e_i, X_e)$. Eventually, the resulting vector representation is summed to the entity mention representation ($X = E_e + X_e$) and given as input to the next Transformer layer. After integrating this knowledge, KnowBERT demonstrates improved performance on various tasks such as relationship extraction, entity typing, and word sense disambiguation [54].

EaE Another example of incorporating entity knowledge to better inform language models is the work done by Fevry et al. in "*Entities as Experts: Sparse Memory Access with Entity Supervision*" (EaE) [110]. The authors created an encoder-only Transformer model that incorporates entity knowledge from English Wikipedia text. This was accomplished by adding an entity memory containing Wikipedia entities whose embeddings are updated each time the corresponding entity is mentioned in the text. Similarly to KnowBERT, the authors introduce a custom neural module between Transformer layers.

In particular, tokens are first encoded with four Transformer layers. These representations are then used to identify entity mentions in the text by predicting beginning-inside-outside (BIO)

labels [111]. Each entity mention vector representation (X_e) is then used to retrieve its top-k closest entity embeddings from the memory based on cosine similarity ($E_e = \texttt{top-k}_e(X_e, e)$). Eventually, each entity mention embedding is summed with a weighted combination of its retrieved entity embeddings ($X_e = \texttt{LayerNorm}(X_e + W.E_e)$). The model is trained with (i) the masked language modeling (MLM) objective [17] which corresponds to predicting masked tokens from the input text, (ii) a mention detection objective which corresponds to predicting BIO labels [111] on the input sequence, and (iii) an entity linking objective which corresponds to predicting the correct memory for each entity mention in the text. Entities in the text are automatically extracted with the help of the Google Cloud Natural Language API[3] in order to provide supervised data to train these objectives. Experimental results on question-answering tasks like TriviaQA [112] and WebQuestions [113] show that this model has more factual knowledge than larger-sized BERT models.

To summarize, ERNIE and KnowBERT are systems that have the ability to access external knowledge graphs. Similarly, EaE integrates Wikipedia entities with the help of a "memory". However, this memory is closer to a knowledge base than an external memory since the model does not add new information or remove old information from it, it only learns to represent a fixed set of Wikipedia entities. The next section first introduces the origins of memory nets before summarizing more recent work such as Verga et al, [114] that will resolve some of the limitations in EaE by introducing an explicit fact memory.

2.5.2 Memory Based Networks

Initial brainstorming for this thesis considered memory-based models as a way to save logical rules and be able to re-use them arbitrarily when needed. Although not in the resulting contributions of this thesis, this idea is what evolved into some of the current explorations described in Chapter 8 involving external API calls performing logical operations systematically. Additionally, some work summarized below contributed to the intuition that entity-aware models could be better logical reasoners, as further explored in Chapter 5.

Memory Networks Weston et al. [115] introduced Memory Networks in 2014 as a new type of neural model capable of writing to and reading from a dynamic memory in the context of question answering. A memory is defined as a list of string vector representations. The model then learns to write, read, and edit memory slots with specialized learnable components. An input feature map first converts input tokens to an internal feature representation: $I(x)$. A generalization module updates memory slots with the new information:

[3]https://cloud.google.com/natural-language/docs/basics

$m_i = G(m_i, I(x))$. An output feature map then retrieves the top-k most relevant memory slots based on the current input: $o_i = \text{argmax}_j O(m_j, [x, o_1, ..., o_{i-1}])$. Eventually, a response module decodes the output features into tokens: $r = \text{argmax}_w R(w, [x, o_1, ..., o_k])$ for all w in the vocabulary of tokens. The authors argue that any classical ML tool such as RNNs, support vector machines and decision trees can be used as implementation for each of these modules (I, G, O, and R). Experiments on a simulated question-answering task showed the reasoning advantages of such models compared to vanilla RNNs.

The limitation of memory networks is that they require full supervision during training: input questions, output answers and supporting sentences must be available. In addition, they can be challenging to train end-to-end because they need supervision at each layer.

End-to-end Memory Networks In many question-answering tasks however the model needs to perform multi-step reasoning as previously motivated. In this case, it would be beneficial for the model to access the memory multiple times (multiple "hops") before generating an answer. This is what Sukhbaatar et al. proposed in "*End-to-end Memory Networks*" [116]: an extension of memory networks.

Formally, sentences from the context are encoded twice: once to represent input memory features ($A(x)$) and once to represent output memory features ($C(x)$). Questions are encoded separately ($B(q)$). For each memory lookup, the inner product between the question and memory input features is used to compute attention weights ($W = \text{softmax}(B(q)A(x))$). The output of the memory lookup is then the weighted linear combination of the sentences in memory: $O = C(x) \times W$. Eventually, the question representation gets updated with this memory output ($B(q) = B(q) + O$) and can be reused for another memory hop. The above procedure is repeated a fixed number of times before the final answer is predicted. End-to-end Memory Networks also simplify the original approach to be able to train the model end-to-end with less supervision while achieving comparable results on question-answering and language modeling tasks.

FaE Since then, multiple works have been experimenting with memory-based models, and more recently with memory-based Transformer language models. One such example is the work done by Verga et al. in "*Facts as Experts: Adaptable and Interpretable Neural Memory over Symbolic Knowledge*" (FaE) [114]. This work proposes an interface between explicit, symbolically bound memories and sub-symbolic distributed neural models. The authors extend the "entities-as-experts" (EaE) model [110] by introducing a symbolic memory of fact triples which are made of learned entity representations, and as in EaE, the entity repre-

sentations are learned end-to-end. The model makes predictions by integrating contextual embeddings with retrieved knowledge from the external memory, where those memories are bound to symbolic facts which can be added and modified. Importantly, the external fact memory can be updated with any new facts without retraining the whole network.

The fact memory is defined as a dictionary mapping head pairs {subject s_i, relation r_i} to tail sets {objects $o_{i;1}, ..., o_{i;n}$}. Note that all 'subjects' and 'objects' are entities with their corresponding embeddings shared with the entities in the EaE model. Each head pair is encoded as the concatenation of the subject embedding and relation embedding $head_i = W_1.[s_i; r_i]$. Each tail set is encoded as a weighted sum of each object embedding ($tail_i = W_2.[o_{i;1}, ..., o_{i;n}]$). The model first encodes a piece of text with the entity-enriched Transformer (EaE [110]). After the final Transformer layer, the contextual representation of the masked token (recall that encoder-based Transformers are trained to predict masked tokens in sentences) is used to 'query' the fact memory and the top-k {subject, relation} pairs are selected. Eventually, a weighted sum of the corresponding top-k object tails is computed and used to update the final representation of the mask token before predicting its label from the vocabulary.

Experimental results show improved performance on question-answering datasets such as Freebase [97] and WebQuestion [113] compared to previous state-of-the-art and EaE. The authors also showed that the fact memory can be updated without retraining the network as long as entities and relations from the same vocabulary as the one seen during training are used.

EMAT Another example of recent memory-based Transformer language models is the Efficient Memory-Augmented Transformer (EMAT) that combines the strengths of parametric and retrieval-augmented models [117]. Fully parametric methods such as BERT [17], T5 [52] or GPT-3 [50] trained with the language modeling objective on vast amounts of text data have been shown to contain some knowledge about the world in their parameters. This knowledge is computationally cheap and fast to retrieve once the model is trained [62,118], however, such models can also hallucinate false information. On the other hand, retrieval-based methods such as REALM [119], RAG [120] or FiD [121] provide truthful information, by retrieving relevant passages from external knowledge bases to inform generation. However, these methods are often computationally intensive and slow. EMAT tries to combine the benefits of both approaches by building memory within the parameters of the model with truthful QA information. In particular, a memory of key-value pairs is built in which keys are contextual representations of natural language questions, and values are contextual representations of answers based on the Wikipedia-based PAQ question-answering dataset [122]. Given a text input sequence, the model will dynamically retrieve the most relevant memory based on its

key representations and inform the generation with the corresponding value representations. Evaluated on WoW [123] and ELI5 [124], EMAT produces both faster and more accurate answers than retrieval-based methods.

Non-Symbolic Memory Transformers While the previous works introduced a symbolic memory with interpretable content, memory-augmented Transformers is a field that has also seen a lot of contributions with non-symbolic approaches. Some of them are summarized below without going into the same amount of detail as the previous works since they were less influential for the creation of this thesis as they do not focus on the combination of symbolic and neural approaches.

Dai et al. [25] introduced back the concepts of memory through recurrence in Transformer models with the "Transformer-XL". This model addresses the fixed-size input limitation of Transformers by diving long sequences into sub-sequences and caching each layer's hidden state into a fixed-sized queue that is re-used in the following sub-sequence.

Gupta et al. [125] proposed "global memory augmentation for Transformers": a method that prepends M memory tokens to an L-token long input sequence with $M << L$. The L input tokens perform sparse attention to each other, and dense attention to the M memory tokens. The M memory tokens do dense attention to the entire input sequence $(L + M)$.

Similarly, Burtsev et al. [126] proposed the "memory Transformer" which also prepends `[mem]` tokens to input sequences. The authors propose three variants: (i) the *Mem Transformer* which processes the extended sequence with traditional Transformer layers, (ii) the *MemCtrl Transformer* which has a dedicated sub-network for memory tokens, and (iii) the *MemBottleneck Transformer* which process first the memory tokens and then the remaining of the sequence based on the final memory contextual representation.

Wu et al. [127] propose the "Memformer": an efficient sequence modeling Transformer that utilizes an external dynamic memory to encode and retrieve past information. The authors propose an encoder-decoder Transformer architecture with a memory component that gets updated between each encoder layer. The encoder is responsible for reading from and writing to the memory, while the decoder is standard.

Feedback Transformers [128] addresses some limitations of decoder-only Transformers by introducing feedback memories. This model processes each token sequentially through all layers, and summarizes the hidden representation of all layers into a 'memory' representation so that subsequent tokens can use this memory representation in their contextual representation. Simply put, Feedback Transformers add connections from top to bottom layers by introducing a memory cell between each time step in a sequence.

2.5.3 Input/Output Augmented Language Models

Besides graphs and external memories, language models can receive additional information in text modality directly in their input sequence (input augmented), or as target output sequence (output augmented). This sub-section covers some related work augmenting language models in such ways.

Dynamic Entity Representation in Neural LM Incorporating knowledge about entities into neural language models has been a common approach to enhance their capabilities. Before the introduction of Transformers, Ji et al. [129] trained an Entity Neural Language Model to predict sequences of entities with an LSTM [19]. At each sampling step, they predict (i) the next word (traditional language modeling objective), (ii) a binary variable indicating the presence of an entity at that position, (iii) a categorical variable indicating the entity ID, and (iv) a categorical variable indicating the number of remaining tokens in the mention of this entity. Experimental results showed that this "entity aware" model achieves lower perplexity (equivalent to higher likelihood) and better results on co-reference resolution and entity prediction tasks than a 5-gram language model and a traditional LSTM. This is an example of augmenting the language model by training it to predict additional information (output augmented). Training a model to predict additional information can help ensure that the model has that information represented in its parameters. However, one drawback is that this may reduce the model's representation power for its primary task.

KALM More recently, Rosset et al. [130] present "*Knowledge Aware Language Models*" (KALM) an alternative pre-training technique for Transformer language models in order to make them more 'knowledge aware'. In addition to pre-training a model with the traditional language modeling objective, this work proposes an entity prediction task as pre-training similar to Ji et al. [129]. Entity mentions in the text are identified with a pre-existing external dictionary. For each token belonging to an entity, the authors used a contrastive loss on the final Transformer layer to teach the model to differentiate between the corresponding entity and any other one. In addition, KALM incorporates entity representations at the input level by summing the word embedding and its corresponding entity embedding together before feeding it to the Transformer layers. This is an example of both input- and output-augmented language models. In particular, the model learns an additional embedding for each entity so that tokens belonging to an entity are represented by the sum of their word and entity embeddings. Experimental results show that such a pre-training method yields models with greater factual knowledge according to the LAMA knowledge probing task [62] compared to

vanilla GPT2 models [24]. In addition, it is also reported that such models perform better than a GPT2 model of the same size on TriviaQA [112], Natural Questions [131] and Web Questions [113], in a zero-shot setting; and achieve competitive results when compared to larger models.

Abstraction Augmented LMs Previously covered work such as KALM [130], ERNIE [106], KnowBERT [54] and EaE [110] augment language models with entity knowledge but do not add any *abstract* information. Indeed, the additional knowledge added in these models is grounded entity tokens which are specific in nature. Similarly, for FaE [114] covered in Section 2.5.2, the additional information stored in memory was grounded facts. One objective for this thesis is to leverage reusable and abstract knowledge such as logic rules involving different variable types for instance. The following relevant works introduce abstraction into language models by incorporating syntactic information, word senses, predicate-argument structures, and hypernym relations.

Syntax Augmented LMs Prior work by Swayamdipta et al., [132], Eriguchi et al. [133], and Nadejde et al. [134] incorporated syntax information into language models. This was done similarly to Ji et al. [129] and Rosset et al. [130] by introducing an auxiliary loss to the model (output augmented language models). Experimental results show that models trained to also predict syntactic information achieve stronger performances on various tasks such as PropBank semantics [135] and Neural Machine Translation. More recently, Sundararaman et al. [136] incorporated part-of-speech (POS) tags [111] into the input embedding of a BERT model (input augmented). Experimental results show improved BLEU scores on machine translation and higher accuracy than baselines on the GLUE benchmark [109].

WordSense Augmented LMs Levine et al. [56] trained a BERT-like model to learn word senses. They gave the model access to WordNet supersenses [96] at the input level and as an additional training loss. The model, named SenseBERT, is pre-trained to predict not only masked words but also their WordNet supersenses. By infusing word sense information into BERT's pre-training signal, the authors explicitly expose the model to lexical semantics when learning from a large unannotated corpus. This results in achieving better performance than other baselines on the SemEval Word Sense Disambiguation task [137].

Predicate-Argument Augmented LMs Moosavi et al. [138] propose to improve the robustness of language models to data biases by augmenting training datasets with predicate-argument structures. Each sentence in the input sequence is extended with its predicate-

argument structure in text format with the introduction of special tokens such as "`[PRD]`" and "`[ARG0]`". This is thus an example of input-augmented language models. The authors argue that this provides a high-level abstraction over different realizations of the same meaning and helps the model recognize important parts of sentences. They examine the impact of this linguistic augmentation by training a BERT-base model [17] with PropBank-style semantic role labelling [139] on MultiNLI [140] and SWAG [141] datasets. Experimental results show that incorporating predicate-argument structure in the input sequence during training makes the model more robust to adversarial examples during inference even if syntactic information is not available on sequences used at runtime.

Hypernym Augmented LMs More recently, Porada et al. [142] and Bai et al. [143] augmented Transformer models with hypernym relations from the WordNet [96]. Porada et al. [142] replaced input tokens by their hypernyms to evaluate the plausibility of events. Results show that their model is able to better predict human plausibility judgement than other baselines. Bai et al. [143] explore the effectiveness of class-based language models [144] in the context of Transformer LMs. In particular, they substitute a subset of the tokens in the input and output sequences by their WordNet hypernyms and train a model with the classical language modeling objective. The amount of tokens replaced by their hypernyms starts high and is slowly decayed during training. Their curriculum strategy, called Hypernym Class Prediction (HCP) achieves lower perplexity (higher likelihood) on the Wikitext-103 [145] and arXiv [146] datasets.

Overall, all these prior works leverage the general idea of explicitly giving more abstract knowledge to language models. Taking inspiration from all these works, Chapter 5 will leverage entity *types* as an abstraction technique with the objective to improve the reasoning skills of Transformer language models.

2.6 Text-Interactive Environments & Corresponding Methods

Thus far, proof generation and traditional NLP tasks like question-answering (QA), semantic parsing, and natural language inference (NLI) were discussed. Another paradigm that can also be used to evaluate the multi-step reasoning capacity of language models is text-based interactive environments (also referred to as 'text games'). Usually used in Reinforcement Learning (RL), these environments simulate virtual worlds in which an agent must interact with the world by performing actions, which corresponds in this case to generating text commands. Similarly to a dialogue, the environment then responds to the agent with the next state text description. This section describes multiple text-based environments and

some of their most popular online reinforcement learning solutions.

Q-learning & DQN As introduced in Section 2.1.2, the goal of an agent in RL is to select action a_t at state s_t that maximizes the expected sum of future rewards $G_t = \sum_{k=0} \gamma^k r_{t+k}$ with discounting factor γ controlling the importance of future rewards. Q-learning [147] is a method used to learn an optimal $Q(s, a)$ value function. The agent first starts with a random estimation of Q and iteratively updates its estimate according to the Bellman equation [63] by interacting with the environment and observing rewards: $Q_{i+1}(s_t, a_t) = E[r + \gamma \max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1})]$. If successful, the agent can then select actions based on $a_t = \texttt{argmax}_a Q(s_t, a)$. Mnih et al. [148] introduced Deep Q-Networks (DQN) and showed that using deep neural nets only (in contrast to hand-engineered features) to learn such Q-value functions can yield strong performance in the context of Atari games.

LSTM-DQN One limitation of Deep Q-Networks (DQN) (and Q-learning in general) is that they require evaluating the Q-value function for all possible actions. This restricts their use to environments with small action spaces and makes them unsuitable for environments with combinatorially large action spaces such as text environments. As a result, text games as a framework for testing RL and NLP models did not get a wide interest until the work done by Narasimhan et al. [149] introducing LSTM-DQN. The authors combined LSTMs [19] with DQNs [148] in order to learn an optimal Q-value function from text sequences describing the current state. States are first encoded with an LSTM into a fixed-size vector before being passed as input to a DQN. The authors evaluated their approach on two custom text-based games (*Home World* & *Fantasy World*) in which actions can be any combination of a *verb* and an *object*. In order to deal with this combinatorial action space, the authors used two output layers on their DQN to predict two separate Q-values based on the current state representation: one over the list of verbs, and one over the list of objects. The final action would then be the one with the maximum-valued verb and the maximum-valued object based on the current state representation. The resulting LSTM-DQN model yielded better performance than previous bag-of-word baselines.

DRRN Shortly after LSTM-DQN, He et al. [150] introduced the Deep Reinforcement Relevance Network (DRRN) that also learns to discover an optimal Q-value function in the context of text worlds. However, their method does not tackle the combinatorial action space in text environments and instead assumes to have access to a small set of candidate actions for each state. In particular, the authors compared multiple approaches to learning a Q-value function: (i) Max-action DQN, which corresponds to encoding s_t and all possible

actions $\{a_t^i\}_{i=1}^n$ together to predict all q-values $Q(s_t, a_t^1)$, ..., $Q(s_t, a_t^n)$ in one forward pass through the network; (ii) Per-action DQN, which corresponds to encoding s_t and a_t^i together to predict one $Q(s_t, a_t^i)$ value function per forward pass; and (iii) DRRN, which corresponds to encoding s_t and a_t^i separately and predict $Q(s_t, a_t^i)$ based on the pairwise interaction function (usually the inner product) between their respective vector representations. Another differentiating factor from LSTM-DQNs, is that DRRNs also consider a natural language action space in addition to the state space. Evaluated on two different custom text-based games (*Saving John* & *Machine of Death*) with a small action space, the authors showed that DRRNs learn better Q-value functions faster than other approaches.

TextWorld As seen with the previous two works, many teams performed independent research, creating their own text environments to evaluate their methods, making comparisons between different approaches difficult. In an effort to create a common test benchmark for text-based environments, Côté et al. [151] introduced TextWorld, a logic engine to create game worlds all under the same design rules. Agents can interact with TextWorld via a set of primitive actions (e.g. '*take*', '*move*', '*eat*', '*drop*' etc...) that can be compositionally combined with objects (e.g. '*knife*', '*map*', '*microwave*', etc...), locations ('*river*', '*kitchen*', '*garden*', etc...) and spacial and logical connectors ('*and*', '*on top of*', '*inside*', etc...) based on simple grammatical templates. By default, the library provides a "Home" and a "Medieval" environment in which goals are generally to put a specific object in a certain state (e.g. "*put the green book on top of the library shelf*"). Users can specify the number of rooms, the number of objects and the number of steps or commands required to accomplish the goal. The TextWorld game generator will then generate a random environment with a random task accordingly. Other environment features such as partial state observability, state transition stochasticity, and reward sparsity can be controlled. TextWorld can thus be used to design environments with specific difficulty features and allows to research generalization capacities of language models in the context of Reinforcement Learning. The release of TextWorld resulted in multiple text environments such as Treasure Hunter [151], Coin Collector [152], CookingWorld [153], and the QAit dataset [154].

Jericho Shortly after the release of TextWorld, Hausknecht et al. [40] released a collection of 56 interactive fiction (IF) games through the Jericho python library. Similarly to TextWorld, the Jericho games are fully textual: states and actions are described in natural language only. However, unlike TextWorld, the Jericho library consists of 'real' games designed by humans to entertain human players in the 1970s back when computers were not as powerful as today. As a result, the Jericho games are generally much more complex in nature, covering a variety

of genres (dungeon crawl, sci-fi, mystery, comedy, horror) with long quests (requiring more than 1000 actions depending on the game), large state space, sparse rewards, partial state observability (mentioned as 'pitch-black' rooms in games), stochastic state transitions, and numerous sub-quests per game. The authors collected these games from the online Interactive Fiction Database [4] and created a common python environment to test RL agents on these, similarly to TextWorld. In order to make the environment more accessible for automated agents, the library provides one human experience (sequence of actions) completing the game from start to finish achieving the maximum score for one specific random seed, called the "walkthrough" or "golden" path. Additionally, at every state in a game, the Jericho game engine can provide a list of candidate actions that are valid for that specific state. The work presented in Chapter 6 of this thesis leverages these two pieces of information to propose a solution to these environments.

TDQN Alongside the Jericho games, Hausknecht et al. [40] proposed the Template-DQN (TDQN) model to try to solve some of these games. TDQNs build upon LSTM-DQNs [149] by leveraging template-based action generation. Jericho games are quite complex and can understand multiple action templates. For instance, Zork1 can understand 697 vocabulary words and has multiple templates combining up to 4 words, which would result in $697^4 \approx 200$ billion potential actions. To deal with this added complexity the authors added a third output layer to their DQN that is responsible to compute the Q-value over templates based on the current state. The other two output layers are responsible for predicting the Q-value over verbs and objects in the vocabulary similarly to LSTM-DQN. Another particularity of TDQN is that to represent the current game state, the model takes as input the previous action taken by the agent in addition to the current narrative text, inventory, and room description. Since then, it is common practice to include the previous action as part of the current state representation. The authors evaluated their method on a selection of 33 Jericho games and compared it to random action selection, a heuristic rule-based agent NAIL [155] and DRRN. On many games, DRRN outperformed TDQN because it assumes access to the list of available actions per state. However, TDQN managed to exceed DRRN on a few other games. Nevertheless, all previously proposed methods fail to achieve the games' maximum score (and this is surprisingly still the case in early 2023), thus showing how complex these environments can be.

[4] https://ifdb.org

2.6.1 Recent Online RL Models

More recent approaches were proposed to tackle text-based games such as Jericho. Most of them are online reinforcement learning methods. Some of them are described here, and Chapter 6 will present a new offline method that outperforms previous baselines.

Graph based methods: KG-DQN, KG-A2C, SHA-KG As mentioned in Section 2.4 and 2.5.1 graph-based method can be powerful representation tools. Thus multiple deep RL approaches leverage graphs topology to represent complex state spaces such as in interactive fiction games. Ammanabrolu and Riedl [156] present KG-DQN a deep RL method that combines knowledge graphs and deep Q-networks. The authors design update rules to dynamically construct a graph representing the current state of the game based on past and current observations. This graph tracks object locations ('*chamber,has,bed*'), the location of entrance and exits ('*basement,exit,north*'), and relative room locations ('*chamber,east-of,basement*'). A Graph Attention Network (GAT) [101] is used to embed this information into a state vector. A list of candidate actions (pruned based on current graph) is encoded with an LSTM [19] and multiplied to the state vector to predict Q-values $Q(s_t, a_t^i)$.

Shortly after, Ammanabrolu and Hausknecht [157] present KG-A2C, an agent that similarly builds a knowledge graph dynamically while exploring its environment. The graph is made of <*subject, relation, object*> triples automatically extracted with templates based on past and current observations. Their model is based on template-based action generation similarly to TDQN [40] in order to deal with the combinatorial large action space of text-games. Additionally, they use the Advantage Actor Critic method (A2C) [158] to train their model, which consists of training a model (the 'actor') to predict actions and a model (the 'critic') to evaluate the state value function $V(s_t)$.

Shortly after, Xu et al. [159] introduce Stacked Hierarchical Attention with Knowledge Graphs (SHA-KG) as an extension of KG-A2C. One key difference with previous approaches is that in this work the authors define four sub-graphs within the complete graph representing the current game state: (1) representing the connectivity of visited rooms, (2) representing the objects within the current room, (3) representing the objects in the agent's current inventory, and (4) representing any information that is not linked to 'you' (the agent playing the game). They combine this hierarchy of information with GATs [101] and multiple attention mechanisms [21] to construct a vector representation of the current state. Similarly to previous works, SHA-KG is using template-based action generation to deal with the combinatorial large action space of text-games [40], and is trained with the Advantage Actor Critic method (A2C) [158].

LLM approaches: CALM & DBERT-DRRN As previously mentioned in Section 2.1.1 pre-trained Large Language Models (LLMs) are powerful tools to process text because they encode world knowledge and NLP inductive biases in their parameters [50, 60–62]. As such, multiple methods also leverage LLMs to tackle text-based environments. The work presented in Chapter 6 follows the same motivation and also leverages large pre-trained language models.

Yao et al. [160] decided to leverage large pre-trained language models to deal with the combinatorial action space of text-world games and proposed Contextual Action Language Model (CALM). With games having multiple templates using 1 to 7 words in a vocabulary of hundreds of words, some games can have a potential list of actions in the hundreds of billions. To alleviate this challenge, the authors propose to use a pre-trained GPT-2 model [24] to suggest a set of potential actions to a DRRN agent [150]. In particular, their GPT-2 model is fine-tuned with the language modeling objective to predict actions (token by token, from left to right) given the previous interactions, on sequences of observation-action pairs $(o_1, a_1, ..., o_l, a_l)$. In practice, the authors use a context window size of two steps: they predict a_t conditioned on (o_{t-1}, a_{t-1}, o_t). To generate multiple answers at inference time, the authors use beam search with a beam size of 40 and select the top 30 most likely actions. The current observation o_t and candidate actions $a_{t,1}, ..., a_{t,30}$ are then passed to a DRRN [150] to compute the Q-value function $Q(s_t, a_{t,i})$. Observations and actions are encoded with separate GRU networks [20] before being combined in a multi-layer feed-forward network trained to predict the optimal Q-value function. Evaluated on Jericho games, CALM performs similarly on average to previous methods having access to the list of candidate actions given by the environment such as KG-A2C and DRRN.

Another method leveraging pre-trained language models is the work done by Singh et al. [55] in which they present DBERT-TDQN and DBERT-DRRN, a combination of a DistilBERT model [161] (a smaller yet as powerful version of BERT) with traditional RL methods such as TDQN [40] and DRRN [150]. Specifically, the authors first fine-tune a DistilBERT model with the masked language modeling objective on sequences of observation-action pairs from a collection of game interactions. They then use this fined-tuned Transformer encoder to obtain contextual vector representations of state observations, room descriptions, agent inventory, and agent actions. The resulting vectors are eventually given to GRU networks [20] inside either a DRRN or a TDQN model to predict the corresponding Q-value function. DBERT-DRRN takes as input the current observation, room description and inventory to represent the current state. It also assumes to have access to a small list of candidate actions which are encoded separately before being combined with the state vector representation in a multi-layer feed-forward network trained to predict the optimal Q-value function. DBERT-TDQN

takes as input the current observation, room description, inventory, and previous action to represent the current state. This information is combined in a multi-layer feed-forward network to predict the optimal Q-value function based on action templates, verbs, and objects separately similarly to TDQN [40]. Evaluated on Jericho games, DBERT-DRRN outperforms previous baselines, including CALM.

Both CALM and DBERT-DRRN show that interactive text environments can benefit from pre-trained language models, similarly to more traditional NLP tasks.

2.7 Reinforcement Learning via Supervised Learning

Note that all solutions discussed in the previous section were online RL methods: learning from interacting with the environment. Although not applied to text-based environments, this section discusses some influential work on offline RL methods that inspired the work presented in Chapter 6.

As briefly introduced in Section 2.1.2, offline RL consists of learning an optimal policy from a dataset of previous interactions collected by an agent with the environment of interest. This approach offers the benefit of learning from existing data instead of learning from interactions that may be costly, unfeasible, or hazardous in real-life environments [65]. One limitation though is that the dataset of previous interactions must come from a near-optimal or expert agent in order for the model to learn to imitate it. This is referred to as Imitation learning or Behavior Cloning [63]. However, collecting expert data on environments can also be costly making the approach as impractical [65] as online RL. From this limitation, the research community started exploring methods to learn from non-optimal offline data.

UDRL & RCP One solution proposed by Schmidhuber [162] is to "*turn traditional RL on its head*" calling it upside-down reinforcement learning (UDRL), which consists of mapping rewards to actions. At the same time, Kumar et al. [6] introduced reward-conditioned policies (RCP), which consist of learning policies of the form $\pi(s_t, Z) = a_t$ with Z being a measure of value such as next step reward of total return. In both works, the main idea is to condition the prediction of action a_t on both the current state s_t and a measure of quality Z such as the next step reward. The key observation is that any experience can be used as optimal supervision data if the model is also conditioned on the quality of these experiences. In other words, actions that result in weak returns provide valuable guidance to learning a weak policy. Conditioning on the future effect of taking an action can seem unfeasible or counter-intuitive, but in offline RL, this is possible because one has access to full trajectories $(s_1, a_1, r_1, ..., s_T, a_T, r_T)$ before training the agent. As such, one can easily change the order

in which a model sees this information to $(s_1, r_1, a_1, ..., s_T, r_T, a_T)$ when constructing their training data, and thus condition the generation of a_t given both s_t and r_t. The hypothesis of UDRL and RCP is that if the agent is able to learn a correct mapping from reward to action, it should be able to generalize to higher rewards and perform close to an expert when conditioned on expert rewards, even though it may have only been trained on few experts or only non-expert interactions.

RvS This idea of using non-expert offline trajectories to train RL agents shifts the paradigm of reinforcement learning to a supervised learning problem, which tends to be a better fit for deep neural networks. As a result, a multitude of works was published in the field such as goal-conditioned policies [163], goal-conditioned imitation learning [164], hindsight inference [165], inverse dynamics models [166], decision transformers [3, 7], and trajectory transformers [9]. In an effort to unify all these works under the same paradigm, Emmons et al. [64] coined "Reinforcement Learning via Supervised Learning" (RvS) to encapsulate the idea of shifting the RL problem to a supervised learning one. In this work, the authors probe the limits of RvS methods and they found that if the training data comes from random interactions with the environment, the model will not be able to perform well. Indeed, shifting the RL problem to supervised learning introduces a well-known limitation of supervised learning, which is the (in)capacity of generalizing to out-of-domain data. As a result, if expert reward conditions are not in the training data it can become difficult for the model to understand those at test time.

D2D In a similar effort to unify previous work under the same paradigm to better compare them, Piché et al [2] introduce the Density-to-Decision (D2D) framework. D2D is formulated as a two-step process: (1) density estimation, often implemented as learning some conditional distribution over actions $p(a|s, Z)$ (2) decision-making by sampling with exponential tilt on the density. Using their framework the authors categorize popular RvS methods and propose 'Implicit RvS'. One important aspect present in this work is the concept of exponential tilt. As mentioned previously, supervised learning on non-expert data is likely to result in non-expert behavior. In order to alleviate this issue, one proposed solution is to sample actions that have a high likelihood under the density estimated by the model but that also result in high rewards. This can be done by '*tilting*' the density estimation towards higher valued actions. Concretely, assuming a learned distribution p_θ over actions a conditioned on states s and a measure of quality Z, one can sample actions based on $p_\theta(a|s, Z)\exp(\alpha Z + \beta)$ with α and β being hyper-parameters. This method allows to 'steer' the model towards high return actions even though such actions may not be likely under the training data distribution.

Chapter 6 will use this method and empirically show its effectiveness in the context of text-based environments.

2.7.1 Decision Transformers

This sub-section focuses on recent RvS methods leveraging Transformers as they influenced the third contribution of this thesis.

DTs One popular instance of the RvS paradigm is the work done by Chen et al. [3] proposing Decision Transformers (DTs). In this work, the authors use a decoder-only GPT model [22] to learn to predict actions based on previous interactions and the return-to-go (RTG). Concretely, they give ordered sequences of return-to-go (R_i), state (s_i), and action (a_i) such as $(R_1, s_1, a_1, ..., R_T, s_T, a_T)$ to an auto-regressive model trained to predict the next sequence item as if the sequence items were regular text tokens. However, in this case, the loss is only taking into consideration the prediction of the action items a_i. The return-to-go is defined as the (un-discounted) sum of all future rewards: $R_i = \sum_{t=i}^{T} r_t$. Again, this is possible because the dataset is constructed *after* having collected entire trajectories from another agent. To be able to represent each of the sequence items as one vector, they first learn a linear embedding for each modality: return-to-go, state, and action, which projects raw inputs to an embedding dimension. To encode visual states, they use a convolutional neural network (CNN) instead of a linear layer. In addition, they learn a time-step embedding that is summed to each {RTG, state, action} embeddings to give additional temporal information to the model as one step consists of three embeddings in this case. Evaluated on individual Atari games, DTs performed better than Q-learning and Behavior Cloning. One limitation of this work is that at test time, the initial return-to-go condition must be given by a human according to his/her judgment on the maximum achievable score for each specific game. For instance, in their experiments, the authors used 90 for the game *Breakout* ($\approx 1\times$ max in the dataset), 2500 for the game *Qbert* ($\approx 5\times$ max in the dataset), 20 for *Pong* ($\approx 1\times$ max in the dataset), and 1450 for *Seaquest* ($\approx 5\times$ max in the dataset). These numbers are arbitrary and may result from multiple trial-and-error experiments.

Multi-game DTs This limitation was later addressed by Lee et al. [7] which introduced Multi-game Decision Transformers. In this work, the authors decide to predict the return-to-go instead of manually giving it to the model. Here the authors re-order the trajectory sequences from the training data to be of the form $(s_1, R_1, a_1, r_1, ..., s_T, R_T, a_T, r_T)$ with s_i, R_i, a_i, and r_i being the state, return-to-go, action, and reward at time step i respectively.

As such, given to an auto-regressive Transformer decoder, the model learns to predict RTG based on the current state ($p(R|s)$), next action given state and RTG ($p(a|s, R)$), and observed reward given state, RTG, and action ($p(r|s, R, a)$). However, as mentioned in the previous section, sampling from $p(R|s)$ only according to the model's likelihood will result in the most likely behavior from the training data which may not be optimal. To mitigate this problem, the authors 'tilt' their distribution towards higher return-to-go conditioning values by multiplying $p(R|s)$ by $\exp(\alpha R + \beta)$ with α and β being hyper-parameters (exponential tilt). Another distinction from the original DT work is that here the authors train the same model on multiple Atari games collectively by mixing their trajectories in the training set. As a result, they show that their model can generalize to unseen Atari games and outperform previous baselines.

TTs Around the same time as Decision Transformers was released, Janner et al. [9] introduced a similar approach called Trajectory Transformers (TTs). In this work, the authors also leverage auto-regressive causal transformers and train them on sequences of interactions. However, their setup is different in that they are motivated by learning a realistic model of the environment (world modeling). As such their training data trajectories are of the form $(s_1, a_1, r_1, ..., s_T, a_T, r_T)$ with each state and action items discretized independently in continuous cases (such as in multi degrees of freedom robots): $s_i = [s_{i,1}, ..., s_{i,n}]$ and $a_i = [a_{i,1}, ..., a_{i,m}]$. Trained with the maximum likelihood objective on entire sequences coming from multiple interactions, the model will learn the dynamics of the environment and be able to predict not only the next actions but also the next states and rewards. The authors then demonstrate that imitation learning, goal-conditioned RL, and offline RL are all instances of their model. In the case of imitation learning nothing needs to be changed, the model's likelihood will try to imitate the behavior presented in the training data. Given the left-to-right attention mechanism of causal decoder-only Transformers, one can prepend a goal state at the beginning of the sequence to implement a goal-conditioned RL agent. Eventually, to sample good trajectories, the authors note that the model can also be trained to predict return-to-go values after each reward prediction. At inference time, the strategy used by the authors is to sample multiple trajectories according to the model's likelihood using beam search and select the one with the highest cumulative reward plus reward-to-go. Tested on control tasks where the goal is to generate 'realistic' plans such as in the '*Humanoid walk*', TTs generate more realistic trajectories compared to standard dynamic model parametrizations such as feedforward Gaussian.

To summarize, Section 2.6 introduced interactive text environments and multiple online RL solutions including some leveraging pre-trained language models. Section 2.7 introduced a

new paradigm consisting of reframing reinforcement learning as a supervised learning problem (RvS) and multiple works leveraging Transformers to better learn sequential data. The last contribution of this thesis in Chapter 6 will combine these two ideas and propose an offline RL solution to interactive text environments by leveraging large pre-trained Transformer language models.

2.8 Relevant Work after publications

This section describes some relevant work that was published *after* the first publication of this thesis.

2.8.1 On Prompting-Based Methods

With the ever-growing model size of TLMs, an interesting pattern started to emerge from the literature. LLMs with hundreds of billions of parameters such as GPT-3 175B [50] and PaLM 540B [59] can elicit reasoning abilities in a few shot prompting mechanism called chain-of-thought prompting [167].

Let's first describe what standard prompting is. Popularized by Brown et al. [50], the idea of prompting is that once you have a pre-trained LLM, instead of finetuning it on each desired individual task and saving multiple model weights for each task, one can simply "prompt" the model with a few input/output examples of the task in the input sequence. For instance, one can prompt the model with "*Translate English to French: see otter -> loutre de mer . cheese ->* " to perform machine translation, or with " *Q: what is 98 plus 45? A: 143 Q: what is 17 minus 14? A:*" for mathematical reasoning; and let the model generates the rest, token by token. This method has been surprisingly successful for a range of simple question-answering tasks [50].

When dealing with more complex reasoning tasks, Wei et al. [167] showed that standard prompting is not enough. They thus introduced chain-of-though (CoT) prompting, which corresponds to adding intermediate reasoning steps before the final answer in the prompt. For instance, in the following story: "*Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?*", instead of giving this question and the final answer "*A: The answer is 11.*" to the model, CoT prompting includes "*A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5+6=11. The answer is 11.*". The authors showed that giving examples with step-by-step reasoning instructions to the model as part of its prompt improves the performance of LLMs in mathematic, commonsense, and symbolic reasoning tasks, even surpassing fine-

tuned models in some cases.

Since the introduction of prompting and in particular CoT prompting, multiple works tried to extend these ideas to multi-step reasoning tasks. For instance, Yao et al [168] use LLMs to generate reasoning traces and perform task-specific actions in an interleaved manner in their proposed system called Reason+Act. Creswell et al [169] present a new way (Selection-Inference) to perform multi-step logical reasoning tasks with frozen pre-trained LLMs. In particular, they first select a few (most often two) pieces of information by scoring each sentence in the input context with the model's log-likelihood (Selection step). The retrieved facts are then concatenated and the model generates a conclusion based only on those facts (Inference step). The resulting conclusion is added back to the list of facts in the model's context and the process is repeated until a final answer is generated.

Other prompting methods are also explored recently. For instance, Zhou et al [170] show that algorithmic prompting yields better generalization on arithmetic tasks. Levy et al. [171] show that creating prompts with diverse examples that cover all possible grammatical structures yields better performance in semantic parsing tasks. Overall, these research directions seem to indicate that prompting may become a new 'programming' language for LLMs, which would revolutionize (once again) the way we build NLP systems. Very recent work presented as closing remarks of this thesis in Chapter 8 further confirm this vision.

One limitation of CoT prompting-based method however, is the need to create natural language rationales explaining each logical step of a problem. This can become a limiting factor when dealing with datasets that do not have such explanations or that require multiple skills. Some early work was proposed by Zelikman et al. [172] to mitigate this limitation. Their method consists of using pre-trained LLMs to generate rationales and use those that yielded correct responses as fine-tuning examples to improve the model's ability to generate better rationales. This self-loop is applied over multiple iterations and the authors show that the resulting model performs comparably to a fine-tuned 30× larger model on CommonsnseQA [173].

Another limitation of prompting-based methods with Transformers is that TLMs have a fixed input window size. As a result, the more information one puts in a prompt, the less space is available to input the original problem of interest. This can become a limiting factor when complex problems need long chains of reasoning or when multiple examples are needed for the model to correctly understand what is required. In fact, Fu et al. [174] demonstrated that models prompted with longer and more complex chain-of-thought sequences perform significantly better than models prompted with standard CoT sequences. Long context size and efficient Transformers were an active research direction before the introduction of

prompting, and it has become even more relevant recently. Some interesting work in these directions include Longformer [175] and LongT5 [58].

2.8.2 On Proof Generation

Another active field of research since the releases of large pre-trained language models such as RoBERTa [51] and GPT-2 [24] is the domain of proof generation. Some related work in this field are described below.

PROVER Shortly after the first contribution presented in this thesis, Saha et al [176] propose a system called PROVER that relies on a pre-trained RoBERTa model to predict yes/no answers and generate proof graphs for the QA dataset RuleTaker [1]. Their model is fine-tuned on examples containing a series of facts and logic rules (the context) and a question in the form of an unknown fact. Since RoBERTa is an encoder only model, they rely on the embedding of the "[CLS]" token to predict a binary label (true/false) for the unknown fact. The authors additionally predict which fact (node in their proof-graph construction) from the context are part of the proof and which edges are present between each predicted graph node. Their experiments show that while predicting proof-graphs yields stronger results, performance drops as the depth of reasoning increases, similar to our findings.

GPT-f Polu and Sutskever [177] trained a decoder-only Transformer model (GPT-f) with 774 million parameters on proof trajectories from the Metamath library. They processed 38,000 theorems with their respective proofs, linearize them in sequences of the form "`GOAL <GOAL> PROOFSTEP <PROOFSTEP> <EOT>`", and trained their model with the regular conditional language modeling objective. At runtime, the authors test their model by conditioning it on a specific *GOAL* and let the model generates the remaining of the sequence (the proof steps). Their experiments show that GPT-f can generate new short proofs that were accepted into the Metamath library. This thesis is more interested in natural language explanations rather than formal mathematical proofs, but nevertheless this work shows the promises of LLMs in more formal setups.

LAMBADA Previous work such as CoT prompting [167] and Selection Inference [169] search for proofs in the forward direction from known facts to the conclusion. These methods can result in combinatorial explosion of the search space, resulting in failure cases on problems requiring long chains of reasoning. To address this limitation, Kazemi et al. [178] propose LAMBADA, a system that perform backward-chaining proof resolution: going from the goal

to classify as true or false, to the set of known axioms. In particular their method leverages the power of prompting with the PaLM 540B model to define four modules: *Fact Check*, *Rule Selection*, *Goal Decomposition*, and *Sign Agreement*. Each of those module is "just" a carefully designed prompt that shows the model some input/output examples to tell it what is expected. Remarkably, the authors show that their method, without any fine-tuning, outperforms previous methods on the ProofWriter dataset [37]. These results confirm the impressive effectiveness of pompting-based method on very large language models as discussed in the previous sub-section.

A flurry of other works was published in the recent months (late 2022) on using LLMs for proof generation as it is an important research direction as motivated in Chapter 1. Yang et al. [179] propose the NLProofS method that searches natural language proofs by generating a proof tree and fine-tuning a verifier module (based on RoBERTa [51]) that checks the validity of each proof step. Welleck et al. [180] generate mathematical proofs with a GPT-3 based model [50] by conditioning on background references such as theorems and definitions, and by adding some constraints to the decoding procedure. Lample et al. [181] propose HyperTree Proof Search (HTPS) which combines an encoder-decoder Transformer model of 600 million parameters with Monte Carlo Tree Search to prove theorems from the Metamath and Lean environment. They pre-train their model on 40GB of latex code from the mathematical section of arXiv before finetuning it on their different environments. To deal with more complex environments such as mathematical competition problems, Jiang et al. [182] introduce a system called Draft, Sketch, and Prove (DSP) that relies on both LLMs and automated theorem provers. They first prompt the Codex model [183] (LLM pre-trained on code data) to generate informal proofs, and give those to guide an automated theorem prover. Overall, all these works demonstrate the enthusiasm around the use of LLMs for proof generation as the field is rapidly evolving.

CHAPTER 3 ORGANIZATION OF THE THESIS

In order to answer the research question introduced in Chapter 1: "*are Transformer language models able to reason logically across multiple steps and what is required for them to be better at it?*", this thesis is composed of three contributions presented in chronological order in Chapter 4, 5, and 6. Each of these chapters addresses the three research objectives defined in Chapter 1. We describe below how the three chapters fit together.

Chapter 4 investigates the systematic generalization ability of TLMs on a logical reasoning task expressed in natural language. Inspired by how humans reason by thinking step by step, the first contribution studies if training a model to generate intermediate reasoning steps before generating a final answer is beneficial in the context of relation prediction. The dataset used for this first analysis is a collection of synthetic knowledge graphs expressed in natural language that can be of arbitrary complexities. The goal of the task is to predict which relation links two entities in the underlying graph. The advantage of using this dataset is that we can carefully control the amount of compositional generalization required to solve the task, and we can use the graph of each problem to construct multiple reasoning paths (proofs) by traversing the graph. Multiple proof strategies are explored and compared against each other based on the generalization capacity of the model in both interpolation and extrapolation settings.

Chapter 5 continues to research methods that improve the reasoning capacity of TLMs, but to gain a wider perspective, it extends the first contribution by considering a broader range of problems since not all question-answering datasets have proofs associated with them. Inspired by how humans reason with variables rather than grounded entities, the second contribution explores if entity-type abstraction is beneficial to pre-trained encoder-decoder TLMs in the context of multi-step question answering. This contribution uses the same compositional relational reasoning dataset as the first contribution but also considers three other QA datasets ranging in the type of reasoning required (abductive reasoning, multi-hop question answering, conversational question answering) and their vocabulary sizes. After leveraging an automatic named entity tagger, several model architecture designs are proposed to incorporate entity type abstraction into a pre-trained encoder-decoder TLM. Each method is evaluated and compared based on the model's performance on each QA dataset.

Chapter 6 extends the previous two contributions by exploring the reasoning capacity of TLMs in an even broader scope: interactive text environments. These human-generated natural language environments require multi-step reasoning by design, as the task is completed

after multiple interactions with the environment. As a good proxy for such environments, text-based games are used in this last contribution. These games are originally designed by humans for human players to have fun while struggling to solve them. The goal is to obtain the maximum score by interacting with the environment through text commands multiple times until a final state is reached. Although the player's text commands can be synthetic templates, the text responses from the environment describing the state the player is in are human-generated text and can be very long and varied, not following any template. Each game can have sub-quests in them requiring an agent to decompose a complex task into simpler sub-tasks, solve each of them, and compose their solutions to eventually solve the original problem, which is exactly how multi-step reasoning is defined in Chapter 1. Previous work in the literature envisioned online Reinforcement Learning methods to tackle these environments, some leveraging pre-trained language models, but none in an offline RL setting. On the other hand, previous work in offline RL leverage Transformers to learn from sequences of interactions, but none taking advantage of pre-trained TLMs or in the context of text environments. The third contribution of this thesis bridges this gap by proposing an offline method (Long-context Language Decision Transformers) that leverages pre-trained TLMs in interactive text environments.

CHAPTER 4 ARTICLE 1: MEASURING SYSTEMATIC GENERALIZATION IN NEURAL PROOF GENERATION WITH TRANSFORMERS

Authors Nicolas Gontier, Koustuv Sinha, Siva Reddy, Christopher J. Pal

Published at 34th Conference on Neural Information Processing Systems (NeurIPS 2020) on December 8th 2020.

Abstract We are interested in understanding how well Transformer language models (TLMs) can perform reasoning tasks when trained on knowledge encoded in the form of natural language. We investigate their systematic generalization abilities on a logical reasoning task in natural language, which involves reasoning over relationships between entities grounded in first-order logical proofs. Specifically, we perform soft theorem-proving by leveraging TLMs to generate natural language proofs. We test the generated proofs for logical consistency, along with the accuracy of the final inference. We observe length-generalization issues when evaluated on longer-than-trained sequences. However, we observe TLMs improve their generalization performance after being exposed to longer, exhaustive proofs. In addition, we discover that TLMs are able to generalize better using backward-chaining proofs compared to their forward-chaining counterparts, while they find it easier to generate forward chaining proofs. We observe that models that are not trained to generate proofs are better at generalizing to problems based on longer proofs. This suggests that Transformers have efficient internal reasoning strategies that are harder to interpret. These results highlight the systematic generalization behavior of TLMs in the context of logical reasoning, and we believe this work motivates deeper inspection of their underlying reasoning strategies.

4.1 Introduction

Systematic Generalization has been characterized as the capacity to understand and produce a potentially infinite number of novel combinations from known components [33, 34]. For example, in Figure 4.1, a model could be exposed to a set of facts (e.g., "*Nat is the granddaughter of Betty*", "*Greg is the brother of Nat*", "*Flo is the sister of Greg*"), but not to all the possible facts that can be inferred by combination of the known components (e.g., "*Flo is the granddaughter of Betty*"). More recent work has examined systematic generalization in terms of the ability of "a model to manipulate concepts in new combinations after being

trained on all concepts, but only on a limited set of their combinations" [35]. This view of systematic generalization shifts emphasis from reasoning to learning. Here we examine systematic generalization through measuring the ability of a model to reason about new inference step combinations despite being trained on a limited subset of them.

Recent developments in natural language processing (NLP) have shown that Transformer [10] Language Models (TLMs) are able to capture linguistic knowledge [184–186], and yield state-of-the-art performance for many NLP tasks [17, 22], including but not limited to answering reading comprehension questions [24, 50] and generating factual knowledge [62] with little to no task supervision. These models are optimized on large corpora to predict the next words or a set of masked words in a sentence. While yielding impressive results, it is not clear if TLMs rely on many superficial patterns in the data or if they actually learn re-usable skills, enabling them to generalize to new tasks by leveraging the compositionality of those skills [27, 67]. Training on massive data can give certain advantages with respect to understanding the meanings of words, but we conjecture that such data gives models less experience with reasoning over inference chains.

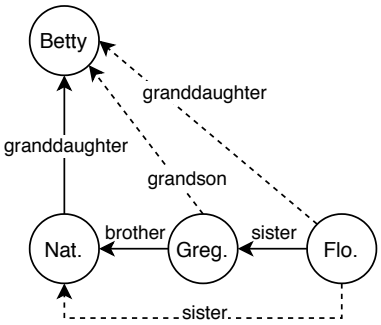


Figure 4.1 Example of a CLUTRR graph with known facts (solid lines) and unknown facts to infer (dotted lines).

In our work, we study the less understood issues related to how well TLMs are able to perform long chains of reasoning. In particular, we use TLMs for the task of theorem proving, where facts and proofs are specified in natural language. Using theorem proving, we test if TLMs can generate interpretable proofs with logically consistent language modeling as their main objective. In particular, we study their *behavior* as logical reasoners on text by analyzing the generated proofs and the final answer. This setup allows us to evaluate the reasoning and generalization capabilities of TLMs. Recent work such as Petroni et al. [62]; Raffel et al. [52]; Brown et al. [50] suggest that language models can be treated as knowledge bases. This directly motivates us to investigate if language models can also learn certain reasoning strategies. Studying these abilities can give us insights to facilitate the use of such models as dynamic knowledge bases that could infer new knowledge even if it is not seen during pre-training.

For natural language theorem proving, we use the question answering CLUTRR benchmark suite [36] to perform controlled studies. This dataset is of interest because: (i) the compositional nature of tasks involved make it well suited for evaluating systematic generalization,

and (ii) each question–answer pair is accompanied by a proof that can be used to explain how to arrive at the answer. We use this dataset as a medium to understand the reasoning capacity of TLMs.

Our experiments reveal the following:

1. TLMs suffer from length generalization: they cannot *extrapolate* to proofs requiring more proof steps than seen during training time.
2. They generalize better when trained to generate long proofs compared to short proofs.
3. They generalize better when trained to generate backward-chaining proofs rather than forward-chaining.
4. Surprisingly, they generalize better when they are trained to directly generate the answer instead of learning to generate the proof and then the answer.

To the best of our knowledge, we are the first to use a language modeling objective to do interpretable theorem proving with a Transformer. We hope that this work can shed some light on the reasoning capacity of TLMs and inspire future research to design models with greater reasoning capacity.

4.2 Related Work

Systematic generalization has recently been in spotlight due to its importance in understanding the strengths and weaknesses of neural networks. Bahdanau et al. [35, 187] identify and evaluate the generalization capacity of visual question answering models. We however focus this study on a fully natural language domain. Dasgupta et al. [28] introduce a natural language inference (NLI) dataset which proves to be challenging for language understanding models for compositional generalization. Goodwin et al. [29] also evaluate systematic generalization in an NLI setting with controlled test cases to observe the failures of neural architectures. We however focus this study on the systematic generation of logical reasoning sentences by Transformer-based [10] language models in a question answering setting with the CLUTRR suite [36]. Similar datasets include SCAN [67] which has been instrumental to test systematic generalization [30, 31] and CFQ [72] which measures the systematicity of language understanding via a question answering setup. Sinha et al. [36] propose a series of baseline models with the CLUTRR dataset but none of them took advantage of the provided proof attached with each example. In addition, their Transformer baselines were not

fine-tuned on the task. Unlike them, we focus on learning and generating proofs for studying systematic generalization.

Neural proof generation [188] and neural theorem proving [89, 92, 94] have been explored in previous work. They tend to combine symbolic and statistical approaches to leverage the compositionality and interpretability of symbolic systems and the flexibility of statistical systems. Nevertheless, these combined systems all assume some predefined set of atoms and rules making up the environment. We instead use natural language text to define our environment and measure the limits of a purely statistical approach.

Similarly to us, Clark et al. [1] leverage logical rules expressed in natural language to answer compositional questions. However their system is not generative, rather they predict a true/false binary label on candidate answers. We instead focus on the systematic generalization capacity of generating proofs and using them to generate the final answer.

4.3 Evaluating systematic generalization through interpretable reasoning

4.3.1 The task

Background. We use the family relation CLUTRR benchmark suite [36] to generate our dataset[1]. Each example is composed of: (i) a family graph $G = (V, E)$ (referred as *story*) with entities as nodes ($v \in V$) and relationships as edges ($e \in E$), (ii) a *query* about the relationship between two entities (v_1, _, v_n) separated by more than one hop in the family graph (iii) a reasoning path (referred as *proof*) expressed as a list of (v_i, e_j, v_k) tuples, referred to as *facts* and (iv) the target relationship e^* between the two queried entities (referred to as the *answer*). The dataset contains 272 distinct entities and 20 relationship types, ordering to $\sim$ 1.5M possible facts. Each (v_i, e_j, v_k) fact can be expressed in natural language using either one of 5 factual sentences (referred to as `facts` template), or by using one of $6,000$ noisy but more natural sentences written by mechanical turkers (refered as `amt` template). Family graphs are expressed using either the `facts` template or the `amt` template, while queries, proofs and answers are always expressed with the `facts` template. A CLUTRR example can be seen in Table 4.1 and Figure 4.1.

Terminology. In order to evaluate systematic generalization, we define the following building blocks that constitute a *proof*:

- `entity`: one node (e.g., *"Anna"*).
- `relation`: one edge (e.g., *"mother"*).

[1]Dataset and code can be downloaded at https://github.com/NicolasAG/SGinPG

	raw	facts	amt
story	[(Natasha, granddaughter, Betty), (Florence, sister, Gregorio), (Gregorio, brother, Natasha)]	<STORY> Natasha is a granddaughter to Betty. Florence is Gregorio 's sister. Gregorio is a brother of Natasha.	<STORY> Betty likes picking berries with her son 's daughter. Her name is Natasha. Gregorio took his sister, Florence, to a baseball game. Gregorio and his sister Natasha love it when their grandmother visits because she spoils them. She is coming this week to watch them while their parents are out of town.
query	(Florence, _, Betty)	<QUERY> Who is Florence for Betty ?	
proof	[{(Florence, granddaughter, Betty): [(Florence, sister, Gregorio), (Gregorio, grandson, Betty)]}, {(Gregorio, grandson, Betty): [(Gregorio, brother, Natasha), (Natasha, granddaughter, Betty)]}]	<PROOF> since Florence is a sister of Gregorio, and Gregorio is a grandson to Betty, then Florence is a granddaughter to Betty. since Gregorio is a brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is a grandson of Betty.	
answer	granddaughter	<ANSWER> Florence is the granddaughter of Betty	

Table 4.1 CLUTRR example of level 3 (ie: 4 entities, 3 relations, 2 proof steps). The proof follows the `short-proof-rev` strategy. We refer the reader to Figure 4.1 to visualize the corresponding graph in which solid lines refer to the facts given in the story and dotted lines refer to the new facts inferred in each proof step.

- `fact`: one factual sentence representing a (v_i, e_j, v_k) tuple using `facts` template (e.g., *"Anna is the mother of Bob"*).
- `proof_step`: one inference step combining two `facts` to get a new one (e.g., *"since Anna is the mother of Bob and Bob is the brother of Carl then Anna is the mother of Carl."*).
- `proof`: the entire *resolution chain*, consisting of multiple ordered `proof_step`s.

Following the setup of CLUTRR, we define the relative *difficulty* of individual examples according to the number of edges present in the family graph. For instance, Table 4.1 and Figure 4.1 show a level-3 example because there are 3 solid edges (known facts) between 4 entities. In general, a **level** k **task consists of** k **edges** (corresponding to k sentences in the story) **between** $k+1$ **nodes and** $k-1$ **hidden edges to infer** (corresponding to $k-1$ proof steps to solve the task). As the levels increase, so does the number of sentences in the story and the number of proof steps in the proof.

Problem Setup. We trigger a model to: (1) given a story and query, generate a proof followed by an answer, and (2) given a story, query, and a proof, generate an answer. In particular, we train a Transformer-based decoder [189] with the language modeling objective on entire sequences of "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER> [answer]":

$$L(\theta) = \sum_i \log P(w_i | w_1, \ldots, w_{i-1}; \theta)$$

This setup enables to generate both the answer to a query and the proof to arrive at this

answer, given as input the family graph story and a question. Concretely, we inject sequences of the story and query having delimiters "<STORY>" and "<QUERY>" to the language model and trigger it to generate the corresponding proof and answer with tokens "<PROOF>" and "<ANSWER>" respectively.

4.3.2 Proof resolution strategies

sp	since Gregorio is a brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is a grandson of Betty. since Florence is a sister of Gregorio, and Gregorio is a grandson to Betty, then Florence is a granddaughter to Betty.
spr	since Florence is a sister of Gregorio, and Gregorio is a grandson to Betty, then Florence is a granddaughter to Betty. since Gregorio is a brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is a grandson of Betty.
lp	since Gregorio is the brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is the grandson of Betty. since Florence is the sister of Gregorio, and Gregorio is the brother of Natasha, then Florence is the sister of Natasha. since Florence is the sister of Natasha, and Natasha is the granddaughter of Betty, then Florence is the granddaughter of Betty.
lpr	since Florence is the sister of Natasha, and Natasha is the granddaughter of Betty, then Florence is the granddaughter of Betty. since Florence is the sister of Gregorio, and Gregorio is the brother of Natasha, then Florence is the sister of Natasha. since Gregorio is the brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is the grandson of Betty.

Table 4.2 Proof resolution types for an example of level 3. We refer the reader to Figure 4.1 for the kinship graph corresponding to this example. **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed.

In our task, we turn language models into approximate proof generators. Specifically, we train TLMs to generate `proofs` (as defined in Section 4.3.1). We do not explicitly perform inference on the generated `proofs`, but reformulate the language generation objective to generate the inferred answer after the `proof` sequence. This allows to leverage TLMs to generate *forward* and *backward* chaining resolution paths used in Inductive Logic Programming (ILP) [190]. In our case, these resolution paths are expressed in natural language. To simulate approximate theorem generation, we introduce four different types of `proof` that can be used to derive the answer given a story and query. An example of each type can be seen in Table 4.2 and we describe them below:

short-proof-rev (spr). This setup is the backward-chaining resolution path provided by the CLUTRR dataset, which is generated by recursive application of the kinship logical rules. This proof strategy can be viewed as an *explain-why* scenario, where the first sentence in the proof contains the answer (target relationship) and the subsequent sentences contain the intermediate steps required to explain that answer. We refer the reader to Sinha et al. [36] for further details on the generation of this proof setting.

short-proof (sp). Here we reverse the resolution chain provided by the CLUTRR dataset by swapping all sentences from the `short-proof-rev` setup. Doing so, we arrive at a forward-chaining inference path, in which the final proof step consists of the target relationship. Specifically, the first sentence in the proof combines two facts from the given story to infer a new fact. In subsequent proof steps, the inferred fact from the previous step is combined with a fact from the story, to infer a new fact until the answer is found.

long-proof (lp). Forward-chaining inference in ILP consists of generating all possible new facts from the starting facts, and evaluate each of them for the resolution of the target answer [91]. Similarly, in this setup, we extend the `short-proof` setup where we attempt to infer all possible facts given the ones present in the input story. Each proof step combines any two previously known facts to infer a new fact until the answer is found. Pseudo-code for generating this type of proof can be found in Appendix A.

long-proof-rev (lpr). This setting is the same as the previous one, but in reverse. It starts from the answer and goes back to the facts originally given in the story. This resolution strategy can be viewed as a backward-chaining strategy where all possible paths are considered. This proof strategy is obtained by swapping all sentences from the `long-proof` setting.

We compare each strategy in our experiments to understand which form of logical resolution is easier to learn for TLMs. In particular, we note that the reversed proof strategies (`spr` and `lpr`) fall in the backward-chaining family of logical resolution, while the non-reversed strategies (`sp` and `lp`) represent the forward-chaining resolutions. Backward-chaining family features the proof step containing the answer at the beginning of the proof. On the other hand, forward-chaining type proofs (`sp` and `lp`) feature the proof step containing the answer at the end of the proof.

4.3.3 Systematic generalization in proof generation

Now that we have defined the task and various proof generation strategies available in our setup, we proceed to define the aspects of generalization we aim to test. Our initial CLUTRR formulation tested the generalization capacity of a model to new `facts` hence new `proof_steps` and new `proofs`, after being trained on all `entities` and `relations`. Initial

ANON TEST	lvl.**2**	lvl.**3**	lvl.**4**	lvl.**5**	lvl.**6**	lvl.**7**	lvl.**8**	lvl.**9**	lvl.**10**
proofs (*many proof steps*)	16.28%	0%	0%	0%	0%	0%	0%	0%	0%
proof steps ("*since A-r1-B and B-r2-C then A-r3-C*")	73.08%	58.06%	52.75%	54.28%	50.93%	59.04%	56.92%	53.55%	52.17%
facts (*A-r-B*)	100%	100%	100%	100%	100%	100%	100%	100%	100%
entities (*A*)	100%	100%	100%	100%	100%	100%	100%	100%	100%
relations (*r*)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4.3 Percentage of the test proof's building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value of 100% to better visualize which building blocks were entirely contained in the training set.

experiments on this setup showed that TLMs fail to generalize to unseen `facts`. Due to the presence of a large number of entities in CLUTRR, we ended up with a combinatorially large number of possible facts. The model may thus not be able to learn how to represent each entity effectively, hence reducing its chances to learn higher-order structures such as unseen `facts`. Experimental results on this original setting are provided in Appendix 4.A.

We instead slightly simplify the generalization evaluation and allow the model to also be exposed to all possible `facts`. This formulation tests a model capacity to generalize to new `proof_steps` hence new `proofs`, after being trained on all `entities`, `relations` and `facts`. Since providing a training corpus covering all possible facts would significantly increase the training data, we instead reduce the number of entities by replacing them by one of k[2] randomly sampled entity tokens, resulting in significantly fewer possible facts, and thus all facts being contained in the training set (Table 4.3).

Interpolation and Extrapolation. Having access to the level of difficulty of each test examples, we evaluate both how Transformers can generalize to *unseen* proofs of the *same* difficulty as seen during training (inductive generalization); and how they can generalize to *unseen* proofs of *unseen* difficulty levels. In particular, we test *interpolation* in which the testing difficulty levels are less than training levels; and *extrapolation* in which the test difficulty levels are higher than training levels. This systematically tests the length generalization capabilities of TLMs in logical theorem proving.

[2] $k = 20$ in our case because we know that the maximum number of entities in a story is less than 20.

4.4 Experiments and Analysis

We aim to answer the following questions to analyze the proof generation capabilities of Transformer-based language models (TLMs):

1. Are TLMs able to reason better after being trained to generate interpretable proofs expressed in natural language?
2. Which types of proof are easier to learn and to generate for TLMs?
3. Which types of proof are more useful for TLMs to generate accurate answers?

Setup. In all our experiments we used a Transformer decoder architecture [189] with 2.5M and 3.5M parameters with a vocabulary size of 90 and $1,800$ tokens for stories expressed with the `facts` and `amt` template respectively. Detailed parameter settings for our models are given in Appendix 4.B. We also ran preliminary experiments with a larger model (145M parameters) (Appendix 4.C), with a GPT2 model [24] (Appendix 4.D), and with a more complex network (an encoder-decoder transformer) (Appendix 4.E) but found similar conclusions or further investigation being required. We generate $390,000$ CLUTRR examples of level 2 to 10. We train the models on $300,000$ examples of levels $2, 4$ and 6 and evaluate the model on a test set of $10,000$ examples for all levels from 2 to 10. Specifically, we test levels 3 and 5 for interpolation; levels $2, 4$ and 6 for inductive generalization; and levels $7, 8, 9$ and 10 for extrapolation.

Evaluation Metrics. In the following experiments, we evaluate both the generated *proof factual consistency* (that we denote '*validity*' in the rest of this document) and *answer accuracy*. The answer is defined as the first sentence after the "<ANSWER>" tag in the generated sequence. Since all answers during training were expressed using the `facts` template, we inverse this template to extract the $(entity, relation, entity)$ triple from the generated answer. If the extraction fails, we consider the generated answer wrong. We then compare the extracted triple to the ground truth provided in the CLUTRR dataset. For comparison, in all experiments, we also report the accuracy of the naive most-frequent-relation (MFR) baseline consisting of predicting the relation that is the most frequent in the training set for the queried entity pair.

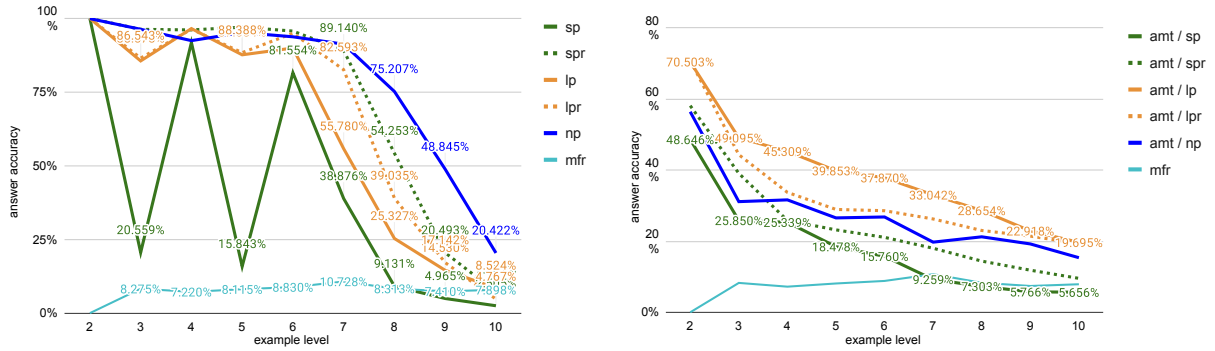
A proof is defined as the ordered sequence of all sentences generated between the "<PROOF>" and "<ANSWER>" tokens. For validating a proof, since all proofs during training were expressed using the `facts` template, we inverse this template to extract all $(entity, relation, entity)$ triples from the generated proof sentences. If the extraction process fails at any point, the

entire proof is considered invalid. The ordered sequence of each proof step is then evaluated against the transitivity rules defined by the CLUTRR environment. In addition, we also check that all the facts necessary for the proof are either given in the input story, or inferred from a previous proof step. If any of these conditions fail, we consider the proof invalid, otherwise we consider the proof 'valid' (ie: factually consistent).

No proof setup. In addition to the four proof strategies defined in Section 4.3.2, we also compare in all our experiments with a model that is trained to directly generate the answer after the story and query. In particular, this *no-proof* model is trained on sequences of "<STORY> [story] <QUERY> [query] <PROOF> none . <ANSWER> [answer]". This allows us to estimate how important is the proof for our models to be able to generalize.

4.4.1 Answer Accuracy

We evaluate the answer accuracy of models trained with different proof settings on the test set described earlier by Table 4.3. Each model is given as input a story, query and the proof trigger token ("<STORY> [story] <QUERY> [query] <PROOF>"), and we let them decode the next tokens, that is, the proof followed by the answer.



(a) stories expressed with the `facts` template. (b) stories expressed with the `amt` template.

Figure 4.2 Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

Q: *Are TLMs able to generalize to unseen proof steps?* **A**: *For simple language, yes in interpolation and no in extrapolation. For complex language, no in both cases.*

In Figure 4.2a we evaluate models trained wit stories expressed with the `facts` template. We

observe that in all proof setups, with the exception of short-proofs, TLMs are able to systematically generalize to predict the correct answer inferred from unseen `proof_steps` and `proofs`, both in inductive (levels 2, 4, 6) and interpolation levels (levels 3 and 5). However, in all proof setups TLMs have difficulties to extrapolate to longer problems requiring a larger number of reasoning steps, conforming to length generalization issues discovered in related tasks [30].

In Figure 4.2b we note that models trained on noisy `amt` stories fail to systematically generalize to predict the correct answer. In addition, we can see a linear decrease in accuracy with the level of difficulty. Having to de-noise the input stories to extract relevant kinship relations, in addition to running logical inference, makes the task much more challenging for our network. We conjecture that generalizing in this harder setting may require additional capacity added to the model, either in terms of model size, model architecture, training data, or a combination of all the above. For instance, we explore the benefit of fine-tuning GPT2 [24] in Section 4.D as an initial step, but leave room for further improvement in future work.

Q: *Which reasoning strategy generalizes better?* **A**: *Backward-chaining is better than forward-chaining, but no-proof can be better than both. Long-proofs are better than short-proofs.*

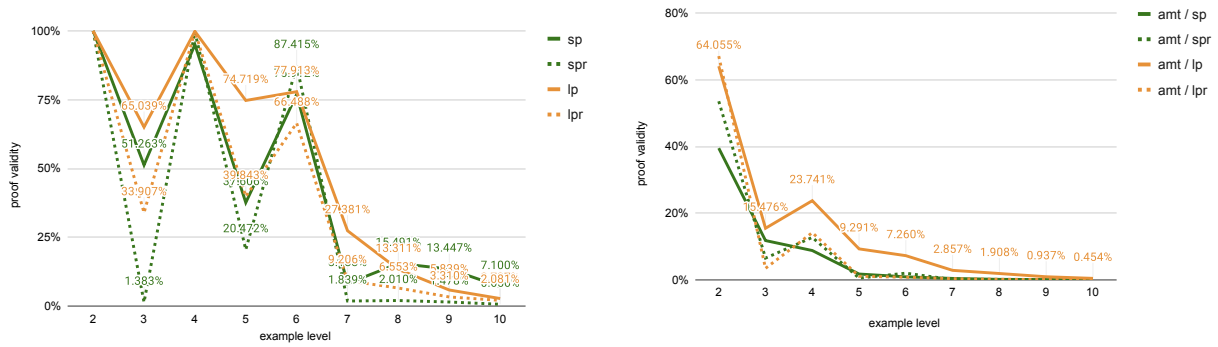
We observe that backward proof strategies (spr, lpr) better help the model to answer accurately than their respective forward strategies (sp, lp) (Figure 4.2), with the exception of long proofs in the `amt` story template. This suggests that backward chaining is easier to learn, easier to use, or both, than forward chaining for TLMs. We believe this effect is due to the position-dependent exploitation of TLMs. Indeed, the answer is in the first generated proof-step in case of backward-chaining proofs. In addition, we note in Figure 4.2 that long-proofs (lp, lpr) yield better generalization performance than short-proofs (sp, spr) with the exception of reversed strategies in the `facts` story template.

It is also interesting to see that models trained to go directly to the answer by generating the "*none*" token as a proof tend to perform better than all other models required to generate the proof in `facts` stories (Figure 4.2a). One hypothesis is that the generated proof may be invalid most of the time and hence the extra information given by the proof is actually deteriorating the model's performance. To see if that may be the case, we next look at the validity of the generated proofs for all models (except the trivial no-proof).

4.4.2 Proof Validity

We evaluate the proof validity of models trained with different proof settings on the test set (previously described by Table 4.3) in Figure 4.3. Similarly as above, each model is given as

input a story and query and we trigger the model to decode the proof and answer with the trigger tokens "<PROOF>" and "<ANSWER>" respectively.



(a) stories expressed with the `facts` template. (b) stories expressed with the `amt` template.

Figure 4.3 Proof validity for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof.

Q: *Which reasoning strategy is easier to generate?* **A**: *forward-chaining is easier than backward-chaining and long-proofs are easier than short-proofs.*

From Figure 4.3a we observe that forward-chaining strategies (sp, lp) tend to be easier to generate than their respective reversed strategies (spr, lpr). This is contrary to the previous observation where backward-chaining strategies were easier for the models to understand. We believe that this is due to the fact that the model has a higher chance of generating the first proof step correctly than the final proof step. Since backward chaining proofs contain the answer in the first proof step, when re-using that information to predict the answer, there is a higher chance that the answer will be correct. This explains why the answer accuracy of such model is relatively high while their proof validity is relatively low.

In addition, we observe that in both `facts` and `amt` stories (Figure 4.3), long proof strategies are easier to generate than shorter ones. This was not expected at first since long sequences are usually harder to model in language models. One hypothesis is that since long-proofs come from a systematic construction (see Appendix A) they are easier to generate than the more arbitrary short proofs.

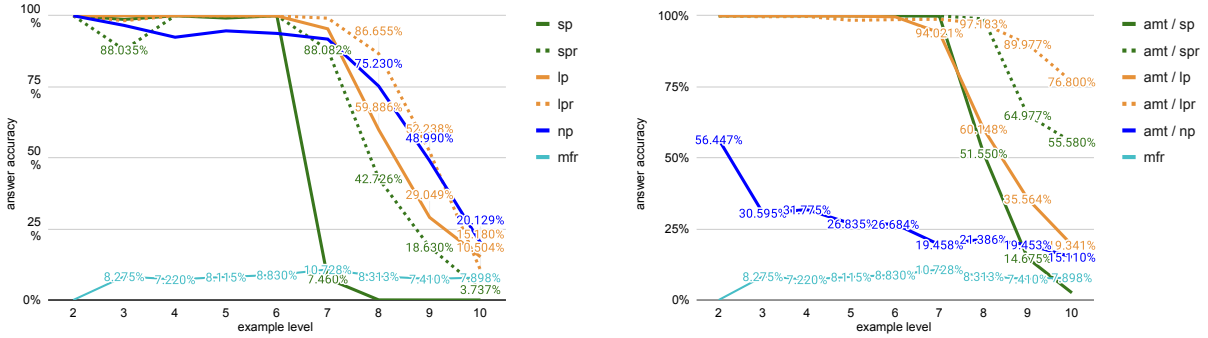
Q: *Are TLMs able to generate valid proofs of unseen lengths?* **A**: *No.*

We observe that valid proofs are difficult to generate for TLMs in unseen difficulty levels, both in interpolation and extrapolation setting (Figure 4.3a). This partially explains why

the no-proof setting in the previous section yielded better generalization performances. In addition, we note in Figure4.3b that the generated proofs from models trained on noisy `amt` stories are mostly invalid. We believe that this is due to the fact that models need to de-noise the information from the input story in addition to generating a valid proof, making the task much harder. To understand if models rely on the validity of the proof, we next evaluate their answer accuracy when given the real proof as input rather than the generated one.

4.4.3 Proof is given

To understand if models rely on the proof, we again evaluate the answer accuracy as in Section 4.4.1, but this time the models are given as input the story, the query and the real proof followed by the answer trigger token: "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>". We then let the language model decode the next tokens making up the answer. Note that the no-proof model is given "none" as its "[proof]" so we don't expect this model performance to change from Section 4.4.1.



(a) stories expressed with the `facts` template.

(b) stories expressed with the `amt` template.

Figure 4.4 Answer accuracy for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>" and they generate the answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

Q: *Are ground-truth proofs useful for TLMs to generalize systematically?* **A**: *Yes.*

When the proof is provided in the input, all models outperform the no-proof model in inductive and interpolation test cases (Figure 4.4). In extrapolation test cases, models trained on `facts` stories (Figure 4.4a) benefit from the proof compared to Section 4.4.1, and models trained with `amt` stories outperform the no-proof model (Figure 4.4b). This suggests that

models do learn to use the correct proof to better generalize during inference. However, as the difficulty of the examples increases, the generalization performance of all models decreases. Even when given the proof containing the correct answer, TLMs fail to copy the correct information from sequences of greater length than seen during training. Our hypothesis for this is that Transformers strongly rely on the position of the answer and have trouble learning simple tasks – such as copying the answer from the proof – if the information for this task happens at unseen positions.

Q: *Which reasoning strategy is easier to use when generating answers?* **A**: *backward-chaining is easier to use than forward-chaining and long-proofs are easier to use than short-proofs.*

Another interesting observation is that, in general, the reversed proofs (dotted lines in Figure 4.4) tend to be more useful than forward strategies for our model in generating the correct answer, aligning with our findings in Section 4.4.1. Similarly as above, we believe that this is due to the facts that Transformers strongly rely on the position of the answer. Indeed, in reversed proofs (**spr**, **lpr**), the answer is always in the first proof step, for which the position depends only on the story length; whereas in **sp** and **lp** the answer is always in the last proof step, for which the position depends both on the story length and on the proof length.
We also see that long, exhaustive proofs are easier to be used when generating the final answer, compared to short-proof strategies. This suggests that while being a longer sequence of tokens to encode, if a model was able to generate such proofs, it would ease its generalization capacities.

4.5 Conclusion

TLMs are state of the art models for a wide variety of natural language processing tasks. Given their widespread use, it is important to understand the limits of their ability to reason on knowledge expressed in natural language and to extrapolate learned inference procedures to unseen problem instances. Our explorations reveal multiple insights. Firstly, TLMs suffer from length-generalization issues in generating proofs. Secondly, TLMs get better at reasoning when trained with longer, exhaustive proofs. In addition, the fact that backward-chaining proof models perform better than forward-chaining ones makes us believe that backward-chaining strategies are easier to use albeit being harder to generate. Moreover, we find that no-proof models perform better than those trained to produce proofs. We conjecture that benefiting from naturally stated logical proof statements requires more complex internal representations. Recent work on developing position-agnostic attention mechanisms for Transformers [191] can be useful as a future direction to develop generalizable models. Furthermore, our results motivates the use of neuro-symbolic methods such as Neural Theorem

Provers [89] as an alternative avenue to achieving systems that systematically generalize on logical and compositional reasoning tasks. Combining these approaches with large pre-trained language models is left as future research. We hope that this work will inspire research on the systematic generalization capacity of language models and motivate further study and the creation of neural models with greater reasoning capacity.

Broader Impact

Transformer based models have been very effective for various language understanding and generation tasks. Due to their recent successes, there is significant interest in the applications of these models to real world scenarios such as: Dialogue, Question Answering and text-classification. However, failure of such systems could produce nonsensical, wrong or racially-biased results [192]. Therefore, a logical analysis of their limitations and issues in generalization to unseen data, such as in this work, could have a positive impact on building safer, more robust and interpretable systems in these domains.

In this work, we rely on systematic tests to trigger Transformer-based models to generate an interpretable proof in natural language, and then evaluate the robustness properties of that proof. Using a first-order logic based dataset, we explicitly test the logical consistency of such proof. This research can shed some light into developing more robust and systematic models in the future. In addition, it can help us understand the reasoning strategies employed by Transformer-based models for both inference and generation. However, the fact that proof-free inference works so well, may also imply that models which generate proofs, do so in a decoupled way from the computations yielding the final answer. This negative result could give users a false sense of explainability.

Acknowledgments and Disclosure of Funding

The authors would like to acknowledge support from Element AI for providing computational resources which were used to run the experiments in this work. We also acknowledge the help provided by Sandeep Subramanian in sharing part of his experimental code. Nicolas is partially funded by a scholarship from the Fonds de Recherche Quebec Nature et Technologie. We thank CIFAR for their support through the CIFAR AI Chairs program. We also thank NSERC and PROMPT for their support.

4.A Original CLUTRR evaluation

ORIGINAL TEST	lvl.**2**	lvl.**3**	lvl.**4**	lvl.**5**	lvl.**6**	lvl.**7**	lvl.**8**	lvl.**9**	lvl.**10**
proofs (*many proof steps*)	99.62%	0%	0%	0%	0%	0%	0%	0%	0%
proof steps ("*since A-r1-B and B-r2-C then A-r3-C*")	99.62%	0%	99.96%	0%	100%	0%	0%	0%	0%
facts (*A-r-B*)	100%	0.47%	100%	0.83%	100%	0.20%	0.20%	0.10%	0.42%
entities (*A*)	100%	23.81%	100%	35.72%	100%	26.19%	21.43%	30.95%	30.95%
relations (*r*)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4.4 Percentage of the **original test** proof building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value close to 100% to better visualize which building blocks were entirely contained in the training set.

The original CLUTRR data generation framework made sure that each test `proof` is not in the training set in order to test whether a model is able to generalize to unseen proofs. Initial results on the original CLUTRR test sets resulted in strong model performance ($\sim$ 99%) on levels seen during training (2, 4, 6) but no generalization at all ($\sim$ 0%) to other levels. After further analysis, we noticed that due to the cloze style nature of CLUTRR tasks, the first names representing entities were chosen arbitrarily. This resulted in level-k test set's `proof_steps` and `facts` also being in the level-k training set. In addition, level-k test set's `entities` were mostly seen *only* in level-k training set. This resulted in a big overlap between training and test sets for examples of the same level, but a weak overlap on other levels as we can see in Table 4.4.

NAMED TEST	lvl.**2**	lvl.**3**	lvl.**4**	lvl.**5**	lvl.**6**	lvl.**7**	lvl.**8**	lvl.**9**	lvl.**10**
proofs (*many proof steps*)	2.13%	0%	0%	0%	0%	0%	0%	0%	0%
proof steps ("*since A-r1-B and B-r2-C then A-r3-C*")	2.13%	0%	1.33%	1.74%	1.42%	1.80%	1.38%	0.99%	1.40%
facts (*A-r-B*)	15.48%	5.52%	6.77%	10.92%	6.38%	9.63%	10.51%	10.33%	8.33%
entities (*A*)	100%	100%	100%	100%	100%	100%	100%	100%	100%
relations (*r*)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4.5 Percentage of the **Named test** proof's building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value of 100% to better visualize which building blocks were entirely contained in the training set

In our case, the entity names are important to evaluate systematic generalization. We want to evaluate the capacity of a model to generalize to new `facts`, `proof_steps`, and `proofs`, but keeping the `entities` and `relations` the same. We thus modified the original CLUTRR dataset to select test entities according to entities present in the training set. We devise a test set that uses all `relations` and `entities` from the training set but new `facts`, `proof_steps` and `proofs` for all levels. We call this dataset the *Named* data: all entities are referred by their original first name. Train and test overlap percentages between all building blocks are in Table 4.5.

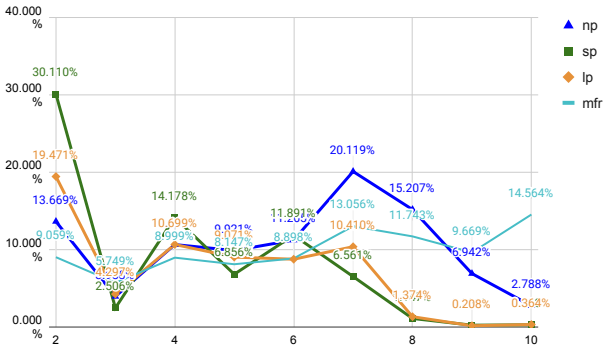
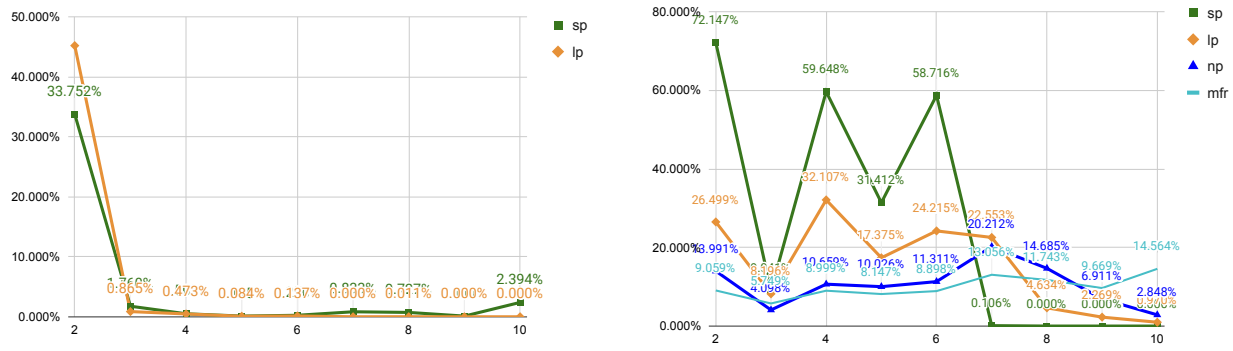


Figure 4.5 Answer accuracy on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and asked to generate the proof and answer. Models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp** = short-proof, **lp** = long-proof, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

Given as input the story and the query followed by the proof trigger token ("<STORY> [story] <QUERY> [query] <PROOF>") the model generated the corresponding proof and answer. We report in Figure 4.5 the answer accuracy and in Figure 4.6a the proof validity of all our models. Similarly, in Figure 4.6b we report the answer accuracy of our models when they are given as input the story, the query and the real proof, followed by the answer trigger token ("<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>").

Experiments on this setup show that Transformer language models fail to generalize to unseen `facts`. Indeed, due to the presence of a large number of entities in CLUTRR, we end up with combinatorially large number of possible facts. The model may thus not be able to learn how to represent each entity effectively, hence reducing its chances to learn higher-order structures such as unseen `facts`.



(a) Proof validity on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and asked to generate the proof and answer.

(b) Answer accuracy on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>" and asked to generate the answer.

Figure 4.6 Evaluation of models trained on levels 2, 4, 6 only.

4.B Experiments parameter settings

	small	**large**
patience	20	20
batch size	512	256
float precision	16	16
embedding dimension	192	768
number of layers	5	20
dropout	0.1	0.1
transformer mlp hidden size	768	3072
attention heads	3	12
max length	$1,024$	512
activation	gelu	gelu
number of warmup steps	$20,000$	$20,000$
optimizer	adam	adam
total parameters	$\sim 3,000,000$	$\sim 145,000,000$

Table 4.6 Parameter settings.

All experiments in the main section of the paper were run with the **small** model size.

Additional experiments in Section 4.C were run with the **large** model size.

4.C More parameters

In this section we report the answer accuracy of a model trained with ∼145M parameters and compare its generalization performance with our initial smaller network (∼2.5M parameters). Models are trained on levels 2, 4 and 6. Each model is given the story and query as input, and triggered to generate the proof and answer with the "<PROOF>" and "<ANSWER>" tokens respectively.

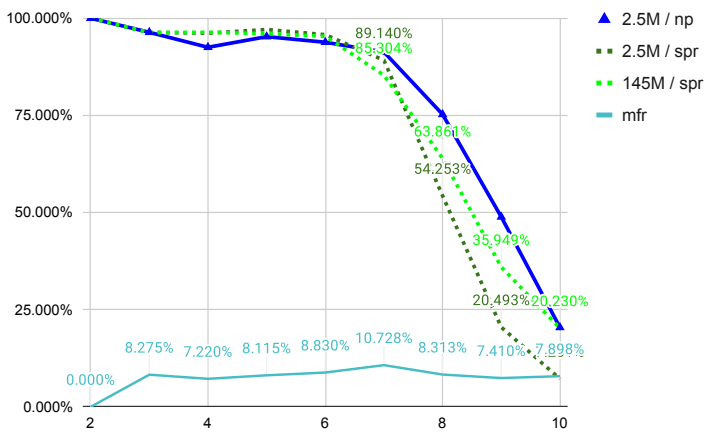


Figure 4.7 Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. Models are trained on levels 2, 4, 6 only. Stories are expressed with the `facts` template. Different proof settings are evaluated: **np** = no-proof and **spr** = short-proof-reversed. We also report the naive most-frequent-relation (**mfr**) baseline. Results on other proof settings with the 2.5M parameter network can be found in Figure 4.2a.

We observe in Figure 4.7 that the generalization capacity of the larger 145M network is almost identical to the smaller 2.5M parameter network trained on the same data (`facts` stories and short-proof-reversed). In addition, we also observe that the 145M model trained on reversed short proofs (145M / spr) is not better than the 2.5M model trained without any proof (2.5M / np). Overall, results show that model size improves only marginally the generalization capacity in our task.

4.D Fine-tuning GPT2

In this section we report the answer accuracy of GPT2 models [24] trained from-scratch (`gpt2FS-`) on the CLUTRR dataset and of pre-trained GPT2 models fine-tuned (`gpt2FT-`) on the CLUTRR dataset. We leverage the GPT2 implementation from the huggingface library [193]. The resulting models have ∼125M parameters. In all experiments the models are trained on stories expressed in the `amt` template. Models are fine-tuned on levels 2, 4 and 6. Each model is given the story and query as input, and triggered to generate the proof and answer with the "<PROOF>" and "<ANSWER>" tokens respectively.

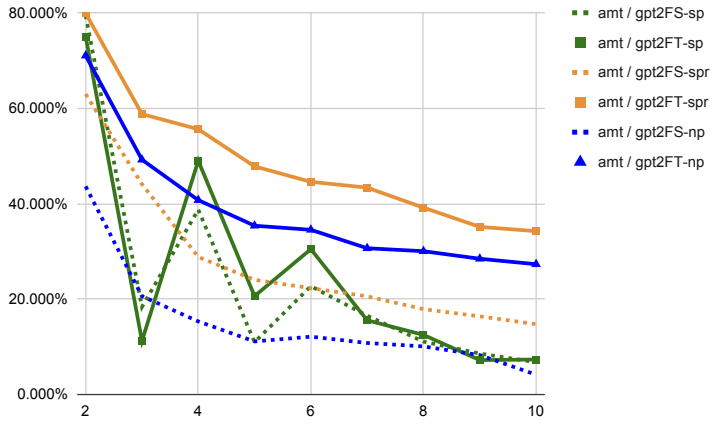


Figure 4.8 Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. Models are fine-tuned on levels 2, 4, 6 only. Stories are expressed with the `amt` template. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **np** = no-proof. We compare the performance of models trained from scratch (dotted lines; `gtp2FS-`) and of fine-tuned models (solid lines; `gpt2FT-`).

In Figure 4.8 we observe that in general, fine-tuned models perform better than the ones trained from scratch. We can also see that reversed-proof strategies are better than their forward proof counterpart, which is in accordance with what we discussed in Section 4.4.1. Although fine-tuning seems to improve the generalization capacity of GPT2, it is also interesting to note that the benefit of fine-tuning GPT2 on short-proofs (sp) is negligible compared to the benefits of fine-tuning GPT2 on short-proofs-reversed (spr) or no-proof (np). This suggests that fine-tuning alone is not enough to yield strong generalization performance, but the choice of proof strategy also influences greatly the answer accuracy.

4.E Encoder-Decoder Network

In this section we evaluate the answer accuracy of sequence-to-sequence models trained on `facts` templated stories of level 2, 4 and 6. These models consist of a 5-layer Transformer encoder and a 5-layer Transformer decoder, each of them following the same parameter settings than what is described in the '**small**' column of Table 4.6. This resulted in 5.22M parameter models. Sequence-to-sequence models are trained to encode the story and question with the encoder, and generate the proof and answer with the decoder. Models trained on levels 2, 4 and 6. Each model is given the story and query as input, and triggered to generate the proof and answer with the "<PROOF>" and "<ANSWER>" tokens respectively.

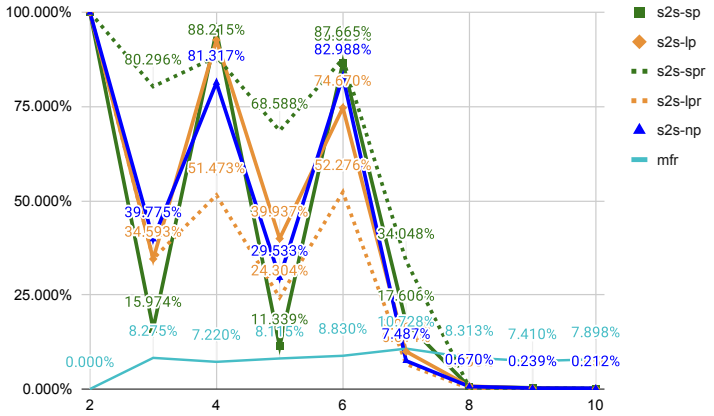


Figure 4.9 Answer accuracy for all test levels from 2 to 10. The models encodes the input "<STORY> [story] <QUERY> [query]" and they decode the proof and answer. Models are trained on levels 2, 4, 6 only. Stories are expressed with the `facts` template. Different proof settings are evaluated: **sp** = short-proof, **spr** = short-proof-reversed, **lp** = long-proof, **lpr** = long-proof-reversed, **np** = no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

In the results shown in Figure 4.9, we see that sequence-to-sequence models do not generalize well to unseen difficulty levels, both in extrapolation settings (levels 7–10) but also in interpolation settings (levels 3 and 5). This suggests that encoder-decoder architectures are more sensible to the sequence length seen during training. On the other hand, it is important to note that the encoder network was trained with the auto-regressive language modeling objective back-propagated from the decoder. It would be interesting to see if pre-training the encoder with a more traditional objective, that is masked language modeling [17], would improve the generalization performance. We leave this exercise as future work. In addition, we plan to explore pre-trained models such as T5 [52] in future work in order to improve performance with this type of architecture.

CHAPTER 5 ARTICLE 2: DOES ENTITY ABSTRACTION HELP GENERATIVE TRANSFORMERS REASON?

Authors Nicolas Gontier, Siva Reddy, Christopher J. Pal

Published in Transactions on Machine Learning Research (TMLR) on November 20th 2022.

Abstract We study the utility of incorporating entity type abstractions into pre-trained Transformers and test these methods on four NLP tasks requiring different forms of logical reasoning: (1) compositional language understanding with text-based relational reasoning (CLUTRR), (2) abductive reasoning (ProofWriter), (3) multi-hop question answering (HotpotQA), and (4) conversational question answering (CoQA). We propose and empirically explore three ways to add such abstraction: (i) as additional input embeddings, (ii) as a separate sequence to encode, and (iii) as an auxiliary prediction task for the model. Overall, our analysis demonstrates that models with abstract entity knowledge performs better than without it. The best abstraction aware models achieved an overall accuracy of 88.8% and 91.8% compared to the baseline model achieving 62.9% and 89.8% on CLUTRR and ProofWriter respectively. However, for HotpotQA and CoQA, we find that F1 scores improve by only 0.5% on average. Our results suggest that the benefit of explicit abstraction is significant in formally defined logical reasoning settings requiring many reasoning hops, but point to the notion that it is less beneficial for NLP tasks having less formal logical structure.

5.1 Introduction

Transformer language models (TLMs) [10] have enabled rapid progress in natural language processing (NLP). When pre-trained on large corpora (such as the web) to predict the next tokens or a set of masked tokens from an input sequence, TLMs can capture linguistic knowledge [184–186], and yield state-of-the-art performance on many NLP tasks with little to no task supervision [17, 22, 24, 50]. However, it is not clear if these models can capture higher level knowledge such as reasoning skills that can be re-used in arbitrary contexts, and in ways that leverage the compositionality of those skills [27, 67], something logical reasoners can do relatively well on a smaller scale [194, 195]. Simple compositional tasks such as SCAN [67], CLUTRR [36], and ProofWritter [1, 37] can help diagnose the compositional generalization behavior of language models. Recent work on some of these datasets showed that TLMs

still struggle to learn reasoning strategies that can be re-used in out-of training distribution settings [67, 196].

Humans do abstraction to simplify reasoning and become very efficient. This is particularly true in mathematics when manipulating variables instead of numbers. Manipulating abstract concepts allows humans to generalize knowledge across domains. If we look at how logical reasoners operate, we find that they have an important abstraction component (going from grounded entities to higher level concepts) before logical reasoning can start [194]. Going from an original text sequence to its higher-order meaning is an important part of the NLP pipeline (part of it being entity type tagging). Similarly in mathematics, the introduction of generic variables allows to progress in a logical reasoning process without keeping track of every (grounded) atomic entity. Overall, this idea that we call abstraction, seems to be an important part of logical reasoning. Recent work suggests that incorporating external knowledge about grounded entities could improve language models' abilities to reason and generalize [106, 129, 130, 138]. However the empirical effect of incorporating generic entity types remains unclear, especially with recent studies suggesting that pre-trained models already encode some of that linguistic knowledge in their parameters [60, 61, 197, 198]. In this work, we study the effect of explicitly providing entity type abstraction *in addition to* the original input to pre-trained Transformers.

We explore and evaluate different ways to incorporate entity type abstraction and observe that some methods are more effective than others. To construct the abstract representation incorporated into TLMs, we leverage entity type information given by fixed pre-trained models. This allows for automatic and reproducible data processing. In general, our approach is the following: given an input sequence, we use an entity tagger to label entity types in the sequence. We then use these labels to construct a copy of the original sequence in which all entities are replaced by their corresponding entity types. This new sequence can then be used as extra input (Sections 5.3.1 & 5.3.2) or as extra training signal to the model (Sections 5.3.3). In particular, we explore three different ways to augment pre-trained Transformers with this abstract knowledge:

1. by combining token embeddings from both the original and the abstract sequence before encoding (Section 5.3.1) (Figure 5.1a & 5.1b).

2. by encoding both the original and the abstract sequence and combining them before decoding the target output (Section 5.3.2) (Figure 5.1c & 5.1d).

3. by adding a second language model head on top of the Transformer decoder to predict the abstract sequence (Section 5.3.3) (Figure 5.1e).

A series of controlled experiments on two synthetic datasets show that models having access to abstract knowledge about entity types yield better performance at inference time both when interpolating and extrapolating to unseen lengths of reasoning chains. Since no (non-synthetic) natural language dataset systematically and explicitly requires long chains of reasoning, we used synthetic datasets in which we can control for the degree of compositional generalization required. Nevertheless, in order to understand if the benefits observed could also be applicable in more realistic settings, we ran a series of experiments on two question answering datasets requiring some degrees of multi-hop reasoning. Results on these more natural language datasets show that abstraction aware models are not significantly better than baseline models. We conclude that these "real-world" problems do not have strong enough logical structure to benefit from the abstraction technique and that the pre-trained weights of large language models seem to be "enough" for such natural language tasks, confirming previous results [60, 185]. It is only when tasked on logical problems that explicitly require reasoning depths unseen during training that abstraction becomes significantly beneficial.

Overall our work contributions are the following:

1. we introduce and compare empirically different ways to incorporate abstraction into pre-trained TLMs.
2. we show that incorporating abstract knowledge can significantly improve compositional generalization to unseen lengths of reasoning chains in multi-step reasoning tasks.
3. we show that abstraction aware models may not benefit much when language is more natural and less procedural.

We hope that our work will inspire future research in the field to look for simple inductive biases that can complement pre-trained models in their quest to achieve logical reasoning at scale.

5.2 Related Work

Augmenting neural language models with knowledge about entities has been a popular method to improve their functionality. Ji et al. [129] trained an entity neural language model to predict sequences of entities with an LSTM [19]. At each sampling step, they predict the next word alongside a categorical variable indicating the current token's entity ID. They obtained lower perplexity and better results on co-reference resolution and entity prediction tasks than a variety of baselines. Similarly, Rosset et al. [130] trained a GPT2 model [24] by giving it access to entity knowledge at the input level and as an additional pre-training

loss. Their model achieved better factual correctness on benchmarks such as LAMA [62], and performed better than a baseline GPT2 model in various question answering tasks. Inspired by this work and motivated by the goal of building better reasoning language models, we instead focus on the prediction of entity *types* rather than entity *identifiers* taken from a fixed list of entities. This allows our solution to be robust to new entities. In addition, we explore and compare different ways to incorporate the entity knowledge in an encoder-decoder architecture.

Besides entity knowledge, other types of explicit information has also been given to language models. Prior work by Swayamdipta et al. [132], Eriguchi et al. [133], Nădejde et al. [134] tried incorporating syntax information into language models by introducing an auxiliary loss to the model. Results show that models trained to also predict syntactic information achieved stronger performances on various tasks such as PropBank semantics and Neural Machine Translation. Inspired by this work, we also introduce an auxiliary loss but to predict entity types, and with an application towards reasoning tasks. Also working with syntax information, Sundararaman et al. [136] incorporated POS tags into the input embedding of a BERT model. Results show improved BLEU score on machine translation and higher accuracy than baselines on the GLUE benchmark [109]. Similarly, Sachan et al. [199] augmented a pre-trained BERT model with a syntax graph neural network in order to encode syntax trees. Their results show that the quality of the trees are highly tied to the performance boost observed. Levine et al. [56] trained a BERT-like model to learn word senses. They gave their model access to WordNet supersenses at the input level and as an additional training loss. They achieve better performance than other baselines on the SemEval Word Sense Disambiguation task [137]. Moosavi et al. [138] propose to improve robustness to data biases by augmenting the training data with predicate-argument structures. They train a BERT-base model [17] with PropBank-style semantic role labeling [139] on MultiNLI [140] and SWAG [141] datasets. Their results show that incorporating predicate-argument structure in the input sequence (only during training) makes the model more robust to adversarial examples in MultiNLI. More recently, Porada et al. [142] extended a RoBERTa model [51] with hypernym abstraction based on WordNet to evaluate the plausibility of events. Their model is able to better predict human plausibility judgement than other RoBERTa baselines. **Although different in application, all these prior works leverage the general idea of explicitly giving more abstract knowledge to language models, hence showing how flexible and generic this strategy can be.** We take a similar approach with entity types, but in the hope of improving the reasoning skills of our baseline model.

A flurry of recent work has also examined ways to augment TLMs with entities from external knowledge bases [54, 106, 110, 114]. However, most of the time, these solutions rely on

external components such as knowledge graphs with pre-trained entity embeddings, and/or an additional memory. While they often use entity linking as a way to perform co-reference resolution, they do not incorporate higher level of abstractions such as entity types like we do here.

5.3 Introducing Abstraction Inductive Biases

In this section we describe five different ways to incorporate abstraction into a pre-trained encoder-decoder model. Given an input sequence X, we use `spacy` named entity tagger[1] to make a simplified copy X_s of the input. This is a more generic copy of X.

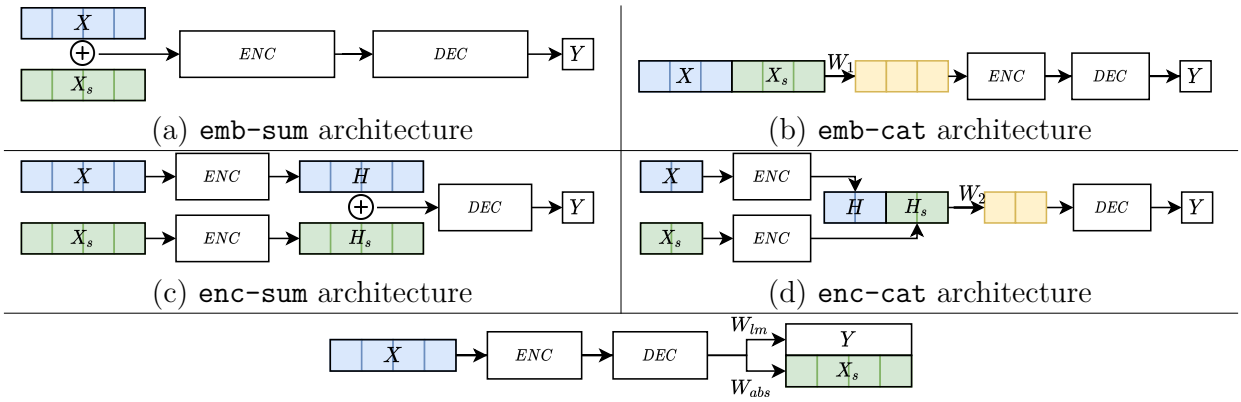
We run the `spacy` recognizer on X to extract entity tags such as *PERSON*, *ORG*, *GPE*, etc... For each entity type, we create n additional vocabulary entries (with randomly initialized embeddings) such as [`PERSON_1`, ..., `PERSON_n`, `ORG_1`, ..., `ORG_n`, ...]. Every token in X is then replaced by their (randomly numbered) entity tag to make the simplified sequence X_s. If the same entity is present multiple times in X, each occurrence will be replaced by the same entity tag in X_s. If no entity is found for a token in X, the original token's text is kept in X_s (*e.g.* "*Bob Smith has a cat that he loves. Bob also loves Alexandra.*" would be transformed into "*PERSON_11 has a cat that he loves. PERSON_11 also loves PERSON_3.*").

We select n greater than one as we believe that for some tasks it is important to differentiate between two entities of the same type. At the same time we select n to be much smaller than the total number of entities in the dataset, thus forcing abstraction. In particular, the hyper-parameter n is set to the smallest possible value such that each distinct entity within the *same* example gets a different entity tag. This is different for each dataset depending on the number of unique entities per example. Individual values can be seen in Appendix 5.A. If the same entity appears more than once in a single example, it will get the same tag every time within that example. Since we re-use the same finite set of entity tags across all examples, each entity tag will be used for different entity tokens, thus after seeing many examples, entity tags of the same type will likely have a similar embedding. We discuss some result highlighting this phenomenon in Appendix 5.B.

In order to better understand the influence of having multiple tags for the same entity type, we also ran experiments in which we set $n = 1$, thus forcing all entities of the same type to be mapped to the same embedding. However we noticed both more variance and weaker performance of our models, so the rest of this work will focus on the $n > 1$ setting described above. Results can be seen in Appendix 5.C.

[1] https://spacy.io/models/en

In the following subsections, we describe different strategies to incorporate X_s into an encoder-decoder Transformer model.



(a) `emb-sum` architecture (b) `emb-cat` architecture

(c) `enc-sum` architecture (d) `enc-cat` architecture

(e) `dec-loss` architecture

Table 5.1 Different architectures for different abstraction strategies. X (blue) is the original sequence embedding, X_s (green) is the embedding of the simplified sequence with entities replaced by their entity type tags, "*ENC*" is the T5 encoder, H (blue) is the contextualized representation of sequence X, H_s (green) is the contextualized representation of sequence X_s, "*DEC*" is the T5 decoder, and Y is the target sequence to predict.

5.3.1 Abstraction as an additional embedding

Our first strategy is to combine X_s with X at the embedding level. To do that, we construct X_s to be of the same length of X. If a `spacy` entity spans over multiple tokens (*e.g.* ["*Alex*", "*andra*", "*Smith*" "*is*", "*the*", "*wife*", "*of*", "*Bob*"]), we copy its entity tag at each sub-token positions (*e.g.* ["*PERSON_3*", "*PERSON_3*", "*PERSON_3*", "*is*", "*the*", "*wife*", "*of*", "*PERSON_11*"]). For each token within both sequences we either sum (`emb-sum` experiments) or concatenate (`emb-cat` experiments) their respective token embeddings.

sum. In `emb-sum` experiments (Figure 5.1a), if tokens do not have entity tags associated with them, we ignore their embedding to avoid summing the same embedding twice for non-entity tokens. We only sum embeddings of tokens that do have an abstract tag associated with them. This is to ensure that we only modify pre-trained embeddings that correspond to entity tokens. This is done by masking out the tokens in X_s that are the same in X. The input given to the model's encoder is then $emb(X) + mask \times emb(X_s) + positional$ with $emb()$ being the embedding matrix, $mask$ defined as the $X \neq X_s$ binary tensor, and $positional$ being the regular Transformer positional embedding. This resembles the setting used by Rosset et al. [130], however their knowledge-aware embedding comes from a sequence of entities from a dictionary lookup, rather than a sequence of entity *types* from a pre-trained

tagger. The advantage of our method is that it is robust to unseen entities of the same type.

concat. In `emb-cat` experiments (Figure 5.1b), if tokens do not have an entity tag associated with them, we replace their embedding with a generic (learnable) "<*grounded*>" token embedding. This ensures that all non-entity token embeddings gets modified in the same way compared to entity tokens. We eventually resize the concatenated embeddings with a learnable matrix $W_1 \in R^{2e \times e}$ with e being the model's embedding size. The input given to the model's encoder is then $[emb(X); emb(X_s)] \cdot W_1 + positional$ with $emb()$ being the embedding matrix, $[;]$ defined as the concatenation operator, and *positional* being the regular Transformer positional embedding. In this setup the model has $2e^2 + e$ more parameters.

5.3.2 Abstraction as an additional sequence to encode

Our second strategy is to combine X_s with X at the encoding level. To do that, we again construct X_s to be of the same length of X. Similarly as above, if a `spacy` entity spans over multiple tokens, we copy its entity tag at each sub-token positions. We then encode both X and X_s with the same encoder weights to have two contextualized encodings: H and H_s respectively. For each token within both sequences we either sum (`enc-sum` experiments) or concatenate (`enc-cat` experiments) their respective contextualized encodings.

sum. For `enc-sum` experiments (Figure 5.1c), the input given to the model's decoder becomes $H + H_s$. Unlike in Section 5.3.1, in these experiments we do not mask any position because the encodings are contextualized over the entire sequence. All token representations were influenced by all other tokens because of the Transformer encoder attention mask. Thus even non-entity token representations were influenced by entity tokens.

concat. For `enc-cat` experiments (Figure 5.1d), we introduce a learnable matrix $W_2 \in R^{2d \times d}$ with d being the model's encoding size in order to resize the concatenated encodings, similarly to Section 5.3.1. The input given to the model's decoder is then $[H; H_s] \cdot W_2$ with $[;]$ defined as the concatenation operator. In this setup the model has $2d^2$ more parameters.

5.3.3 Abstraction as an auxiliary task

Our third strategy is to incorporate X_s into the model with an additional cross entropy loss (`dec-loss` experiments). The model will now be tasked to predict both the target output Y as well as the abstracted input X_s. To do that, we introduce a second language model head $W_{abs} \in R^{d \times vocab}$ (with d being the model's decoder size) on top of the model's decoder, initialized to have the same weights than the original language model head W_{lm}, and fine-tuned during training with a second cross-entropy loss. The final model's loss is then the

average between the two cross entropy losses:

$$0.5 * \frac{1}{|X_s|} \sum_{i=0}^{|X_s|} P(X_s)_i * log(\text{softmax}(H^{dec} \cdot W_{abs}))_i$$
$$+0.5 * \frac{1}{|Y|} \sum_{j=0}^{|Y|} P(Y)_j * log(\text{softmax}(H^{dec} \cdot W_{lm}))_j,$$

with H^{dec} being the output tensor of the model's decoder. Note that X_s and Y have different lengths ($|X_s|$ and $|Y|$ respectively), which is why we differentiate indices between the two sums. By sharing the decoder weights (except for the additional language model head), we make sure that most of the network parameters are influenced by the additional cross-entropy loss. This acts as a regularizer and forces the model to "*know*" about entity types within its original parameters. In this setup, the model has $d * vocab$ more parameters. Figure 5.1e illustrates this strategy. While not in the scope of our initial experiments, we evaluated the performance of this model to predict abstract sequences with its additional language model head. Curious readers can refer to Appendix 5.B for more details.

5.4 Experiments

In this section we describe the experiments we ran on various datasets. We start with the CLUTRR benchmark [36] in Section 5.4.1 as controlled experiments in which we know how much compositional generalisation is required. We then test our models on the ProofWriter [1, 37] dataset in Section 5.4.2. This allows to verify if entity type abstraction is beneficial in formally defined logical reasoning environments with simple language. We next report experiments on the multi-hop question answering task HotpotQA [38] in Section 5.4.3. This allows to test if entity type abstraction is beneficial in two-hop question answering settings with more natural language which is by nature more noisy. Eventually, we also report results on the conversational question answering task CoQA [39] in Section 5.4.4. This allows to test if entity type abstraction is beneficial in conversational settings in which the entity being discussed may be originally introduced much earlier in the conversation, thus requiring some entity linking before answering.

For all experiments we trained one model for each of the 5 different strategies presented in Section 5.3, in addition to a baseline model fine-tuned without any abstraction knowledge. We used the `AllenNLP` library [200] with the HuggingFace `transformers` library [193] PyTorch implementation of `T5-small` with 16-bit floating point precision. Each experiment was run on tesla V100 32gb GPUs with early stopping and a patience of 10 epochs on the validation

set (defined as a 10% split of the training set). We report all hyper-parameters and library versions in Appendix 5.A for reproducibility purposes.

5.4.1 Compositional generalization with CLUTRR

Although synthetic, CLUTRR is used because it allows for controlled experiments in which we can clearly measure both interpolation and extrapolation performance of our model. We generated $390,000$ examples that were roughly split 77/23 between training and testing. Each example consists of a unique (non-cyclic) family graph. The goal of this task is to infer the type of edge (family relation) between two nodes (two entities) that are the further apart in the input graph. The bigger the graph, the more hops are required to infer the missing edge. We express each family graph, along with its question and answer in text using a simple "*[e_1] is the [rel] of [e_2]*" template. Some examples of input/output sequence pairs can be seen in Appendix B.

We evaluate the generated answer accuracy. The answer is defined as the first sentence in the generated sequence. Since all answers during training were expressed using a simple template, we inverse this template to extract the $(\hat{e}1, \hat{rel}, \hat{e}2)$ triple from the generated answer. If the extraction fails, we consider the generated answer wrong. We then compare the extracted triple to the ground truth provided by the CLUTRR dataset. If the reference solution is $(e1, rel, e2)$, we accept both $(e1, rel, e2)$ and $(e2, inv_rel, e1)$ as valid solutions, with inv_rel being the inverse relation of rel. For instance, the inverse relation of "*father*" can be "*son*" or "*daughter*", we accept both.

Testing Compositional generalization

We carefully divided train and test sets to force the model to generalize to both unseen graph sizes (*i.e.*: unseen reasoning depth) and unseen $(e1, rel, e2)$ triples. Specifically, the training set is made of graphs with 2, 4 and 6 relations between 3, 5 and 7 entities respectively, while the test set is made of graphs with up to 10 relations between 11 entities. In addition, all possible entities and relations are seen during training but only 9.58% of $(e1, rel, e2)$ triples from the test set are also in the training set. This small overlap makes the test set harder than originally designed and allows to analyse the compositional generalization capacity of our model.

We fine-tuned a `T5-small` model on $300,000$ training examples of levels $2, 4$ and 6 and evaluate the model on 9 test sets of $10,000$ examples each for all levels from 2 to 10. The level is defined as the number of edges (relations) in the graph. The higher the level is,

the bigger the graph is, the further apart the two entities to link are, and the greater the number of hops required to answer the query is. Specifically, we test levels $2, 4$ and 6 for compositional generalization, levels 3 and 5 for interpolation, and levels $7, 8, 9$ and 10 for extrapolation.

Results

CLUTRR	2	3	4	5	6	7	8	9	10	avg.
no abstraction	100%	84.4%	64.8%	61.5%	54.2%	56.6%	45.5%	42.6%	56.8%	**62.9%**(±0.18)
emb-sum	100%	85.3%	74.6%	59.3%	64.3%	67.7%	65.1%	58.9%	68.4%	**71.5%**(±0.13)
emb-cat	94.6%	60.0%	35.6%	42.0%	40.2%	74.3%	78.2%	77.9%	80.6	**64.8%**(±0.21)
enc-sum	**100%**	**94.8%**	**86.9%**	**89.9%**	**85.6%**	**87.2%**	**85.1%**	**84.3%**	**85.5%**	**88.8%**(±0.05)
enc-cat	100%	86.1%	72.6%	70.2%	66.7%	69.1%	63.8%	61.9%	74.0%	**73.8%**(±0.12)
dec-loss	100%	74.7%	64.9%	59.3%	56.2%	61.3%	52.8%	47.8%	61.3%	**64.2%**(±0.15)

Table 5.2 Prediction accuracy on CLUTRR test set for all difficulty levels. Models have been trained on levels 2, 4, 6 with only 9.58% of all the $(e1, rel, e2)$ triples present in the test set. Per-level performance is colored in shades of green for better visualisation. The red boxed area indicates test problems at depths unseen during training.

Table 5.2 shows answer accuracy on each test set level for models trained with $n = 20$ tokens per entity type. As mentioned in Section 5.3, this helps keeping track of "who's who" in the abstracted sequence, which is important for solving a task such as CLUTRR. We see that the best model (`enc-sum`) strongly outperforms all other models with an average score of 88.8% compared to 62.9% for the baseline.

sum -vs- concatenation. We can see that for both the embedding strategies (`emb-cat` & `emb-sum`) and the encoding strategies (`enc-cat` & `enc-sum`), summing representations together yields better performance than feeding their concatenation through a linear layer. One hypothesis is that in the `emb-cat` model, all pre-trained embeddings are modified by either a entity type token embedding or the "<grounded>" embedding, whereas the `emb-sum` model keeps most of its pre-trained embeddings unchanged. This result also suggests that sometimes, simpler methods are more effective.

embedding -vs- encoding. From Table 5.2 we also see that, on average, processing the abstract sequence through the encoder yields better performance than only processing it through the token embedder. This is expected as more layers of the Transformer can process the abstracted sequence.

learning the abstraction. In the last line of Table 5.2 the abstraction is given as output to the model. This experiment tests if learning how to abstract along-side the original task helps the model. We can see that learning to predict the abstract sequence can indeed

help generalize as the average score increases from 62.9% for the baseline to 64.2% for the `dec-loss` model. However it is not the most effective method in this case. We believe this is because we test longer reasoning lengths (up to 10 steps), making the input sequence much longer than what is seen during training (up to 6 steps). This is because in CLUTRR, to make long reasoning chains the input family story gets longer. This penalizes the `dec-loss` model since it is designed to generate the abstracted version of the input, regardless of its length.

Overall, we can see that abstraction-aware models can all extrapolate better than the baseline model, suggesting that entity abstraction does help pre-trained Transformers to compositionally generalize. To verify that this is not just a feature of the CLUTRR dataset, we perform the same analysis on another synthetic dataset in the next section.

5.4.2 Abductive Reasoning with ProofWriter

Similarly to CLUTRR, the ProofWriter dataset [37] is a collection of synthetic facts and rules with derived conclusions. The goal of this task is to infer the truth value of an unknown statement given a series of known facts and 1-hop inference rules. Each example is made of a different set of facts, rules, and an unknown statement. The model then has to predict if the unknown statement is "*True*", "*False*", or "*Unknown*" according to the input knowledge. Each fact and rule is expressed in simple templated language given by a grammar. Some examples of input/output pairs can be seen in Appendix B.

The dataset also provides the required chain of inference required to arrive at the final answer. The number of such 1-hop inference steps is considered the "depth" of the example. For instance a depth-0 (D0) example simply requires to see if the unknown statement is present in the input list of fact or not; a depth-1 (D1) example requires to apply one inference rule to one fact to arrive at the answer; etc... The dataset contains examples of up to depth-5 (D5) inference chains. We test for compositional generalisation by training on examples of up to depth 2 reasoning chains (D0, D1, D2) and testing on examples for each depth from D0 to D5.

We fine-tuned `T5-small` models on the official training and development set from the depth ≤ 2 data folder and tested it on the test set from the depth ≤ 5 data folder; consisting of 70,076 training examples and 20,030 testing examples. We trained one model for each of the 5 different strategies presented in Section 5.3, in addition to a baseline model fine-tuned without any abstraction knowledge. Unlike in all other experiments, for ProofWriter examples, we did not use the generic `spacy` named entity tagger because it did not support the entity types covered by ProofWriter examples. Instead, we used the real abstraction labels

ProofWriter	RoBERTa-large [1] CWA	no abstraction OWA	emb-sum OWA	emb-cat OWA	enc-sum OWA	enc-cat OWA	dec-loss OWA
Overall	83.9%(±0.29)	89.8%(±0.11)	90.9%(±0.07)	90.9%(±0.09)	88.4%(±0.08)	90.1%(±0.07)	**91.8%**(±0.07)
D0	**100.0%**	99.5%	99.2%	99.5%	98.9%	99.2%	99.4%
D1	**98.8%**	95.6%	93.5%	95.3%	89.5%	91.8%	95.1%
D2	**98.8%**	87.9%	81.0%	87.1%	75.3%	83.5%	86.6%
D3	71.1%	83.7%	84.6%	85.9%	81.7%	85.2%	**87.4%**
D4	43.4%	77.3%	**87.4%**	82.2%	84.7%	84.1%	85.0%
D5	37.2%	70.0%	**85.3%**	74.8%	84.0%	79.6%	80.6%

Table 5.3 Prediction accuracy on different slices of the ProofWriter D5 test set for all our models and the originally reported numbers by Clark et al. [1]. Models have been trained on depth D0, D1, D2. Models are trained in the "open-world" assumption (OWA), except for the original Clark et al. [1] model which was trained in the "closed-world" assumption (CWA). Per-depth performance is colored in shades of green for better visualisation. The red boxed area indicates test problems at depths unseen during training.

provided by the grammar files[2], and defined the following abstraction tokens: "*PERSON*", "*ATTRIBUTE*", "*ANIMAL*", "*RELATION*".

Table 5.3 shows the prediction accuracy on different slices of the D5 test set for all our models. Although trained on an older version of the dataset ("closed-world" assumption - CWA), we also report the original performance by Clark et al. [1]. While not directly comparable because of the different version of the dataset, we can still see that our `T5-small` experiments all extrapolate (D3-D5) better than the original `RoBERTa-large` model despite having less parameters. `RoBERTa-large` performs better at depths seen during training (D0-D2), which may be due to the model size being bigger (hence having greater capacity to model questions of previously seen depths), however, without abstraction, the larger model struggles to generalize to unseen reasoning depths, unlike our models.

Most importantly, if we compare with our "no abstraction" baseline model, Table 5.3 also shows that our abstraction methods help extrapolate to unseen reasoning depths. While our baseline model performs 83.7%, 77.3%, and 70% on examples from D3, D4, D5 respectively; all other abstraction-aware models extrapolate better, with the best overall model (`dec-loss`) performing 87.4%, 85%, and 80.6% on D3, D4, D5 examples respectively.

We also note that, unlike in CLUTRR, in this dataset the depth of reasoning (D0-D5) is not tied to the input length. In ProofWriter, all examples have a similar average input length, regardless of the depth of reasoning required to predict the answer. Thus the `dec-loss` model is not penalized like it was the case in CLUTRR, which result in it being the best model overall.

Overall, we achieve new state-of-the-art results on the ProofWriter dataset when trained only

[2]https://tinyurl.com/proofwritergrammars

on examples from D0-D2. This is suggesting that entity abstraction does help pre-trained Transformers to **compositionally generalize to unseen reasoning chains**. One important thing to note however is that both CLUTRR and ProofWriter sentences are relatively simple to abstract: 100% of the entities are correctly abstracted in both datasets. In the next sections we will see if explicit abstraction is still beneficial on more realistic but less formally defined tasks.

5.4.3 Multi-hop Question Answering with HotpotQA

In this section we report experiments on the multi-hop question answering (HotpotQA) dataset [38]. HotpotQA contains natural language, making it more diverse and harder to get abstraction labels than CLUTRR. In addition, HotpotQA has 2-hop inference chains both in its training and testing data splits, thus we are not able to test generalisation to longer reasoning chains. Nevertheless, we believe it is a good compromise between natural language and multi-hop reasoning.

Used in the distractor setting, each example consists of a list of 10 Wikipedia paragraphs, a question that requires the model to combine information from two paragraphs, and the answer. Since concatenating all of the 10 paragraphs would result in a context size much larger than what regular Transformer models allowed (512 or 1024 tokens), we instead only took the two golden paragraphs as context, plus the question. While this beats the original purpose of retrieving the useful paragraphs, we are not interested in achieving state-of-the art on this benchmark. We are rather interested in using it to compare the usefulness of our approach on a more natural multi-hop question-answering setup. An example of input/output sequence pair can be seen in Appendix B.

Because the official test set is not public, we used the official validation set as our test set to compare our models and fine-tuned a `T5-small` model on 90% of the training set while keeping the remaining 10% as our custom validation set for early stopping. We trained one model for each of the 5 different strategies presented in Section 5.3, in addition to a baseline model fine-tuned without any abstraction knowledge.

Table 5.4 shows exact match, F1 scores, Precision and Recall on our test set. We can see from these results that the best of our models is the abstraction-aware `dec-loss` model (trained to predict both the answer and the input in its abstract form) with an F1 score of 69.8% against 68.9% for the baseline model ("no abstraction" row). However, the baseline model is a strong candidate and the abstraction does not always benefit the model depending on how it is incorporated. This may be due to the fact that entity abstraction labels are harder to predict on natural language, and that entity type abstraction may not be required in

HOTPOTQA	ExactMatch	F1	Precision	Recall
no abstraction	54.7 (±0.01)	68.9 (±0.01)	72.4 (±0.01)	69.1 (±0.01)
emb-sum	54.3 (±0.01)	68.4 (±0.01)	72.0 (±0.01)	68.6 (±0.01)
emb-cat	52.6 (±0.01)	66.5 (±0.01)	69.7 (±0.01)	67.0 (±0.01)
enc-sum	54.2 (±0.01)	68.5 (±0.01)	72.0 (±0.01)	68.7 (±0.01)
enc-cat	52.9 (±0.01)	67.2 (±0.01)	71.0 (±0.01)	67.3 (±0.01)
dec-loss	**55.7** (±0.02)	**69.8** (±0.01)	**73.3** (±0.01)	**69.9** (±0.01)

Table 5.4 Average test performance for all models on HotpotQA. Average and standard deviation computed with 3 random seeds.

problems with little to no formal logical structure. We further discuss this in Section 5.6.

In an effort to contextualize these results, we note that our models performance is slightly above the "Query Focused Extractor" [201] performance of 53.86 Exact Match and 68.06 F1. At the time of writing, the SOTA model on HotpotQA is the "From Easy to Hard" model [202] with Exact Match of 71.89 and F1 score of 84.44. We note however that the test & training data in our setting is slightly different, so these are not perfectly comparable results. In addition, we are more interested in evaluating the effect of entity abstraction by comparing our model variants.

5.4.4 Conversational Question Answering with CoQA

Eventually, motivated by the use of a generative model, we test the same abstraction strategy in a conversational setting. For that, we leveraged the conversational question answering dataset CoQA [39]. The task presented by this dataset is to understand a text passage and answer a series of inter-connected questions in a conversation. The conversation aspect introduces follow-up questions that forces the model to keep track of what entity is currently being referred to and to look back at previous interactions. Examples of input/output sequence pairs can be found in Appendix B. The dataset does not always explicitly forces multi-hop reasoning steps, but it could still happen (*i.e.* see the last CoQA example of Appendix B in which the model must perform a substraction between all subjects and the ones already mentioned). In addition, the conversational nature of this dataset often introduces a co-reference step to be made before fetching the information from the paragraph in context. In this setting we will thus test if abstraction can help in this multi-step information retrieval procedure.

Similarly to HotpotQA, because the official test set is not public, we used the official validation set as our test set to compare our models and fine-tuned a `T5-small` model on 90% of the training set while keeping the remaining 10% as our custom validation set for early

stopping. We trained one model for each of the 5 different strategies presented in Section 5.3, in addition to a baseline model fine-tuned without any abstraction knowledge.

CoQA	**ExactMatch**	**F1**
no abstraction	66.0%(±0.001)	74.4%(±0.000)
emb-sum	65.7%(±0.002)	74.2%(±0.002)
emb-cat	63.8%(±0.001)	72.3%(±0.001)
enc-sum	65.8%(±0.001)	74.1%(±0.002)
enc-cat	65.2%(±0.002)	73.8%(±0.002)
dec-loss	**66.4%**(±0.001)	**74.9%**(±0.001)

Table 5.5 Average test performance for all models on CoQA. Average and standard deviation computed with 3 random seeds.

Table 5.5 shows the exact match and F1 score on our test set. Although by a small margin, we can see that the best of our model is again the abstraction aware `dec-loss` model (trained to predict both the answer and the input in its abstract form) with an F1 score of 74.9% against 74.4% for the baseline ("no abstraction" row). The baseline model is quite strong already and the benefit of abstraction is questionable in this case. We further discuss this result in the following section. As in previous experiments, the worst performing model is `emb-cat` and one of the best is the `enc-sum` model (after `dec-loss` in this case).

In an effort to contextualize these results, we note that our models performance is better than DrQA + seq2seq with copy attention [39] and BiDAF++ [203] with an F1 score of 67.0 and 71.6 respectively. The next closest model in the CoQA leaderboard is the FlowQA model [204] with an F1 score of 76.3. At the time of writing, the SOTA model on CoQA is RoBERTa+AT+KD [205] with an F1 score of 90.9. We note however that the test data in our setting is slightly different, so these are not perfectly comparable results. In addition, we are more interested in evaluating the effect of entity abstraction by comparing our model variants.

5.5 Discussion

In this section we aim to analyse further the results presented above by comparing them and measuring some key characteristic about each datasets.

Let's first summarize the results from all experiments. Overall, in all datasets (except for CLUTRR) the best model on average was the `dec-loss` model, which is trained to predict both the target output sequence and the input sequence in its abstracted form. As discussed previously, we believe that the reason why the `dec-loss` model did not perform as good as

the other models in CLUTRR is because input sequence length is tied to the reasoning depth required to answer the question. Indeed, a question of level n ($2 \leq n \leq 10$) will have exactly n sentences in its input. Thus, during testing, the model must generate sequences of unseen lengths. Previous work showed that length generalization is a common weakness of classical language models [1, 196, 206–208]. Overall, our results suggest that **when input sequence length is relatively stable across all examples, the `dec-loss` abstraction strategy is beneficial to multi-step reasoning tasks**.

We then investigate why results on the two "natural", less procedural tasks (HotpotQA & CoQA) do not yield strong conclusions like in the synthetic cases of CLUTRR & ProofWriter. The first major difference between these datasets is that CLUTRR and ProofWriter are designed explicitly to test for compositional generalisation and reasoning extrapolation. In both datasets, the model is tested on examples of longer reasoning chains than what is observed during training. The language vocabulary is limited and the required reasoning depth is controlled. This is possible when working with tasks that are formally defined in a logical reasoning setting. On the other hand, HotpotQA and CoQA are designed from human written text (Wikipedia for Hotpot, human conversations for CoQA). This natural setting implies two things. (1) It is harder to control the reasoning depth required: all HotpotQA examples require to combine two pieces of information in order to answer the question. CoQA examples require to solve long co-reference resolution chains, however the test set does not contain longer chains of reasoning. (2) The language vocabulary is more noisy, making it harder to extract the useful information from a piece of text. These distinctions suggest that **entity type abstraction is mostly beneficial for formally defined logical tasks in which the model must reason at unseen depths during inference time**.

A second differentiating factor between these datasets is the quantity and quality of entity tags provided to the model. In an effort to estimate the influence of the tagger accuracy on our results, we compare each dataset in terms of the amount of entity tags they contain, as well as the accuracy of these tags. The reported performance of the entity tagger we used is 0.85 F1[3]. To further validate this metric, for each dataset, we estimate (i) the percentage of tokens tagged as entities, (ii) the

Dataset	Entity Tokens	F1
CLUTRR	22.2% (±0.019)	100% (±0)
ProofWriter	36.0% (±0.036)	100% (±0)
HotpotQA	37.0% (±0.085)	88.5% (±0.092)
CoQA	17.7% (±0.095)	88.4% (±0.085)

Table 5.6 Percentage of tokens being tagged as entities and estimated F1 score of the tagger on each dataset.

[3]https://tinyurl.com/encoreweblg

correctness of the tagged entities (precision), and (iii) the amount of correctly tagged entities out of all the entities that should have been tagged (recall). We estimate these metrics by manual inspection of a random set of examples for each dataset and report the F1 metric in Table 5.6. We can see that the percentage of tokens tagged as entities is roughly the same across datasets, except for CoQA which has fewer entity tokens. This is likely due to the fact that a lot of entities are referred to by co-reference in conversational QA. The average quality of entity tags for HotpotQA and CoQA is in line with the reported 85% F1 performance of the tagger. However, we note that all entities in CLUTRR and ProofWriter are correctly labeled. This distinction suggests that **the quality of the entity tagger can influence the benefits of performing entity type abstraction**.

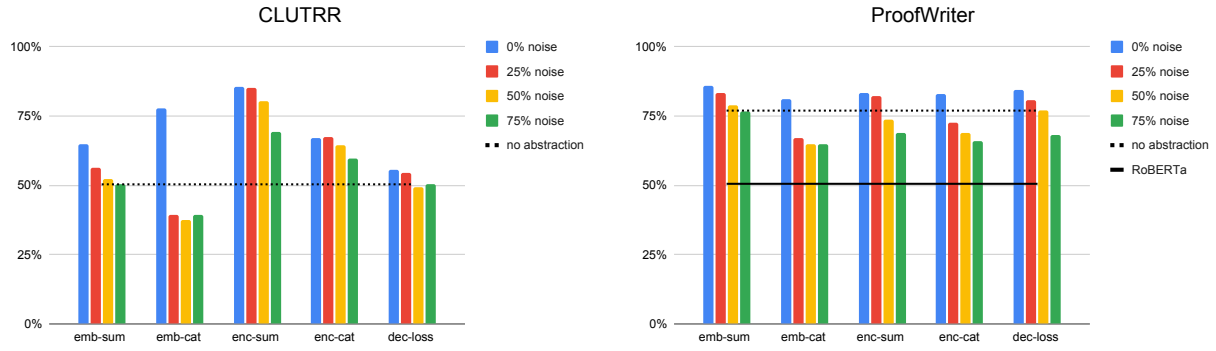


Figure 5.1 Average answer accuracy on extrapolation reasoning levels for models trained with 0%, 25%, 50% and 75% entity tagger noise on CLUTRR and ProofWriter. The baseline model (no abstraction) is represented by a black dotted line. Previous work on ProofWriter is represented by a black solid line.

To further estimate this influence, we ran experiments on the CLUTRR and ProofWriter benchmarks in which we added noise to the abstract labels, thus artificially simulating a weaker tagger. In particular, we modified 25%, 50%, and 75% of the entities by either replacing their entity type or by deleting the tag altogether. We report in Figure 5.1 the average answer accuracy of our models across extrapolation reasoning levels of the test set for both benchmark dataset (lvl 7-10 for CLUTRR and D3-D5 for ProofWriter). Overall, we can see that as the tagging noise increases, all models see their performance reduced and sometimes under-performing the baseline when the performance of the tagger is too weak. For instance once the noise reaches 50% or more, all models on ProofWriter and almost all models on CLUTRR (except `enc-sum` & `enc-cat`) become weaker than the baseline. This confirms that **as the tagger accuracy decreases, the overall performance of the model does so to**. In addition, it also shows that it is better for the model to not have additional complications (abstraction in this case) if the additional parameters needed for

that are used on noise more than 50% of the time.

In an effort to better characterize the failure cases of our models on the more natural datasets we report in Appendix C model predictions on various datasets. In particular, Appendix C shows examples of HotpotQA in which abstraction models answered correctly but not the baseline (no-abstraction) model and vice-versa. Overall, we observe that many examples in HotpotQA don't require multiple reasoning hops, which can be an additional source of noise beside the tagging accuracy.

Overall, we believe three factors are influencing our results on the more natural tasks: (i) the effect of a weaker entity tagger performance as shown in Table 5.6 and Figure 5.1, (ii) the depth of reasoning being relatively shallow and the same across training and testing sets as observed in Appendix B and C, (iii) the natural and conversational language being further away from a formal language. These factors result in "real-world" problems not having strong enough logical structure which can benefit from the abstraction technique. It is only when tasked on more logical problems explicitly requiring reasoning depths unseen during training that abstraction becomes significantly beneficial.

5.6 Conclusion

Conclusion. We presented various ways to incorporate abstract knowledge into Transformer Language Models. Focusing on entity types, this work evaluated model performance on reasoning tasks requiring compositional generalization and multi-hop reasoning. Overall our results demonstrate three things: (i) incorporating abstract knowledge significantly improves reasoning and compositional generalization in both interpolation and extrapolation when the environment is formally defined in a logical reasoning setting; (ii) different ways to incorporate abstraction yields different performance boosts: `enc-sum` and `dec-loss` are generally performing better than others; (iii) abstraction is not beneficial when the task at hand is more natural, less procedural, and not requiring long reasoning chains. This last result is due to the noisy entity tagging from "off-the-shelf" taggers, and due to the nature of the task at hand.

Limitations. One limitation of our work is that the method we present requires annotated data which is not always available. Furthermore, this additional data processing can take time and may not scale well to larger datasets if implemented naively. Overall, we did not find significantly longer training time for all approaches except for the `dec-loss` model that was training on average 2 times longer than the other methods. The most important factor in training time was the dataset used rather than the method used. Models trained on

natural language datasets CoQA and HotpotQA took days to train while models trained on CLUTRR and ProofWriter took a few hours.

Future Explorations. One hypothesis that results from our work is the following question: could *pre-training* Transformer models with additional abstraction data result in stronger performance when fine-tuned with abstraction data like we do in this work? Although we could not train *from-scratch* a T5 model on its original C4 dataset [52], we believe that augmenting C4 with abstract annotations like we do on a smaller scale and training T5 from scratch on this augmented dataset could potentially yield a stronger language model.

Another interesting future direction worth exploring would be the ethical benefit of incorporating explicit abstraction into large language models. Previous work showed that pre-trained language models can have some undesired societal biases [192, 209, 210]. Although not explored in this work, we believe that giving abstract entity types like we do in this work could have a positive societal impact on language models, potentially alleviating some of these biases. For instance, through abstraction, a model can be exposed to female and male names both being "*PERSON*"s and that a "*PERSON*" can equally be a "*manager*" or an "*assistant*" regardless of its gender. Similar exposure could be achieved with other abstraction types such as "*RELIGION*", "*JOB*", "*COUNTRY*", "*NATIONALITY*", etc... Benchmarks such as StereoSet [211] could be used to measure the beneficial impact that explicit abstraction can have.

Acknowledgments

The authors acknowledge support from ServiceNow Research for providing computational resources which were used to run the experiments in this work. The first author is partially funded by a scholarship from the Fonds de Recherche du Quebec Nature et Technologie (FRQNT). We thank CIFAR for their support through the CIFAR AI Chairs program. We also thank NSERC and PROMPT for their support.

5.A Hyperparameters

We used the default `T5-small` hyperparameters from the `HuggingFace` library [193]. We present in Table 5.7 below the library version we used and the model hyperparameters used for all experiments.

The only difference between each dataset is the number of same-type entity tags allowed in each input sequence n. These values were chosen to be the smallest possible number while still being able to identify all the different entities in one sequence. For CLUTRR the maximum number of same-type entities was 20, for ProofWriter it was 10, and for HotpotQA and CoQA it was 100. This is due to the larger context size and the diversity of natural language texts. Results on CLUTRR when $n = 1$ can be found below in Appendix 5.C.

n for CLUTRR	20
n for ProofWriter	10
n for HotpotQA	100
n for CoQA	100
AllenNLP	version 2.2.0
Transformers	version 4.4.2
Spacy	version 2.3.5
batch size	256
16-bit floating point	True
dim embedding	512
dim feedforward	2048
dim key-value	64
dropout	0.1
max length	512
#of heads	8
#of layers	6
optimizer	AdamW
learning rate	1.00E-05
betas	[0.9, 0.999]
epsilon	1.00E-08
gradient norm	1.0
sampler	top-p
p	0.9
temperature	1.0

Table 5.7 Library version and model hyper-parameters.

5.B On the abstraction accuracy of the `dec-loss` model

After training the `dec-loss` models for each task, one question we might ask is whether the model is able to correctly predict abstract tokens with its dedicated language model head.

After generating abstract sequences for all test sets, we measured that the model is correctly predicting entity *types* 100% of the time for CLUTRR and ProofWriter, and around 70% of the time for HotpotQA and CoQA compared to the types predicted by our "off-the-shelve" tagger.

One interesting finding though, is that the model is not consistent with entity numbers across the same example. While it can correctly predict the *type* of an entity ("PERSON" vs "LOCATION"), it is almost impossible to stay consistent with entity IDs of that type ("PERSON_11" vs "PERSON_23"). This suggests that at inference time, the distribution of abstract tokens belonging to the same entity type is very close to uniform, which also suggests that all these token embeddings are close to each other.

5.C CLUTRR results when `n=1`

To better understand the effect of the hyper-parameter n in our models, we ran additional experiments on the CLUTRR benchmark for different values of n. When $n = 20$ all entities in each example gets a unique abstract token ID. When $n = 1$, all entities of the same type are mapped to the same abstract token. In order to keep track of which entity is related to which, token identity must be preserved. When $n = 1$, the only way to preserve token identity is for the model to use the original (non abstracted) sequence. We report below the average answer accuracy of our models on CLUTRR for different values of n. The average is taken across all test levels from 2 to 10.

	n = 20	**n = 1**
no abstraction	62.9%	
emb-sum	71.5%	**75.0%**
emb-cat	**64.8%**	30.4%
enc-sum	**88.8%**	85.3%
enc-cat	**73.8%**	36.1%
dec-loss	64.2%	**75.7%**

Table 5.8 Average CLUTRR answer accuracy for models trained with $n = 20$ and $n = 1$ token per entity tag.

We can see that the `emb-cat` and `enc-cat` models perform much worse in the $n = 1$ setting, while the other models are relatively similar in both settings. This suggests that concate-

nating representations together and then performing a matrix multiplication to resize the representation does not keep entity identity as well as the other methods.

CHAPTER 6 LONG-CONTEXT LANGUAGE DECISION TRANSFORMERS AND EXPONENTIAL TILT FOR INTERACTIVE TEXT ENVIRONMENTS

This chapter presents the last contribution of this thesis. Here we explore the multi-step reasoning capacity of TLMs in the context of text-interactive environments. Text-interactive environments such as text games are by design testing the multi-step reasoning capacity of their agent since they consist of achieving a final goal by interacting multiple times with an environment through text commands. Traditionally used to evaluate Reinforcement Learning agents, we instead propose an offline method that learns from a collection of previous trajectories. We condition our method on a metric that measures the quality of the said trajectory in the hopes of controlling the agent's quality at test time. As such, our method falls into the Reinforcement Learning via Supervised Learning (RvS) paradigm [64].

Text-based game environments are challenging because agents must deal with long sequences of text, execute compositional actions using text and learn from sparse rewards. We address these challenges by proposing Long-Context Language Decision Transformers (LLDTs), a framework that is based on long transformer language models and decision transformers (DTs). LLDTs extend DTs with 3 components: (1) exponential tilt to guide the agent towards high obtainable goals, (2) novel goal conditioning methods yielding significantly better results than the traditional return-to-go (sum of all future rewards), and (3) a model of future observations. Our ablation results show that predicting future observations improves agent performance. To the best of our knowledge, LLDTs are the first to address offline RL with DTs on these challenging games. Our experiments show that LLDTs achieve the highest scores among many different types of agents on some of the most challenging Jericho games, such as Enchanter.

6.1 Introduction

People spend a significant fraction of their lives performing activities closely linked with natural languages, such as having conversations, writing e-mails, filling out forms, reading and writing documents, and so on. Recently, the excitement around the use of Large Language Models (LLMs) for dialogue has brought the setting of interactive dialogue into the spotlight. Interactive text-based games allow one to explore and test interactive agents, alternative neural architectures, and techniques. However, text environments remain challenging for existing Reinforcement Learning (RL) agents since the action space is vast due to the compositional

nature of language, making exploration difficult. Fortunately, language has the advantage that knowledge can often be reused across environments, such as the fact that fire *burns* or that doors *open*. To solve real-world text-based tasks and play rich text-based games well, RL agents can also benefit from the knowledge about the human world acquired from large offline data sources by leveraging pre-trained LLMs.

In real-world settings, the low-performing behavior exhibited by online RL agents during learning makes them impractical to use with humans in the loop. This situation arises in many other contexts [65] and has motivated a lot of research on offline RL. Offline RL methods have a long history, but more recently, several approaches have been proposed that focused on using powerful transformer-based sequence models, including Trajectory Transformers (TTs) [9], and Decision Transformers (DTs) [3]. However, these approaches are formulated and examined within continuous control robotics problems. Unlike the methods above, our approach is designed to handle the complexity and richness of human language by leveraging pre-trained LLMs.

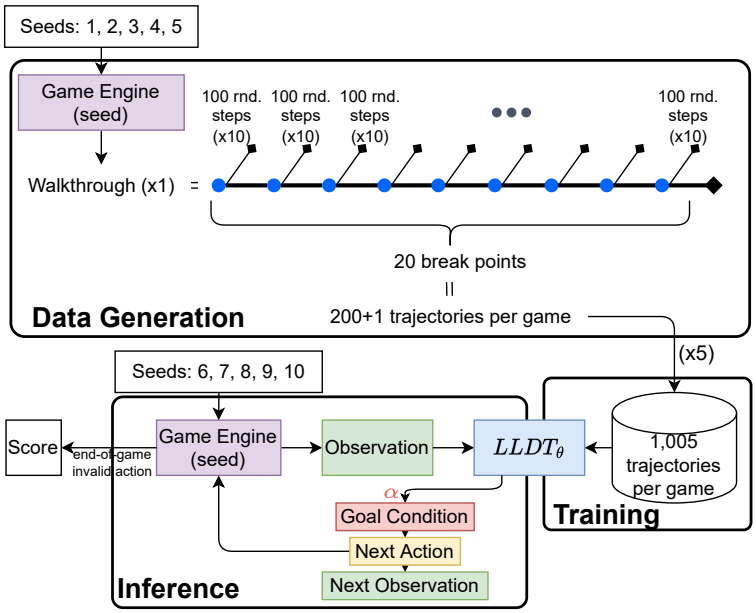


Figure 6.1 Overview of our approach: Noisy trajectories are generated from a high quality game walkthrough by taking 100 random steps at each 5% of the trajectory. The collection of trajectories on multiple games is used to train our LLDT model offline to predict a goal condition, next action, and next observation. The LLDT is then evaluated in each game environment, initialized with 5 random seeds.

Motivated by the analogy of text-games to intelligent text assistants helping people with various tasks, we assume that a few expensive expert demonstrations are available for learning. As such, we use the Jericho text games [40], which provide a single golden path trajectory per game. To create a large and diverse dataset, we then generate trajectories with perturbations from that golden path as described in Section 6.4.2 and depicted in Figure 6.1. The complexity and richness of Jericho games make them a reasonable proxy for the kind of data one might obtain in real-world assistive agent settings.

In this work, we use a pre-trained Transformer language model that we fine-tune on offline game trajectories to predict: trajectory goal conditions, future actions, and observations. To

sample high goal conditions from our model, we convert distributions over discrete token representations of numerical values into continuous ones, allowing us to maximize goal conditions that are likely achievable through an exponential tilting technique. In addition, we compare different conditioning methods and introduce an auxiliary loss to predict future observations. We will refer to our approach as Long-Context Language Decision Transformers (LLDTs) with exponential tilt. Our approach is visualized in Figure 6.2. See Table 6.1 for a comparison of how our formulation for density estimation and decision-making is situated with respect to prior frameworks. We also note that none of these previous frameworks have been applied to text-based action spaces, so none have leveraged pre-trained LLMs as in our framework.

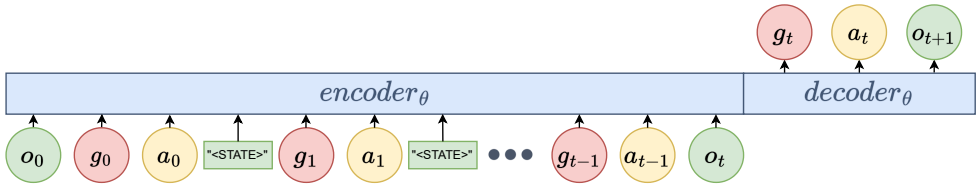


Figure 6.2 Our Long-Context Language Decision Transformer framework. A trajectory of length T is split at a random index $t \in [0, T-1]$. The model encodes the sequence of observations (o), goal conditions (g), and actions (a) up to time step t. The first o_1 and last o_t observations are fully written, but to shorten the input sequence, the other intermediate observations are replaced by a special "< STATE >" token. The decoder predicts the goal condition g_t, action to take a_t, and next observation o_{t+1}.

To conclude, our contributions can be summarized as follows: (1) Our work is the first to address the challenging Jericho text-based games in an offline return conditioned sequence learning setup, wherein we train models on noisy walkthrough trajectories from multiple games simultaneously. (2) We improve agent behavior with fewer assumptions by letting the model predict goal conditions in a manner where no knowledge of the maximum score is needed through our use of an exponential tilting technique. (Section 6.5.1). (3) We explore and empirically compare 3 novel definitions of goal conditioning that perform better than the return-to-go perspective of Decision Transformers (DTs). (Section 6.5.2). (4) We propose a novel auxiliary loss to train DTs that draws parallels to model-based RL and empirically shows better performance compared to the traditional model-free loss of DTs (Section 6.5.3).

6.2 Methodology

6.2.1 Problem setup

Text-based games can be formulated as partially observable Markov decision processes (POMDP) described by (S, T, A, O, R, γ). The current game state $s_t \in S$ is partially observable in $o_t \in O$ which is often a text description of the current scene (inventory, location, items). The agent can take an action $a_t \in A$ to interact with the environment and causes a state change based on a transition function $T(s_t, a_t)$ leading to a new state $s_{t+1} \in S$. Some games are stochastic in that the same action for the same state can lead to different states. Once the agent transitions to the new state, a reward r_t is given by an unknown reward function $R(s_t, a_t)$ that the game designers defined. The reward can either be positive, negative, or neutral.

Offline Reinforcement Learning. The goal of the agent is to learn a policy $\pi(a_t|s_t)$ which maximizes the expected return $\mathbb{E}[\sum_{t=0}^{T} r_t]$ in the POMDP by observing a series of static trajectories obtained in the same or similar environments. Each trajectory is defined as $\tau = (o_0, a_0, r_0, o_1, a_1, r_1, ..., o_T, a_T, r_T)$, and it is obtained by observing rollouts of arbitrary policies. This setup is similar to supervised learning, where models are trained from a static dataset. It is more difficult than online reinforcement learning since agents cannot interact with the environment to recollect more data.

Reinforcement Learning in text-based games. One of the main differences between traditional RL environments, such as Atari or Mujoco, and text-based environments is that both A and O consist of text. Therefore, due to the compositional nature of language, A is significantly more complex than in common RL scenarios, where the action space is restricted to a few well-defined actions. To deal with such complexity, we model A, O and R with a large pre-trained language model: $x_i = \text{LLM}(x_i|x_{1:i-1})$, where x_i is the ith text token in a text sequence of length L. The goal is that the LLM uses its pre-existing knowledge about the world (e.g., doors can be *opened*), to propose valid actions given an observation.

Decision Transformers. To perform offline learning on text-based games, we adapt the language model (particularly LongT5 [58]) to be a decision transformer (DT) [3] which abstracts reinforcement learning as a sequential modeling problem. DTs are trained with the language modeling objective on sequences of $\{g_t, o_t, a_t\}_{t=0}^{T}$ triples, where the goal condition g_t is defined as the undiscounted sum of future rewards, or return-to-go: $g_t = \sum_{i=t}^{T} r_i$. Consequently, we have a model that can be conditioned on a desired goal (or return, in this case).

In the following subsections, we discuss the novelties we bring to the original formulation of DTs.

6.2.2 Goal conditioning

One limitation of DTs is that the best final score of a game must be known to condition on it at the first step with g_0 [3]. Although we have g_0 for the training trajectories, it is impossible to know the best target score when starting a new game. This is especially problematic for Jericho games where maximum scores vary greatly between games [40].

One solution is to normalize g_0 during training with the maximum game score. This procedure leads to goal conditions between 0 and 1 for the training games and allows to use an initial goal condition of 1 at test time. However, this solution also assumes that we know the maximum score of every game since intermediate rewards returned by the environment r_t also need to be normalized to update g_{t+1}:

$$g_{t+1} = g_t - \frac{r_t}{\text{max score}}, \tag{6.1}$$

To remove the dependence on manual goal conditioning, we take a similar approach to Lee et al. [7] and train the model on ordered sequences of $\{o_t, g_t, a_t\}_{t=0}^{T}$ triples instead of $\{g_t, o_t, a_t\}_{t=0}^{T}$. Moving the goal condition g_t after the observation o_t allows us to predict the goal condition based on the current observation rather than manually defining it. This variation allows deciding at inference time if we want to sample or replace the value of g_t based on $P_\theta(g_t|o_t)$ (θ being the parameters of our system) by modeling the joint probability of a_t and g_t as:

$$P_\theta(a_t, g_t|o_t) = P_\theta(a_t|g_t, o_t) \cdot P_\theta(g_t|o_t). \tag{6.2}$$

One challenge is that sampling g_t can produce low and inaccurate target returns. To alleviate this issue, we perform exponential tilting on the predicted probabilities of g_t. In particular we sample g_t like so:

$$g_t = \text{argmax}_{g_t}\Big[P_\theta(g_t|o_t) \cdot \exp(\alpha g_t)\Big], \tag{6.3}$$

with $\alpha \geq 0$ being a hyper-parameter that controls the amount of tilting we perform. This allows us to sample high but probable target returns. We compare results with $\alpha = \{0, 1, 10, 20\}$ in Section 6.5.1.

Another significant advantage of predicting the goal condition g_t based on o_t is that we can explore various strategies of goal conditions that cannot be defined manually at inference time. We describe below the original return-to-go used by decision transformers and three

novel goal condition strategies.

Return-To-Go (RTG): $g_t = \sum_{i=t}^{T} r_i$, is the original strategy of the return-to-go. It is the undiscounted sum of future rewards, which will be high at the beginning of trajectories achieving a high score. These values will decrease as the agent progresses since fewer future rewards will be available in a trajectory with intermediate rewards.

Immediate Reward (ImR): In the setting where $g_t = r_t$, each step is conditioned on the reward observed right after the predicted action. We expect that with this goal condition method, the agent will learn what type of actions usually yield higher rewards (opening chest -vs- moving in a direction). We expect this strategy to encourage the model to get high rewards as fast as possible. However, we expect this strategy to work well only for environments with dense reward signals.

Final Score (FinS): $g_t = \sum_{i=0}^{T} r_i$. In this setting, each step is conditioned on the final score achieved by the agent. The final score is defined as the sum of all rewards observed during the entire trajectory. Note that, unlike all the other goal condition definitions, this score will not change over the course of a trajectory. This setting is closer to the traditional RL paradigm in which we often define rewards based on the final performance of an agent: did it win or did it lose. We expect the agent to learn to differentiate successful from unsuccessful trajectories in this setting. Since the model is not conditioned on immediate rewards, we expect it will produce longer trajectories, which can eventually achieve higher final scores.

Average Return-To-Go (AvgRTG): $g_t = \frac{\sum_{i=t}^{T} r_i}{(T-t)}$. In this setting, each step is conditioned on the average of all future rewards. This is also defined as the return-to-go divided by the number of steps remaining. The motivation for this goal condition is that it will capture the sparsity of rewards in a trajectory, unlike all the others.

To reduce the variance in the numbers observed between different games, all goal condition numbers during training are normalized by the maximum score of the current game, where:

$$g_t = \text{int}\left[100 \cdot \frac{g_t}{\text{max score}}\right]. \tag{6.4}$$

At inference time, we can either manually specify goal condition numbers (assuming we know the game maximum score), or we can let the model predict those goal condition numbers with exponential tilt (more flexible).

We experiment with all these goal condition definitions in our experiments and report results in Section 6.5.2.

6.2.3 Next State Prediction

	Density Estimation ($\mathcal{L}(\boldsymbol{\theta})$)	Decision Making ($\pi(\mathbf{a}\|\mathbf{s};\eta)$)
DTs	$\log p_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{o}_t, G_t)$	$p_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{s}_t, G_t)$
RWR	$\exp(\eta^{-1}G_t)\log p_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{o}_t)$	$p_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{s}_t)$
RCP	$\log p_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{o}_t, G_t)p_{\boldsymbol{\theta}}(G_t \mid \mathbf{o}_t)$	$p_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{s}_t, G)p_{\boldsymbol{\theta}}(G \mid \mathbf{s}_t)\exp(\eta^{-1}G - \kappa(\eta))$
RBC	$\log p_{\boldsymbol{\theta}}(G_t \mid \mathbf{o}_t, \mathbf{a}_t)p_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{o}_t)$	$p_{\boldsymbol{\theta}}(G \mid \mathbf{s}_t, \mathbf{a})p_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{s}_t)\exp(\eta^{-1}G - \kappa(\eta))$
IRvS	$\log p_{\boldsymbol{\theta}}(\mathbf{a}_t, G_t \mid \mathbf{o}_t)$	$p_{\boldsymbol{\theta}}(\mathbf{a}, G \mid \mathbf{s}_t)\exp(\eta^{-1}G - \kappa(\eta))$
MB-RCP (ours)	$\log p_{\boldsymbol{\theta}}(\mathbf{o}_{t+1} \mid \mathbf{a}_t, \mathbf{o}_t, G_t)p_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{o}_t, G_t)p_{\boldsymbol{\theta}}(G_t \mid \mathbf{o}_t)$	$p_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{s}_t, G)p_{\boldsymbol{\theta}}(G \mid \mathbf{s}_t)\exp(\eta^{-1}G - \kappa(\eta))$

Table 6.1 Comparison of different policy training and action selection techniques (adapted from Piche et al. [2]). We compare our approach with Decision Transformers (DTs) [3], Reward Weighted Regression (RWR) [4, 5], Reward-Conditioned Policies (RCP) [6] (also used by Multi-Game Decision Transformers [7]), Reweighted Behavior Cloning (RBC) [8] (also used by Trajectory Transformer (TT) [9]), and Implicit RL via supervised learning (IRvS) [2]. Where $\mathbf{s}$ represents the state as encoded by the model and depends on the architecture and inputs used.

To give more training signal to the model and make it more robust to stochastic environments, we also experiment with learning to predict the next observation o_{t+1}. Concretely, we predict o_{t+1} after taking action a_t in state s_t. Although the prediction of the next observation is not used to interact with the environment at test time, we believe that the agent will perform better if it can predict how its action will impact the world. Furthermore, predicting the next observation indirectly informs the model about the stochasticity of the environment. This technique draws parallels with the model-based paradigm in Reinforcement Learning, where the agent can predict how the environment will evolve after each action. Formally, the model estimates the following probability:

$$\begin{aligned} P_\theta(o_{t+1}, a_t, g_t|o_t) =& P_\theta(o_{t+1}|a_t, g_t, o_t)\cdot \\ & P_\theta(a_t|g_t, o_t)\cdot P_\theta(g_t|o_t), \end{aligned} \tag{6.5}$$

which is a type of Reward Conditioned Policy (RCP) with the additional term $P_\theta(o_{t+1}|a_t, g_t, o_t)$. We call our technique model-based reward conditioned policy (MB-RCP). We compare our formulation to prior work in Table 6.1. We are interested in using this additional prediction as a form of regularization and therefore treat predicting the next observation as an auxiliary loss, leading to:

$$\mathcal{L} = \frac{L_{CE}([\hat{g}_t\hat{a}_t]; [g_t a_t]) + \lambda \cdot L_{CE}(\hat{o}_{t+1}; o_{t+1})}{1+\lambda}, \tag{6.6}$$

with L_{CE} being the regular cross entropy loss and λ being a hyper-parameter set to 0.5 in all our experiments. This weighted average prevents the model from spending too much of its representation power on the next observation prediction, as it is not strictly required to be able to interact in an environment. At inference time, only the next goal condition and next action predictions will be used. We perform an ablation study on this aspect of our approach by comparing models trained with ($\lambda = 0.5$) and without ($\lambda = 0$) this auxiliary loss and report our results in Section 6.5.3.

6.3 Related Work

Upside-down RL (UDRL) [2,6,162] poses the task of learning a policy as a supervised learning problem where an agent is conditioned on an observation and a target reward to produce an action. Instead of generating the next action for a target reward, goal-conditioning methods generate trajectories conditioned on an end-goal [166, 212]. Most relevant to our work, Chen et al. [3] recast supervised RL as a sequence modeling problem with decision transformers (DTs), but they did not examine text environments. DTs have been extended to multi-task environments by training them on multiple Atari games [7]. To address the problem of modeling text-based environments Furman et al. [213] proposed DT-BERT for question answering in TextWorld environments [151]. However, the maximum number of steps in their trajectories is 50, and the environments are only differing in their number of rooms and objects. Here we go a step further and propose to learn agents that fully solve Jericho games [40] with diverse game dynamics and scenarios by training on offline trajectories across multiple games.

Jericho is a challenging python framework composed of 33 text-based interactive fiction games [40]. It was initially introduced with a new Template-DQN, and compared with the Deep Reinforcement Relevance Network (DRRN) [150]. However, both methods are trained online, which requires an expensive simulator and requires domain-specific knowledge, such as the set of possible actions in order to be trained. Yao et al. [160] proposed CALM, extending DRRNs to solve the problem of it needing to know the set of possible actions in advance. They use a GPT-2 [24] language model to generate a set of possible candidate actions for each game state. Then, they use an RL agent to select the best action among the (top-k=30) generated ones.

One of the main challenges of leveraging language models to solve Jericho games is to encode the full context of the game trajectory. As such, KG-A2C [157] and Q*BERT [214] use a knowledge graph to represent the environment state at each step and learn a Q-value function. SHA-KG [159] uses graph attention network [101] to encode the game history and learn a

value function. RC-DQN [215] uses a reading comprehension approach by retrieving relevant previous observations, encoding them with GRUs [20], and learning a Q-value function. DBERT-DRRN [55] leverages a DistilBERT to encode state and action and feed it to an MLP to learn a Q-value function. XTX [216] re-visits different frontiers in the state space and performs local exploration to overcome bottleneck states and dead-ends. CBR [217] stores previous interactions in memory and leverages a graph attention network [101] to encode the similarity between states.

The above previous methods are online-based RL, thus suffering from sample inefficiencies. Here, we take a simpler approach by simply leveraging long context transformers like LongT5 [58] to model the sequence of state observations, target goal scores, and actions of past game trajectories as a sequence of tokens. Then, given a state observation, we leverage exponential tilt [2, 7] to produce the action with the best possible target goal score. We find that our LLDT approach is effective enough to outperform all previous methods that we have examined on Jericho games.

6.4 Experimental setup

6.4.1 Jericho Engine

Jericho is a well-known Python framework that consists of 33 text-based interactive fiction games that are challenging learning environments [40]. Developers manually create them, each having its own way of defining the rules and goals for each game, making the games quite diverse.

Text adventure games are challenging on their own because of their combinatorially large action space and sparse rewards. Usually, text adventure games have a large action vocabulary (around 2000 words on average), and each action is made of multiple words (1 to 4 on average). This makes the action space as big as $2000^4 = 1.6 \times 10^{13}$. To alleviate this issue, the Jericho benchmark provides a list of valid actions for each state. However, this makes the environment much slower as the game engine validates all possible actions against the simulator. In addition, the action space becomes dynamic as it changes from state to state. The above challenge in combination with extremely sparse rewards makes text adventure games very challenging for current RL methods.

In this work, we focus on a subset of Jericho games, in particular, the ones belonging to the Zork Universe: *enchanter*, *sorcerer*, *spellbrkr*, *spirit*, *ztuu*[1]. We generate trajectories (Section 6.4.2) for each of these games and train our model on the collection of all trajectories

[1] we selected games belonging to the same universe to favor transfer of knowledge between games.

from all games.

6.4.2 Data Collection

Jericho provides one human walkthrough trajectory per game that achieves the maximum score. However, since some games are stochastic, every walkthrough is only valid for a specific default seed when initializing the game. To obtain a more diverse dataset with incorrect or partially correct trajectories, we propose to generate trajectories by following the walkthrough trajectories for some steps and then deviating from them. Concretely, to collect a large number of trajectories with different performances we follow the walkthrough trajectory for $X\%$ of its total number of steps and then take 100 additional random steps. We repeat that procedure 10 times for each $X \in [0, 5, 10, ..., 85, 90, 95]$. When $X = 0\%$, this is the same as a fully random trajectory. When $X = 95\%$, the agent follows the walkthrough path for 95% of the steps and then takes 100 random steps. This results in a collection of 201 trajectories, including 1 original walkthrough for each game. Note that we also tried to include TDQN and DRRN trajectories trained on individual games, but these agents did not bring any significant information gain in our collection of trajectories.

To not overfit on the default seed for each game, we ran the same procedure on 5 different seeds. This resulted in 1,005 trajectories of various lengths and qualities for each game. Note that only 1 (or 5 if the game is not stochastic) of those obtain a 100% final score by following the walkthrough actions given by Jericho. We report in Appendix 6.A the normalized scores (Figure 6.4) and lengths (Figure 6.5) observed in the collection of trajectories collected for each game. The top part of Figure 6.1 illustrates the data generation procedure.

6.4.3 Sequence Definition

In this section, we describe in detail what defines ordered sequences of $\{o_t, g_t, a_t\}_{t=0}^{T}$ triples in the setting of Jericho games collected as described in Section 6.4.2.

To train an encoder-decoder architecture, trajectories are split between input and output sequences after a random number of steps. After sampling a random index $t \in [0, T-1]$ to split the trajectory, the input sequence is defined as $[o_0, g_0, a_0, o_1, ..., g_{t-1}, a_{t-1}, o_t]$ and the output sequence is defined as $[g_t, a_t, o_{t+1}]$ (also depicted in Figure 6.2). Each of these $\{o_t, g_t, a_t\}_{t=0}^{T}$ elements are represented in natural language text as described below and concatenated together to form two long text sequences: one for the input, one for the output.

$\boldsymbol{a_t}$: each intermediate action is written as returned by agents playing the game, with the addition of special token delimiters. Each action is written in text like this: "`Action: {a_t} </s>`

</s>", with {a_t} being replaced by the action taken by the agent.

$\boldsymbol{g_t}$: each goal condition is computed based on the list of intermediate rewards returned by the environment against the agent playing, depending on the strategy used among the ones described in Section 6.2.2. With the addition of special token delimiters, each goal condition is written in text like this: "GC: {g_t} </s></s>", with {g_t} being replaced by the goal condition number after being normalized between 0 and 100 (see Equation 6.4).

$\boldsymbol{o_t}$: state observations are defined by multiple state characteristics available to Jericho games. At each step, we use (i) candidate actions available, (ii) the message returned by the game engine, (iii) the description of the current room (if it is not already present in the message returned by the game engine), and (iv) the current inventory of the agent. Each observation, with the addition of special token delimiters, is written in text like this: "Actions: {cand} </s></s> State: {msg} </s></s> Description: {desc} </s></s> Inventory: {inv} </s></s>", with {cand}, {msg}, {desc} and {inv} being the list of candidate actions, the game message after taking the previous action, the description of the current room, and the inventory of the player respectively, all given by the Jericho game engine.

After realizing that game trajectories can be as long as 1000 steps and that the current definition of $\{o_t, g_t, a_t\}_{t=0}^{T}$ triples can make input sequences as long as tens of thousands of tokens, we greatly simplified the definition of input sequences. We replaced state observations o_t to be a single placeholder token "<STATE>" for all intermediate observations except the first (o_0) and current one (o_t) as depicted in Figure 6.2.

6.5 Experimental Results

Since Jericho games have long storylines, we leverage LongT5 [58], a text-to-text Transformer with a wide attention span. We use the pre-trained LongT5-base model in all experiments as the base for our encoder-decoder architecture. We then fine-tuned the model for multiple epochs on the generated trajectories from Section 6.4.2. The hyperparameter settings can be found in Appendix 6.B.

For each game, we initialize its environment with a random seed. We let the model predict the next goal condition and action at each step. The agent performs the predicted action, leading to the next observation in the environment. The model uses this observation as context for the next step. We run these steps in a cycle until we reach the end of the game and compute the final score. The game ends when the agent reaches the final state, the model generates an invalid action, or the model fails to generate an action. We repeat this process on 5 different random seeds and take the average final score. The bottom part of Figure 6.1

illustrates the training and evaluation process. Next we examine the effect of exponential tilt, the effect of different goal conditioning methods, and the effect of learning to predict the next observation o_{t+1} as part of the loss function.

6.5.1 The Effect of Exponential Tilt

To analyze the effect of exponential tilt we compare a model that doesn't have access to the maximum score of a game (but uses various amounts of exponential tilt) with one that has that information.

We fine-tuned our model with the loss function described in Equation 6.6 on all our generated trajectories split into input and output sequence pairs as described in Section 6.4.3 and depicted in Figure 6.2. The model was trained with the regular return-to-go goal condition ($g_t = \sum_{i=t}^{T} r_i$) and with $\lambda = 0.5$ for the auxiliary loss of predicting o_{t+1}. We tested the model on all games, normalized the obtained score based on the maximum human score for each game, and recorded the average across games and 5 random seeds for each game.

To measure the effect of exponential tilt, the predicted g_t were sampled according to Equation 6.3 with $\alpha = 0, 1, 10, 20$ ("Predicted GC / alpha=α" in Figure 6.3). In addition, we evaluated the model with the goal-condition being manually given ("Optimal GC" in Figure 6.3) at each step. In the first step the model is conditioned with $g_0 = 100$ (the maximum possible according to Equation 6.4) and at every step, g_t is reduced by the amount of observed reward as in Equation 6.1. This "Optimal GC" evaluation assumes we know the game's maximum score. We aim to achieve similar performance by simply predicting g_t instead of manually defining it.

We report in Figure 6.3 the normalized score averaged across games for each method of predicting g_t at different training stages of the model. As we prioritize high numerical return-to-go over their likelihood (α increasing), the model's performance is getting closer to the "Optimal GC" performance. During training the model is exposed to trajectories of various performances (detailed in Figure 6.4), so without any exponential tilt the model will output the most probable goal-condition based on what it observed during training, which is less than ideal (solid red "Predicted GC / alpha=0" line). If we slightly prioritize higher numerical values (solid yellow "Predicted GC / alpha=1" line), the performance of the model improves slightly but is still very unstable. As α increases to 10 (solid green line) and 20 (solid orange line), the performance is on par with the model that was manually given the "optimal" goal condition based on the game's maximum score. In realistic scenarios, we do not have the optimal goal condition when starting a new game. In addition, predicting the goal condition offers greater flexibility in the design of goal-conditions. We can now explore conditioning

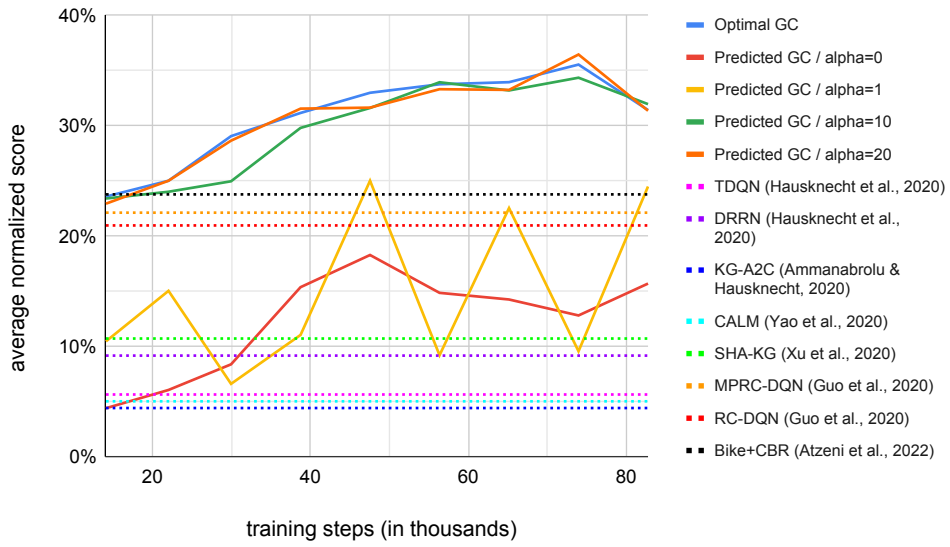


Figure 6.3 Average normalized score across different Jericho games (enchanter, sorcerer, spellbrkr, spirit, ztuu) with various amounts of exponential tilt ("*Predicted GC*" lines). We also report the performance of a model being conditioned on the optimal goal according to each game's maximum score ("*Optimal GC*" line). The average normalized score of various baselines trained on each game is depicted with dotted lines.

methods that would be impossible to define manually during run time. This is exactly what we explore in the next section.

The above results demonstrate two things: (1) the numerical value of the goal-condition has indeed an effect on the quality of the next generated answer, and (2) it is possible to recover the same performance as the "optimal" goal-conditioning by increasing the amount of exponential tilt without knowing the game's maximum score.

In addition, we show in Figure 6.3 the reported average performance of various previous works on the same set of games (dotted lines). Although not directly comparable since all previous methods were trained on each game in an online RL fashion, our offline method beats previous methods with very little training in the case of "Optimal GC" and "Predicted GC / alpha=10 & 20".

6.5.2 Our Goal Conditioning Strategies

To evaluate our goal conditioning strategies, we fine-tune 4 models with the loss function described in Equation 6.6 on all our generated trajectories split into input and output sequence pairs as described in Section 6.4.3 and depicted in Figure 6.2. We train each model with a

different goal condition (as described in Section 6.2.2) and with $\lambda = 0.5$ for the auxiliary loss of predicting o_{t+1}. We test the models on all games after 31.4k training steps and record the average score across 5 random seeds for each game.

In these experiments, we have the model generate goal-conditions because at inference time, unlike with return-to-go (RTG), we cannot compute the immediate reward (ImR) and the average return-to-go (AvgRTG), even if we know the game maximum score. To be able to manually provide the optimal immediate reward condition, we need to know at each step the maximum achievable reward among all candidate actions, which is infeasible in practice. Similarly in order to provide the optimal average RTG condition, we need to know the number of steps remaining after each state, which is also infeasible in practice. Fortunately, our model can generate these two goal-conditions (as well as any others), while leveraging the exponential tilt for producing better trajectories. Therefore, all models in these experiments are evaluated by sampling g_t according to Equation 6.3 with $\alpha = 10$.

GC =	**Return-To-Go**			**Immediate Reward**			**Final Score**			**Avg. Return-To-Go**		
	avg.	stdev.	max.	avg.	stdev.	max.	avg.	stdev.	max.	avg.	stdev.	max.
enchanter	45.00	0.00	45.00	**235.00**	0.00	235.00	231.00	56.69	280.00	175.00	0.00	175.00
sorcerer	124.00	90.63	235.00	112.00	75.93	205.00	124.00	90.63	235.00	**132.00**	100.43	255.00
spellbrkr	31.00	7.35	40.00	31.00	7.35	40.00	25.00	0.00	25.00	**40.00**	0.00	40.00
spirit	18.40	3.88	26.00	22.40	8.69	38.00	**26.00**	15.35	56.00	5.60	0.80	6.00
ztuu	73.00	7.48	85.00	**75.00**	9.49	90.00	**75.00**	9.49	90.00	**75.00**	9.49	90.00
Norm. Avg.	26.74%		35.94%	36.36%		45.90%	**36.62%**		50.02%	33.66%		42.84%

Table 6.2 Average and the maximum score for each game across 5 random seeds for each goal condition (GC) variation. On the bottom line, scores are normalized according to the maximum human score (enchanter: 400; sorcerer: 400; spellbrkr: 280; spirit: 250; ztuu: 100) and averaged across games.

Table 6.2 reports the average score, standard deviation, and maximum score obtained on each game across 5 random seeds for all goal-conditioning methods. The bottom line reports the normalized average score based on the maximum human score for each game. On average, the classical return-to-go goal conditioning method yields weaker performance than all other variants. The average performance of immediate reward (36.36%) and final score (36.62%) conditioning is even higher than the maximum average score of return-to-go conditioning (35.94%). It is also interesting to note that the best-performing method on average is the "Final Score" conditioning strategy with 50.02% average max score. As mentioned in Section 6.2.2, this setting is closer to the traditional RL paradigm in which we often define rewards based on the final performance of an agent, which is only based on whether it won or lost a game.

Overall, these results show that the classical return-to-go conditioning method performs poorly in all environments. However, the winning goal conditioning strategy depends on the

game which can vary between ImR, FinS, or AvgRTG but not RTG. These results further motivate the advantages of generating goal-conditions that cannot be computed at runtime such as ImR and AvgRTG.

6.5.3 Predicting the Next Observation

Here we analyze the effect of predicting the next observation o_{t+1} as part of the loss function. Therefore, we fine-tuned another 4 models, each with a different goal condition similar to the above section, but with the loss function described in Equation 6.6 with $\lambda = 0.0$ for the auxiliary loss of predicting o_{t+1}. We tested the models on all games after 31.4k training steps and recorded the average score across 5 random seeds for each game. To compare the effect of the auxiliary loss, we averaged the scores again across all goal-conditioning methods.

	$\lambda = 0.0$			$\lambda = 0.5$		
	avg.	stdev.	max.	avg.	stdev.	max.
enchanter (max=400)	138.75	55.83	235.00	**171.50**	81.86	280.00
sorcerer (max=400)	79.50	39.43	130.00	**123.00**	90.12	255.00
spellbrkr (max=280)	25.75	3.27	40.00	**31.75**	7.46	40.00
spirit (max=250)	10.15	9.75	36.00	**18.10**	11.87	56.00
ztuu (max=100)	55.75	33.10	90.00	**74.50**	9.07	90.00
Normalized Average	24.71%		41.99%	**33.34%**		52.09%

Table 6.3 Average and the maximum score for each game across 5 random seeds and 4 goal conditioning methods, with ($\lambda = 0.5$) and without ($\lambda = 0.0$) the auxiliary loss on the prediction of the next observation o_{t+1}. Scores are then normalized according to the maximum human score ('*max=#*') and averaged across games on the bottom line.

Table 6.3 reports the average score, standard deviation, and maximum score obtained on each game over 20 runs (5 random seeds $\times$ 4 goal-conditioning methods) for models trained with ($\lambda = 0.5$) and without ($\lambda = 0.0$) the auxiliary loss on the predicted next observation o_{t+1}. The bottom line reports the normalized average score based on the maximum human score for each game.

In all games, models trained to predict the next observation o_{t+1} resulting from the predicted action a_t and goal-condition g_t perform better than models trained to only predict the goal-condition g_t and next action a_t. Overall, these results show that our proposed model-based reward-conditioned policy (MB-RCP) learning objective yields stronger performance than the classical reward-conditioned policy (RCP) objective.

6.6 Conclusion

In this work, we have proposed Long-Context Language Decision Transformers (LLDTs) as an offline reinforcement learning method for interactive text environments, and we have performed experiments using the challenging text-based games of Jericho. LLDTs are built from pre-trained LLMs followed by training on multiple games simultaneously to predict: the trajectory goal condition, the next action, and the next observation. We have shown that by using exponential tilt, LLDT-based agents get much better performance than otherwise. In fact, the model obtains similar performance as if it was conditioned on the optimal goal, despite the fact that in most realistic scenarios, we do not have access to that optimal goal condition. We have also explored different conditioning methods and observed that the traditional return-to-go was the weakest strategy. Finally we have seen that training the model to predict the next observation as an auxiliary loss improves performance. As future work, we plan on extending this framework to multiple and more diverse games and environments. We hope this work can provide a missing piece to the substantial advances in the application of large language models in the context of real-world interactive task-oriented dialogues.

6.A Trajectories Statistics

In this section, we report the normalized scores (Figure 6.4) and lengths (Figure 6.5) observed in the collection of trajectories collected for each game as described in Section 6.4.2.

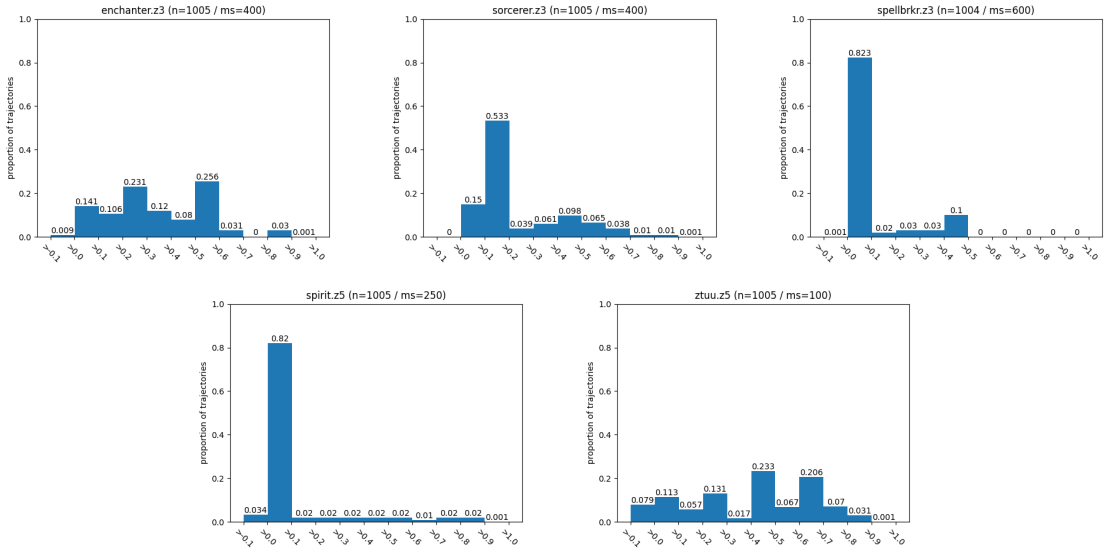


Figure 6.4 Proportion of trajectory normalized scores for a selection of games. In each subfigure title, n is the number of trajectories and ms is the maximum score. The X-axis is the normalized score the trajectory achieves. The Y-axis is the proportion of trajectories finishing with that score.

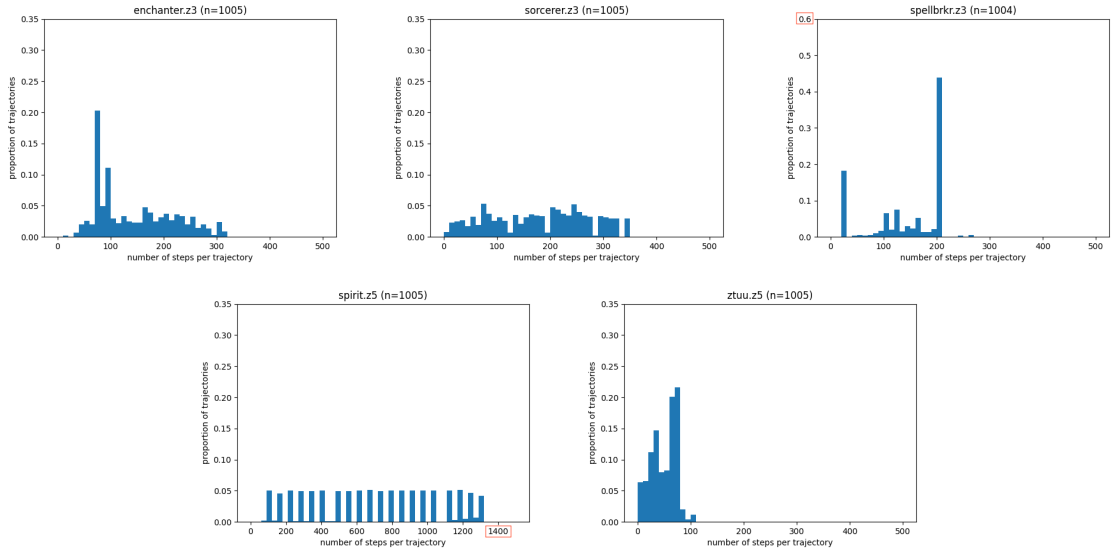


Figure 6.5 Proportion of trajectory lengths for a selection of games. In each sub-figure title, n is the number of trajectories. The X-axis is the number of steps in a trajectory. The Y-axis is the proportion of trajectories of that length.

6.B Hyperparameters

optimizer	Adafactor
learning rate	1e-4
precision	32
batch size	16
max input length	4096
max output length	1024
base model	LongT5-base with Transient Global Attention

CHAPTER 7 GENERAL DISCUSSION

This chapter first summarizes the three contributions of this thesis that evaluated the reasoning ability of Transformer Language Models (TLMs) in various settings ranging from proof generation, entity type abstraction, and Interactive Fiction (IF) games. Second, this chapter discusses the limitation of each contribution and of the whole thesis more broadly.

7.1 Summary of Works

In Chapter 4 a Transformer decoder architecture is trained from scratch on the CLUTRR dataset [36] after constructing a train/test data split that tests for compositional generalization based on the presence of (entity, relation, entity) triples and their combinations over multiple proof steps. Multiple versions of the data are created with either no proof, short proof, short proof reversed, long proof, and long proof reversed before answering questions. To test compositional generalization in both interpolation and extrapolation settings, the models are trained on CLUTRR examples of reasoning difficulty 2, 4, 6, and tested on examples from levels 3, 5, 7, 8, 9, and 10. Experiments reveal that models trained to generate exhaustive proofs tend to be better reasoners than the ones trained with shorter ones. In addition, experiments show that backward-chaining proofs are easier to use albeit being harder to generate than forward-chaining ones. Eventually, in all experimental setups TLMs suffer from length generalization issues, thus models trained to generate only the final answer (shorter output sequence) tend to be more robust than models trained to generate intermediate proof steps before the answer (longer output sequences). Additional experiments indicate that pre-trained models are also better reasoners and more robust to length generalization than models trained from scratch.

In Chapter 5 a T5-small model [52] is fine-tuned on different question-answering datasets after being given entity-type information. The spacy named entity recognizer[1] is used to automatically label documents with named entities. This information is given to the encoder-decoder model through five different model variants: embedding sum, embedding concatenation, encoding sum, encoding concatenation, or decoder auxiliary loss. Thorough experiment analysis shows that entity-type abstraction does help TLMs to reason at unseen complexities (compositionally generalize), especially in the context of synthetically designed tasks that rely on a symbolic structure beneath them. In particular, for the same CLUTRR [36] setup as in Chapter 4, entity type abstraction significantly improves the compositional generalization

[1] https://spacy.io/models/en

of the pre-trained model more than proof generation did in the first contribution. Models trained with the encoding sum and decoder auxiliary loss tend to show stronger generalization capacities than models trained with the other three augmentation methods (emb-cat, emb-sum, enc-cat). Nonetheless, entity-type abstraction has little effect on natural language tasks composed of human-generated text. Additional analysis suggests that entity-type abstraction is only beneficial in tasks where (1) good quality abstraction labels are available and (2) train/test data is split according to the reasoning complexity of each example, which is often not the case for human-generated language tasks.

Chapter 6 introduces an extension of Decision Transformers (DTs) for interactive text environments. A LongT5-base model [58] is fine-tuned with the language modeling objective on sequences of interactions created automatically by mixing a golden path and some random exploration on Jericho text games [40]. An offline dataset of trajectories is created by ordering state observations o_t, some action quality metric that we called 'goal condition' g_t, and the action taken a_t. To reduce the length of these sequences so that they fit in the context window of the LongT5 model some observations o_t in the middle of the trajectory are replaced by a special single token. Experimental analysis indicates that pre-trained TLMs can learn to map performance metrics to actions, thus allowing greater control over their behavior and strong long-term results when using the exponential tilt method. In addition, this work proposes multiple 'goal condition' definitions in addition to the traditional return-to-go (RTG) used in DTs. These include immediate reward, final score, and average return-to-go. When tested across several environments, these new conditioning methods appear to perform better than RTG. Eventually, further experiments demonstrate that TLMs behave more optimally if they are also trained to predict the effect of their actions, which is similar to world modeling. This approach is particularly advantageous because it is easy to implement with pre-trained language models as the observations from the environment are in textual form.

7.2 Limitations

One limitation of this research is that most experiments were conducted with relatively small model sizes compared to what big industrial labs can use. Recent work showed that by scaling up model sizes and data quantities, TLMs can (or at least pretend they can) perform impressive tasks (such as reasoning) simply by being prompted in specific ways [50, 218]. In an effort to mitigate this limitation, most of the work presented in this thesis was done in collaboration with an industrial research lab that provided a lot of computing resources.

Another important limitation of this research is that, by nature, Transformer language models are statistical machines: they generate sentences by sampling tokens one after the other based

on some learned distribution. As a result, it is challenging for them to have deterministic behavior and perform systematic logical operations. In cases when we know how to solve a small problem systematically and have a program for that, it may be more natural to use that deterministic program rather than a language model.

At very large scales (models with tens of billions of parameters), TLMs seem to be able to perform logical reasoning when prompted with a few examples or key phrases like "*let's think step-by-step*". However, there are no verification mechanisms and the model is still free to generate according to its belief of the most likely output. It is the human's responsibility to verify the model's output and fact-check it. This is a potentially dangerous limitation: if TLMs can produce convincing hallucinations they can be used to create disinformation or propaganda at scale. In the end, it all comes down to interpretability and transparency: understanding how and why a model generated a particular output. However, as competition increases in the field of generative AI, large corporations are not incentivized to be transparent at the risk of losing their technological advantage.

Another limitation of large language models, as briefly mentioned above, is their enormous computing resource requirements to train them. For example, GPT-3 has 175 billion parameters and was estimated to cost about $12 million to train on public cloud GPU/TPU cost models. Such high costs can concentrate the development of LLMs to the richest organizations of the world, and restrict their development in places with limiting resources such as academia, developing countries, and smaller enterprises. Moreover, training LLMs also consumes a large amount of energy and contributes to carbon emissions.

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS

As a starting point to address the previously mentioned limitations, this chapter discusses some promising research directions to improve upon this research.

Let's first discuss some interesting research directions to address the limitation that large language models (LLMs) are statistical machines and hence not always appropriate in situations where a deterministic and reproducible behavior is required. Some of the earlier work to consider this idea is Cobbe et al. [219] which used the python interpreter to evaluate the result of simple math computations. The language model is responsible for generating special tokens that trigger a call to the python interpreter during generation. For instance in the sequence "*Her sister gave her 20+10=*«`20 + 10 = 30`»" the LLM generates all tokens except the "`30`" that is returned by the python command "`20 + 10`" inside the "« »" brackets.

Going one step further, Gao et al. [220] propose Program-aided Language Models (PAL). They train an LLM to generate python code for individual reasoning steps in mathematical reasoning. In their case, the LLM produces both natural language steps (such as "*Roger bought 2 cans of 3 tennis balls each*") and their corresponding meaning in simple python code ("`bought_balls = 2 * 3`"). The model then generates the mathematical formula to answer the question ("`answer = tennis_balls + bought_balls`") and a python interpreter runs the program to return the numerical answer. Importantly, the LLM is never responsible for computing anything, it is only responsible for generating a plan or executable program that can then be run by a reliable program executor. Similar work by Chen et al. [221] introduced program-of-thoughts (PoT) prompting which also relies on generating natural language and code to answer a numerical reasoning task.

This idea is gaining traction in the research community and keeps evolving at a fast pace. Very recently Schick et al. [222] proposed Toolformer, an LLM that learns to use external APIs to accomplish complex tasks beyond mathematical reasoning, such as machine translation, question answering, and Wikipedia searches. Similarly to previous works, their model generates natural language responses with the addition of trigger tokens that are used to call an external API at run time. To gain a feel for how fast things are moving in this space, while I was writing this paragraph, OpenAI announced *ChatGPT pluggins*[1], a library of tools for ChatGPT to access up-to-date information, run computations, or use third-party services.

As exciting as the field of generative AI is, one must take a pause in this environment of "*AI race*" to make sure we do not forget about the aforementioned limitations of LLMs and

[1] https://openai.com/blog/chatgpt-plugins

consider ethical questions associated with them. It remains the case that these models are prohibitively large and expensive to train, and can generate convincing facts while hallucinating. Some interesting research directions to tackle these challenges include developing more efficient and sustainable methods for training LLMs (such as model distillation to reduce their number of parameters) and investigating the ethical and societal implications of deploying LLMs in the world.

Another promising research direction to limit LLM hallucinations is to consider retrieval-based language models such as EMAT [117], REALM [119], RAG [120] or FiD [121]. In these works the model generally generates answers in a two-step procedure, first, a retrieval system returns the most relevant documents from a trusted knowledge source, and second, an LLM generates a response based on the returned documents. This method has the tendency to ground the language model in some truthful context and mitigates hallucinations. A recent implementation of this method is the new Bing search engine[2].

To conclude, this thesis evaluated Transformer Language Models in various settings and proposed novel extensions to improve their reasoning abilities and control their behavior as it is a crucial component that will take conversational interfaces to the next level as we are starting to see in 2023.

[2]https://www.bing.com/new

REFERENCES

[1] P. Clark, O. Tafjord, and K. Richardson, "Transformers as soft reasoners over language," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[2] A. Piché *et al.*, "A probabilistic perspective on reinforcement learning via supervised learning," in *ICLR Workshop on Generalizable Policy Learning in Physical World*, 2022.

[3] L. Chen *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[4] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *International Conference on Machine learning (ICML)*, 2007.

[5] P. Dayan and G. E. Hinton, "Using expectation-maximization for reinforcement learning," *Neural Computation*, 1997.

[6] A. Kumar, X. B. Peng, and S. Levine, "Reward-conditioned policies," *arXiv preprint arXiv:1912.13465*, 2019.

[7] K.-H. Lee *et al.*, "Multi-game decision transformers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[8] A. Piché *et al.*, "Probabilistic planning with sequential monte carlo methods," in *International Conference on Learning Representations (ICLR)*, 2019.

[9] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[10] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[11] A. M. Turing, "Computing machinery and intelligence," *Mind*, 1950.

[12] J. Weizenbaum, "Eliza - a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, 1966.

[13] E. Levin and M. Fleisher, "Accelerated learning in layered neural networks," *Complex systems*, 1988.

[14] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, 1989.

[15] T. Mikolov *et al.*, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[16] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[17] J. Devlin *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *North American Association for Computational Linguistics (NAACL)*, 2019.

[18] J. L. Elman, "Finding structure in time," *Cognitive science*, 1990.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[20] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[22] A. Radford *et al.*, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, 1986.

[24] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI*, 2019.

[25] Z. Dai *et al.*, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Association for Computational Linguistics (ACL)*, 2019.

[26] Z. Yang *et al.*, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[27] A. Liška, G. Kruszewski, and M. Baroni, "Memorize or generalize? searching for a compositional rnn in a haystack," *arXiv preprint arXiv:1802.06467*, 2018.

[28] I. Dasgupta *et al.*, "Analyzing machine-learned representations: A natural language case study," *arXiv preprint arXiv:1909.05885*, 2019.

[29] E. Goodwin, K. Sinha, and T. J. O'Donnell, "Probing linguistic systematicity," in *Association for Computational Linguistics (ACL)*, 2020.

[30] B. M. Lake, "Compositional generalization through meta sequence-to-sequence learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[31] M. Baroni, "Linguistic generalization and compositionality in modern artificial neural networks," *Philosophical Transactions of the Royal Society*, 2020.

[32] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[33] N. Chomsky, "Logical structures in language," *American Documentation*, 1957.

[34] R. Montague, "Universal grammar," *Theoria*, 1970.

[35] D. Bahdanau *et al.*, "Systematic generalization: What is required and can it be learned?" in *International Conference on Learning Representations (ICLR)*, 2019.

[36] K. Sinha *et al.*, "CLUTRR: A diagnostic benchmark for inductive reasoning from text," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[37] O. Tafjord, B. Dalvi, and P. Clark, "ProofWriter: Generating implications, proofs, and abductive statements over natural language," in *Association for Computational Linguistics (ACL)*, 2021.

[38] Z. Yang *et al.*, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," *arXiv preprint arXiv:1809.09600*, 2018.

[39] S. Reddy, D. Chen, and C. D. Manning, "Coqa: A conversational question answering challenge," *Transactions of the Association for Computational Linguistics (TACL)*, 2019.

[40] M. Hausknecht *et al.*, "Interactive fiction games: A colossal adventure," in *Conference on Artificial Intelligence (AAAI)*, 2020.

[41] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, 1982.

[42] D. E. Rumelhart *et al.*, "Sequential thought processes in pdp models," *Parallel distributed processing: explorations in the microstructures of cognition*, 1986.

[43] M. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Cognitive Science Society*, 1986.

[44] Y. Bengio, P. Frasconi, and P. Simard, "The problem of learning long-term dependencies in recurrent networks," in *IEEE International Conference on Neural Networks*, 1993.

[45] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, 1994.

[46] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[47] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems (NeurIPS)*, 2014.

[48] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE transactions on pattern analysis and machine intelligence (PAMI)*, 1983.

[49] U. Khandelwal *et al.*, "Sharp nearby, fuzzy far away: How neural language models use context," in *Association for Computational Linguistics (ACL)*, 2018.

[50] T. B. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeruIPS)*, 2020.

[51] Y. Liu *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[52] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research (JMLR)*, 2020.

[53] R. Bommasani *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[54] M. E. Peters *et al.*, "Knowledge enhanced contextual word representations," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[55] I. Singh, G. Singh, and A. Modi, "Pre-trained language models as prior knowledge for playing text-based games," *arXiv preprint arXiv:2107.08408*, 2021.

[56] Y. Levine *et al.*, "SenseBERT: Driving some sense into BERT," in *Association for Computational Linguistics (ACL)*, 2020.

[57] M. Lewis *et al.*, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Association for Computational Linguistics (ACL)*, 2020.

[58] M. Guo *et al.*, "LongT5: Efficient text-to-text transformer for long sequences," in *North American Association for Computational Linguistics (NAACL)*, 2022.

[59] A. Chowdhery *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[60] I. Tenney, D. Das, and E. Pavlick, "BERT rediscovers the classical NLP pipeline," in *Association for Computational Linguistics (ACL)*, 2019.

[61] K. Clark *et al.*, "What does BERT look at? an analysis of BERT's attention," in *ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2019.

[62] F. Petroni *et al.*, "Language models as knowledge bases?" in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[63] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 1st ed. MIT press, 1998.

[64] S. Emmons *et al.*, "Rvs: What is essential for offline RL via supervised learning?" in *International Conference on Learning Representations (ICLR)*, 2022.

[65] S. Levine *et al.*, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[66] J. A. Fodor and Z. W. Pylyshyn, "Connectionism and cognitive architecture: A critical analysis," *Cognition*, 1988.

[67] B. M. Lake and M. Baroni, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks," in *International Conference on Machine Learning (ICML)*, 2018.

[68] E. K. Miller and J. D. Cohen, "An integrative theory of prefrontal cortex function," *Annual review of neuroscience*, 2001.

[69] J. Russin *et al.*, "Compositional generalization in a deep seq2seq model by separating syntax and semantics," *arXiv preprint arXiv:1904.09708*, 2019.

[70] J. Andreas *et al.*, "Neural module networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[71] E. Perez *et al.*, "Film: Visual reasoning with a general conditioning layer," in *Conference on Artificial Intelligence (AAAI)*, 2018.

[72] D. Keysers *et al.*, "Measuring compositional generalization: A comprehensive method on realistic data," in *International Conference on Learning Representations (ICLR)*, 2020.

[73] J. Andreas, "Good-enough compositional data augmentation," *arXiv preprint arXiv:1904.09545*, 2019.

[74] H. Zheng and M. Lapata, "Compositional generalization via semantic tagging," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[75] P. Price, "Evaluation of spoken language systems: The atis domain," in *Speech and Natural Language Workshop*, 1990.

[76] D. A. Dahl *et al.*, "Expanding the scope of the atis task: The atis-3 corpus," in *Workshop on Human Language Technology*, 1994.

[77] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Conference on Artificial Intelligence (AAAI)*, 1996.

[78] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[79] I. Oren *et al.*, "Improving compositional generalization in semantic parsing," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[80] C. Finegan-Dollak *et al.*, "Improving text-to-SQL evaluation methodology," in *Association for Computational Linguistics (ACL)*, 2018.

[81] S. Iyer *et al.*, "Learning a neural semantic parser from user feedback," in *Association for Computational Linguistics (ACL)*, 2017.

[82] D. Dua *et al.*, "Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs," *arXiv preprint arXiv:1903.00161*, 2019.

[83] T. Wolfson *et al.*, "Break it down: A question understanding benchmark," *Transactions of the Association for Computational Linguistics (TACL)*, 2020.

[84] M. E. Peters *et al.*, "Deep contextualized word representations," in *North American Association for Computational Linguistics (NAACL)*, 2018.

[85] L. Bergen, T. O' Donnell, and D. Bahdanau, "Systematic generalization with edge transformers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[86] N. Kim and T. Linzen, "Cogs: A compositional generalization challenge based on semantic interpretation," *arXiv preprint arXiv:2010.05465*, 2020.

[87] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *North American Association for Computational Linguistics (NAACL)*, 2018.

[88] M. Dehghani *et al.*, "Universal transformers," *arXiv preprint arXiv:1807.03819*, 2018.

[89] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[90] P. Hitzler *et al.*, "Neuro-symbolic approaches in artificial intelligence," *National Science Review*, 2022.

[91] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 3rd ed. Pearson, 2010.

[92] L. Weber *et al.*, "NLProlog: Reasoning with weak unification for question answering in natural language," in *Association for Computational Linguistics (ACL)*, 2019.

[93] J. Welbl, P. Stenetorp, and S. Riedel, "Constructing datasets for multi-hop reading comprehension across documents," *Transactions of the Association for Computational Linguistics (TACL)*, 2018.

[94] P. Minervini *et al.*, "Differentiable reasoning on large knowledge bases and natural language," in *Conference on Artificial Intelligence (AAAI)*, 2020.

[95] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," in *IEEE Transactions on Big Data*, 2019.

[96] G. A. Miller, "Wordnet: A lexical database for english," *Communications of the Association for Computing Machinery (ACM)*, 1995.

[97] K. Bollacker *et al.*, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Association for Computing Machinery (ACM) SIGMOD International Conference on Management of Data*, 2008.

[98] P. Minervini *et al.*, "Learning reasoning strategies in end-to-end differentiable proving," in *International Conference on Machine Learning (ICML)*, 2020.

[99] G. Bouchard, S. Singh, and T. Trouillon, "On approximate reasoning capabilities of low-rank vector spaces." in *Conference on Artificial Intelligence (AAAI)*, 2015.

[100] C. Kemp *et al.*, "Learning systems of concepts with an infinite relational model," in *Conference on Artificial Intelligence (AAAI)*, 2006.

[101] P. Veličković *et al.*, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[102] R. Das *et al.*, "Building dynamic knowledge graphs from text using machine reading comprehension," *arXiv preprint arXiv:1810.05682*, 2018.

[103] D. Chen *et al.*, "Reading wikipedia to answer open-domain questions," *arXiv preprint arXiv:1704.00051*, 2017.

[104] B. D. Mishra *et al.*, "Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension," *arXiv preprint arXiv:1805.06975*, 2018.

[105] Y. Fang *et al.*, "Hierarchical graph network for multi-hop question answering," *arXiv preprint arXiv:1911.03631*, 2019.

[106] Z. Zhang *et al.*, "ERNIE: Enhanced language representation with informative entities," in *Association for Computational Linguistics (ACL)*, 2019.

[107] P. Ferragina and U. Scaiella, "Tagme: On-the-fly annotation of short text fragments (by wikipedia entities)," in *ACM International Conference on Information and Knowledge Management*, 2010.

[108] A. Bordes *et al.*, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[109] A. Wang *et al.*, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[110] T. Févry *et al.*, "Entities as experts: Sparse memory access with entity supervision," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[111] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Prentice Hall, 2009.

[112] M. Joshi *et al.*, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[113] J. Berant *et al.*, "Semantic parsing on freebase from question-answer pairs," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[114] P. Verga *et al.*, "Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge," *arXiv preprint arXiv:2007.00849*, 2020.

[115] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *International Conference on Learning Representations (ICLR)*, 2015.

[116] S. Sukhbaatar *et al.*, "End-to-end memory networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[117] Y. Wu *et al.*, "An efficient memory-augmented transformer for knowledge-intensive NLP tasks," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

[118] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?" in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[119] K. Guu *et al.*, "Retrieval augmented language model pre-training," in *International Conference on Machine Learning (ICML)*, 2020.

[120] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[121] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," in *European Association for Computational Linguistics (EACL)*, 2021.

[122] P. Lewis *et al.*, "PAQ: 65 million probably-asked questions and what you can do with them," *Transactions of the Association for Computational Linguistics (TACL)*, 2021.

[123] E. Dinan *et al.*, "Wizard of wikipedia: Knowledge-powered conversational agents," in *International Conference on Learning Representations (ICLR)*, 2019.

[124] A. Fan *et al.*, "Eli5: Long form question answering," *arXiv preprint arXiv:1907.09190*, 2019.

[125] A. Gupta and J. Berant, "Gmat: Global memory augmentation for transformers," *arXiv preprint arXiv:2006.03274*, 2020.

[126] M. S. Burtsev *et al.*, "Memory transformer," *arXiv preprint arXiv:2006.11527*, 2020.

[127] Q. Wu *et al.*, "Memformer: A memory-augmented transformer for sequence modeling," in *Association for Computational Linguistics (ACL)*, 2022.

[128] A. Fan *et al.*, "Addressing some limitations of transformers with feedback memory," *arXiv preprint arXiv:2002.09402*, 2020.

[129] Y. Ji *et al.*, "Dynamic entity representations in neural language models," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[130] C. Rosset *et al.*, "Knowledge-aware language model pretraining," *arXiv preprint arXiv:2007.00655*, 2020.

[131] T. Kwiatkowski *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics (TACL)*, 2019.

[132] S. Swayamdipta *et al.*, "Syntactic scaffolds for semantic structures," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[133] A. Eriguchi, Y. Tsuruoka, and K. Cho, "Learning to parse and translate improves neural machine translation," in *Association for Computational Linguistics (ACL)*, 2017.

[134] M. Nădejde *et al.*, "Predicting target language CCG supertags improves neural machine translation," in *Conference on Machine Translation*, 2017.

[135] M. Palmer, D. Gildea, and P. Kingsbury, "The Proposition Bank: An annotated corpus of semantic roles," *Computational Linguistics*, 2005.

[136] D. Sundararaman *et al.*, "Syntax-infused transformer and bert models for machine translation and natural language understanding," *arXiv preprint arXiv:1911.06156*, 2019.

[137] A. Raganato, J. Camacho-Collados, and R. Navigli, "Word sense disambiguation: A unified evaluation framework and empirical comparison," in *European Association for Computational Linguistics (EACL)*, 2017.

[138] N. S. Moosavi *et al.*, "Improving robustness by augmenting training sentences with predicate-argument structures," *arXiv preprint arXiv:2010.12510*, 2020.

[139] P. Shi and J. Lin, "Simple bert models for relation extraction and semantic role labeling," *arXiv preprint arXiv:1904.05255*, 2019.

[140] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *North American Association for Computational Linguistics (NAACL)*, 2018.

[141] R. Zellers *et al.*, "SWAG: A large-scale adversarial dataset for grounded commonsense inference," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[142] I. Porada *et al.*, "Modeling event plausibility with consistent conceptual abstraction," in *North American Association for Computational Linguistics (NAACL)*, 2021.

[143] H. Bai *et al.*, "Better language model with hypernym class prediction," in *Association for Computational Linguistics (ACL)*, 2022.

[144] P. F. Brown *et al.*, "Class-based n-gram models of natural language," *Computational Linguistics*, 1992.

[145] S. Merity *et al.*, "Pointer sentinel mixture models," in *International Conference on Learning Representations*, 2017.

[146] A. Lazaridou *et al.*, "Mind the gap: Assessing temporal generalization in neural language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[147] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, 1992.

[148] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *nature*, 2015.

[149] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[150] J. He *et al.*, "Deep reinforcement learning with a natural language action space," in *Association for Computational Linguistics (ACL)*, 2016.

[151] M.-A. Côté *et al.*, "Textworld: A learning environment for text-based games," in *IJCAI Workshop on Computer Games*, 2018.

[152] X. Yuan *et al.*, "Counting to explore and generalize in text-based games," in *Exploration in Reinforcement Learning Workshop at ICML*, 2018.

[153] A. Trischler, M.-A. Côté, and P. Lima, "First textworld problems, the competition: Using text-based games to advance capabilities of ai agents," *Microsoft Research Blog*, 2019.

[154] X. Yuan *et al.*, "Interactive language learning by question answering," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[155] M. Hausknecht *et al.*, "Nail: A general interactive fiction agent," *arXiv preprint arXiv:1902.04259*, 2019.

[156] P. Ammanabrolu and M. Riedl, "Playing text-adventure games with graph-based deep reinforcement learning," in *North American Association for Computational Linguistics (NAACL)*, 2019.

[157] P. Ammanabrolu and M. Hausknecht, "Graph constrained reinforcement learning for natural language action spaces," in *International Conference on Learning Representations (ICLR)*, 2020.

[158] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016.

[159] Y. Xu *et al.*, "Deep reinforcement learning with stacked hierarchical attention for text-based games," in *Advances in Neural Information Processing Systems*, 2020.

[160] S. Yao *et al.*, "Keep calm and explore: Language models for action generation in text-based games," *arXiv preprint arXiv:2010.02903*, 2020.

[161] V. Sanh *et al.*, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[162] J. Schmidhuber, "Reinforcement learning upside down: Don't predict rewards–just map them to actions," *arXiv preprint arXiv:1912.02875*, 2019.

[163] D. Ghosh, A. Gupta, and S. Levine, "Learning actionable representations with goal conditioned policies," in *International Conference on Learning Representations (ICLR)*, 2018.

[164] Y. Ding *et al.*, "Goal-conditioned imitation learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[165] B. Eysenbach *et al.*, "Rewriting history with inverse rl: Hindsight inference for policy improvement," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[166] K. Paster, S. A. McIlraith, and J. Ba, "Planning from pixels using inverse dynamics models," in *International Conference on Learning Representations (ICLR)*, 2021.

[167] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[168] S. Yao *et al.*, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[169] A. Creswell, M. Shanahan, and I. Higgins, "Selection-inference: Exploiting large language models for interpretable logical reasoning," in *International Conference on Learning Representations (ICLR)*, 2023.

[170] H. Zhou *et al.*, "Teaching algorithmic reasoning via in-context learning," *arXiv preprint arXiv:2211.09066*, 2022.

[171] I. Levy, B. Bogin, and J. Berant, "Diverse demonstrations improve in-context compositional generalization," *arXiv preprint arXiv:2212.06800*, 2022.

[172] E. Zelikman *et al.*, "STar: Bootstrapping reasoning with reasoning," in *Advances in Neural Information Processing Systems (NeurIPS*, 2022.

[173] A. Talmor *et al.*, "Commonsenseqa: A question answering challenge targeting commonsense knowledge," *arXiv preprint arXiv:1811.00937*, 2018.

[174] Y. Fu *et al.*, "Complexity-based prompting for multi-step reasoning," in *International Conference on Learning Representations (ICLR)*, 2023.

[175] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[176] S. Saha *et al.*, "PRover: Proof generation for interpretable reasoning over rules," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[177] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *arXiv preprint arXiv:2009.03393*, 2020.

[178] S. M. Kazemi *et al.*, "Lambada: Backward chaining for automated reasoning in natural language," *arXiv preprint arXiv:2212.13894*, 2022.

[179] K. Yang, J. Deng, and D. Chen, "Generating natural language proofs with verifier-guided search," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

[180] S. Welleck *et al.*, "Naturalprover: Grounded mathematical proof generation with language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[181] G. Lample *et al.*, "Hypertree proof search for neural theorem proving," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[182] A. Q. Jiang *et al.*, "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs," in *International Conference on Learning Representations (ICLR)*, 2023.

[183] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[184] M. Peters *et al.*, "Dissecting contextual word embeddings: Architecture and representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[185] Y. Goldberg, "Assessing bert's syntactic abilities," *arXiv preprint arXiv:1901.05287*, 2019.

[186] I. Tenney *et al.*, "What do you learn from context? probing for sentence structure in contextualized word representations," in *International Conference on Learning Representations (ICLR)*, 2019.

[187] D. Bahdanau *et al.*, "CLOSURE: Assessing systematic generalization of CLEVR models," *arXiv preprint arXiv:1912.05783*, 2019.

[188] T. Sekiyama, A. Imanishi, and K. Suenaga, "Towards proof synthesis guided by neural machine translation for intuitionistic propositional logic," *arXiv preprint arXiv:1706.06462*, 2017.

[189] P. J. Liu *et al.*, "Generating wikipedia by summarizing long sequences," in *International Conference on Learning Representations (ICLR)*, 2018.

[190] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data (extended abstract)," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[191] Y. Dubois *et al.*, "Location attention for extrapolation to longer sequences," in *Association for Computational Linguistics (ACL)*, 2020.

[192] P. Henderson *et al.*, "Ethical challenges in data-driven dialogue systems," in *Conference on AI, Ethics, and Society (AAAI/ACM)*, 2018.

[193] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[194] L. De Raedt, A. Kimmig, and H. Toivonen, "Problog: A probabilistic prolog and its application in link discovery." in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[195] D. Fierens *et al.*, "Inference and learning in probabilistic logic programs using weighted boolean formulas," *Theory and Practice of Logic Programming*, 2015.

[196] N. Gontier *et al.*, "Measuring systematic generalization in neural proof generation with transformers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[197] J. Hewitt and C. D. Manning, "A structural probe for finding syntax in word representations," in *North American Association for Computational Linguistics (NAACL)*, 2019.

[198] N. F. Liu *et al.*, "Linguistic knowledge and transferability of contextual representations," in *North American Association for Computational Linguistics (NAACL)*, 2019.

[199] D. Sachan *et al.*, "Do syntax trees help pre-trained transformers extract information?" in *European Association for Computational Linguistics (EACL)*, 2021.

[200] M. Gardner *et al.*, "Allennlp: A deep semantic natural language processing platform," *arXiv preprint arXiv:1803.07640*, 2017.

[201] K. Nishida *et al.*, "Answering while summarizing: Multi-task learning for multi-hop QA with evidence extraction," in *Association for Computational Linguistics (ACL)*, 2019.

[202] X.-Y. Li, W.-J. Lei, and Y.-B. Yang, "From easy to hard: Two-stage selector and reader for multi-hop question answering," *arXiv preprint arXiv:2205.11729*, 2022.

[203] M. Yatskar, "A qualitative comparison of coqa, squad 2.0 and quac," *arXiv preprint arXiv:1809.10735*, 2018.

[204] H.-Y. Huang, E. Choi, and W.-t. Yih, "FlowQA: Grasping flow in history for conversational machine comprehension," in *International Conference on Learning Representations (ICLR)*, 2019.

[205] Y. Ju *et al.*, "Technical report on conversational question answering," *arXiv preprint arXiv:1909.10772*, 2019.

[206] K. Murray and D. Chiang, "Correcting length bias in neural machine translation," in *Conference on Machine Translation*, 2018.

[207] C. Anil *et al.*, "Exploring length generalization in large language models," *arXiv preprint arXiv:2207.04901*, 2022.

[208] O. Press, N. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," in *International Conference on Learning Representations (ICLR)*, 2022.

[209] V. Shwartz, R. Rudinger, and O. Tafjord, ""you are grounded!": Latent name artifacts in pre-trained language models," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[210] E. M. Bender *et al.*, "On the dangers of stochastic parrots: Can language models be too big?" in *ACM Conference on Fairness, Accountability, and Transparency*, 2021.

[211] M. Nadeem, A. Bethke, and S. Reddy, "StereoSet: Measuring stereotypical bias in pretrained language models," in *Association for Computational Linguistics (ACL)*, 2021.

[212] D. Ghosh *et al.*, "Learning to reach goals via iterated supervised learning," in *International Conference on Learning Representations (ICLR)*, 2021.

[213] G. Furman *et al.*, "A sequence modelling approach to question answering in text-based games," in *Association for Computational Linguistics (ACL)*, 2022.

[214] P. Ammanabrolu *et al.*, "How to avoid being eaten by a grue: Structured exploration strategies for textual worlds," *arXiv preprint arXiv:2006.07409*, 2020.

[215] X. Guo *et al.*, "Interactive fiction game playing as multi-paragraph reading comprehension with reinforcement learning," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[216] J. Tuyls *et al.*, "Multi-stage episodic control for strategic exploration in text games," in *International Conference on Learning Representations (ICLR)*, 2022.

[217] M. Atzeni *et al.*, "Case-based reasoning for better generalization in textual reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2022.

[218] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[219] K. Cobbe *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[220] L. Gao *et al.*, "Pal: Program-aided language models," *arXiv preprint arXiv:2211.10435*, 2022.

[221] W. Chen *et al.*, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.

[222] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," *arXiv preprint arXiv:2302.04761*, 2023.

[223] P. Shaw *et al.*, "Compositional generalization and natural language variation: Can a semantic parsing approach handle both?" in *Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL)*, 2021.

APPENDIX A ARTICLE 1 / LONG PROOF PSEUDO-CODE

```
def get_long_proof(story_facts, rules, query):
    """
    :params story_facts: list of (e_1, r, e_2) facts
    :params rules: list of composition rules. each rule is a dict
                   of the form {r1--r2: r3}
    :params query: tuple of entities for which we must find a relation (src, tgt)
    """
    proof = []  # list of proof steps to return
    # get all known relations (original, and reversed)
    all_facts = []
    for (e1, r, e2) in story_facts:
        inv_r = reverse_fact(e1, r, e2)
        all_facts.append((e1, r, e2))
        all_facts.append((e2, inv_r, e1))

    # go through every possible pair of facts
    for f1, f2 in itertools.combinations(all_facts, 2):
        e11, r1, e12 = f1
        e21, r2, e22 = f2
        inv_r1 = reverse_fact(e11, r1, e12)
        inv_r2 = reverse_fact(e21, r2, e22)

        # find the possible AB+BC combination.
        # there are 4 possible ways to combine 2
        #  sentences with 2 entities each and 1 in common:
        if e11 == e21 and e12 != e11 and e12 != e22:
            # AB+BC <=> inv_f1+f2
            A, new_r1, B = e12, inv_r1, e11
            B, new_r2, C = e21, r2, e22
            inv_r1 = r1
        elif e11 == e22 and e12 != e11 and e12 != e21:
            # AB+BC <=> f2+f1
            A, new_r1, B = e21, r2, e22
            B, new_r2, C = e11, r1, e12
            # swap inv_r1 and inv_r2
```

```
        inv_r1, inv_r2 = inv_r2, inv_r1
    elif e12 == e21 and e11 != e12 and e11 != e22:
        # AB+BC <=> f1+f2
        A, new_r1, B = e11, r1, e12
        B, new_r2, C = e21, r2, e22
    elif e12 == e22 and e11 != e12 and e11 != e21:
        # AB+BC <=> f1+inv_f2
        A, new_r1, B = e11, r1, e12
        B, new_r2, C = e22, inv_r2, e21
        inv_r2 = r2
    else:
        continue  # invalid pair of facts

    # try to combine AB+BC
    if new_r1--new_r2 in rules:
        r3 = rules[new_r1--new_r2]
        inv_r3 = reverse_fact(A, r3, C)
        all_facts.append((A, r3, C))
        all_facts.append((C, inv_r3, A))
        proof.append(since A new_r1 B and B new_r2 C then A r3 C)
    # try to combine CB+BA
    elif inv_r2--inv_r1 in rules:
        r3 = rules[inv_r2--inv_r1]
        inv_r3 = reverse_fact(C, r3, A)
        all_facts.append((C, r3, A))
        all_facts.append((A, inv_r3, C))
        proof.append(since C inv_r2 B and B inv_r1 A then C r3 A)
    else:
        continue  # invalid pair of facts

    # check if we found the link between the two queried entities
    (A, r, B) = all_facts[-1]
    if A==query[0] and B==query[1]: break
    if A==query[1] and B==query[0]: break
return proof
```

APPENDIX B ARTICLE 2 / INPUT - OUTPUT EXAMPLES

CLUTRR.

lvl.2 input:	"question : How is Anne related to Gary ? context : Brett is Anne 's father . Gary is a son to Brett ."
output:	"answer : Anne has a brother named Gary ."
lvl.4 input:	"question: What is the family connection between Patricia and Timothy ? context : May is the aunt of Doris . Patricia has a daughter called Doris . Timothy is Charles 's brother ." May has a son called Charles .
output:	"answer : Timothy is the nephew of Patricia ."

ProofWriter.

Depth-0 input:	"context : The cow is round. The cow needs the lion. The cow needs the rabbit. The cow sees the lion. The cow visits the rabbit. The lion is round. The rabbit is kind. The rabbit visits the tiger. The tiger is big. The tiger is kind. The tiger sees the rabbit. The tiger visits the rabbit. If something is kind and it visits the rabbit then it is young. If something sees the tiger and it visits the lion then it sees the rabbit. If something is big and young then it sees the lion. If something visits the rabbit then the rabbit needs the lion. If something is big then it visits the rabbit. If something sees the tiger then it is rough. If something visits the rabbit and it is kind then the rabbit needs the lion. If something is rough and kind then it visits the lion. If something needs the lion then it is big. question : The tiger visits the rabbit."
Depth-0 output:	"answer : True"
Depth-2 input:	"context : Anne is nice. Charlie is blue. Charlie is furry. Charlie is green. Charlie is kind. Charlie is nice. Charlie is red. Fiona is furry. Fiona is green. Harry is furry. Harry is kind. Harry is nice. All nice people are rough. If someone is red and furry then they are blue. Rough, kind people are furry. If Charlie is furry then Charlie is nice. All furry people are nice. If someone is rough then they are kind. If someone is red and nice then they are blue. All furry people are red. question : Harry is not blue."
Depth-2 output:	"answer : False"

HotpotQA.

input:	"question : Which magazine was started first Arthur's Magazine or First for Women ? context : Arthur's Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th century. Edited by T.S. Arthur, it featured work by Edgar A. Poe, J.H. Ingraham, Sarah Josepha Hale, and others. In May 1846 it was merged into "Godey's Lady's Book". First for Women is a woman's magazine published by Bauer Media Group in the USA. The magazine was started in 1989. It is based in Englewood Cliffs, New Jersey. In 2011 the circulation of the magazine was 1,310,696 copies."
output:	"answer : Arthur's Magazine"

CoQA.

context:	"context : The Vatican Apostolic Library, more commonly called the Vatican Library or simply the Vat, is the library of the Holy See, located in Vatican City. Formally established in 1475, although it is much older, it is one of the oldest libraries in the world and contains one of the most significant collections of historical texts. It has 75,000 codices from throughout history, as well as 1.1 million printed books, which include some 8,500 incunabula. The Vatican Library is a research library for history, law, philosophy, science and theology. The Vatican Library is open to anyone who can document their qualifications and research needs." [...] "Only a handful of volumes survive from this period, though some are very significant."
input:	context above + "question : When was the Vat formally opened?"
output:	"answer : It was formally established in 1475."
input:	context above + "question : When was the Vat formally opened? answer : It was formally established in 1475. question : what is the library for? answer : research. question : for what subjects?"
output:	"answer : history, and law."
input:	context above + "question : When was the Vat formally opened? answer : It was formally established in 1475. question : what is the library for? answer : research. question : for what subjects? answer : history, and law. question : what else ?"
output:	"philosophy, science and theology."

APPENDIX C ARTICLE 2 / PREDICTION EXAMPLES

CLUTRR In this section we present output examples of our models on the CLUTRR level 3 test set.

input	no abstraction	emb-sum	emb-cat	enc-sum	enc-cat	dec-loss
Jonathan is the father of Anne . Anne is a aunt to Stephanie . Stephanie is the daughter of Veronica . How is Veronica related to Jonathan ?	Jonathan is Veronica 's father	Jonathan is Veronica 's father	Jonathan is Veronica 's uncle	Jonathan is Veronica 's father	Jonathan is Veronica 's father	Jonathan is Veronica 's father
Stephanie is the granddaughter of Jonathan . Bryant is Stephanie 's father . Eric is a son of Bryant . How are Jonathan and Eric related to each other ?	Eric is the grandson of Jonathan	Eric is Jonathan 's grand-son	Eric is Jonathan 's grand-son	Eric is Jonathan 's grand-son	Eric is Jonathan 's grand-son	Eric is Jonathan 's grand-son
Joyce has a daughter called Stephanie . Betty is a grandmother to Stephanie . Paul has a wife who is Joyce . For Paul , who is Betty ?	Betty is Paul 's mother-in-law	Betty is Paul 's mother-in-law	Betty is Paul 's aunt	Betty is Paul 's mother-in-law	Betty is Paul 's mother-in-law	Betty is Paul 's mother-in-law
Bryant is a son of Jonathan . Jonathan is the husband of Betty . Craig is a son of Bryant . Who is Craig from the point of relation of Betty ?	Craig is Betty 's son	Betty has a grandson who is Craig	Craig is Betty 's grandson	Betty has a grandson who is Craig	Craig is Betty 's son	Craig is the son-in-law of Betty
Stephanie is a granddaughter to Betty . Anne is Bryant 's sister . Bryant is a brother of Stephanie . Who is Anne from the point of relation of Betty ?	Anne is Betty 's grand-daughter	Anne is Betty 's grand-daughter	Anne is Betty 's grand-daughter	Anne is Betty 's grand-daughter	Anne is Betty 's grand-daughter	Anne is Betty 's grand-daughter
score	**4/5**	**5/5**	**3/5**	**5/5**	**4/5**	**4/5**
score on lvl.3 test set reported in Table 5.2	84.4%	85.3%	60.0%	94.8%	86.1%	74.7%

Table C.1 CLUTRR test set level 3 output examples. Correct answers are in green and wrong answers are in red for better visibility.

Out of these 5 examples, the `emb-cat` model makes two mistakes (line1 and line3), the `no-abstraction`, `enc-cat`, `dec-loss` models make one mistake (line4) and the `emb-sum` and

`enc-sum` models make zero mistake.

The `emb-cat` model mistakes (line1 & line3) are due to the prediction of uncle/aunt relations instead of father/mother-in-law respectively. It is interesting to note that these relations are similar in the sense that they all link one generation to the one just above.

It is also interesting to note that the example in which the `no-abstraction`, `enc-cat` and `dec-loss` models fail (line4), the `emb-cat` model on the other hand answers correctly. In this specific example the correct relationship is "grandson" which links one generation to 2 generations below, however the mistakes made by the failing models only link one generation to 1 below (son, son, son-in-law).

HotpotQA In this section we present HotpotQA examples in which all our abstraction models correctly answered the question but not the baseline (no abstraction) model.

```
--------------------
Question: Are both Elko Regional Airport and Gerald R. Ford International
          Airport located in Michigan?
facts   : [
"Elko Regional Airport (IATA: EKO, ICAO: KEKO, FAA LID: EKO) , formerly Elko
 Municipal Airport, is a mile west of downtown Elko, in Elko County,
 Nevada.",
"Gerald R. Ford International Airport (IATA: GRR, ICAO: KGRR, FAA LID: GRR)
 is a commercial airport in Cascade Township approximately 13 mi southeast
 of Grand Rapids, Michigan. The facility is owned by the Kent County Board
 of Commissioners and managed by an independent authority. The Federal
 Aviation Administration (FAA) National Plan of Integrated Airport Systems
 for 2017-2021 categorized it as a small hub primary commercial service
 facility."]
Answer  : no
baseline: yes
embsum  : no
embcat  : no
encsum  : no
enccat  : no
decloss : no
--------------------
```

This is a yes/no question with many acronyms and entities. The abstraction can be beneficial here in order to simplify the context.

```
--------------------
Question: What act for Innocent Records achieved Platinum sales and shares
          its name with a primary color in the RGB color model?
facts   : [
"Blue is the colour between violet and green on the spectrum of visible
 light. Human eyes perceive blue when observing light with a wavelength
 between 450 and 495 nanometres. Blues with a higher frequency and thus
 a shorter wavelength appear more violet, while those with a lower
 frequency and a longer wavelength gradually appear more green.
 Pure blue, in the middle, has a wavelength of 470 nanometres.
 In painting and traditional colour theory, blue is one of the three
 primary colours of pigments, along with red and yellow, which can be
 mixed to form a wide gamut of colours. Red and blue mixed together
 form violet, blue and yellow together form green. Blue is also a
 primary colour in the RGB colour model, used to create all the
 colours on the screen of a television or computer monitor.",
"Innocent Records was a pop record label created to cater to for EMI's
 Virgin Records more pop oriented acts. Following the success of the
 Spice Girls, Virgin Records decided to delve into the pop market.
 In doing so they poached Hugh Goldsmith from RCA Records (famous
 for steering Take That's initial flagging sales, to a multi-platinum
 act). They let him launch his own Virgin Records offshoot. His first
 signing was Billie Piper, followed by Martine McCutcheon, along with
 several dance acts Todd Terry to name one. The label continued
 to thrive well into the mid-2000s with Atomic Kitten and Blue achieving
 Platinum sales."]
Answer  : Blue
baseline: Atomic Kitten
embsum  : Blue
embcat  : Blue
encsum  : Blue
enccat  : Blue
decloss : Blue
--------------------
```

This is a question with long paragraphs in the context, abstracting entities could help simplify the context to better answer the question. Nevertheless, note that if we assume the model already knows that blue is a color (or that Atomic Kitten is not a color), then the second paragraph is enough to answer the question.

```
--------------------
Question:  William Cammisano was part of which Mafia family?
facts   : [
"William "Willie Rat" Dominick Cammisano Sr. (April 26, 1914 - January 26,
 1995) was a Kansas City, Missouri, mobster and enforcer for Nicholas
 Civella\'s Kansas City crime family.",
"The Kansas City Crime Family, also known as Civella crime family
 (pronounced ] ), is a Mafia family based in Kansas City, Missouri."]
Answer  : Kansas City crime family
baseline: Nicholas Civella
embsum  : Kansas City Crime Family
embcat  : Kansas City Crime Family
encsum  : Kansas City Crime Family
enccat  : Kansas City Crime Family
decloss : Kansas City Crime Family
--------------------
```

Here the baseline model predicted the owner of the mafia family, which is also used to refer to the mafia group according to the second paragraph. Note again, the question can be answered with the first paragraph.

```
--------------------
Question: Who is older out of Bob Saget, the American comedian, and Indian
          director S. Shankar?
facts   : [
"Shankar Shanmugam (born 17 August 1963), credited mononymously as Shankar,
 is an Indian film director and producer who predominantly works in Tamil
 cinema. He was identified by S. A. Chandrasekhar. Recognized for directing
 high budget films, he is also a pioneer of vigilante movies in Tamil. He
 made his directorial debut in "Gentleman" (1993) produced by K. T.
 Kunjumon, for which he was awarded the Filmfare Best Director Award and
 the Tamil Nadu State Film Award for Best Director. He is the highest paid
```

```
 film-maker in India among his contemporaries.",
"Robert Lane "Bob" Saget (born May 17, 1956) is an American stand-up
 comedian, actor, and television host. His television roles include Danny
 Tanner on the ABC sitcom "Full House" (1987-95) and its Netflix sequel
 "Fuller House", and hosting "America\'s Funniest Home Videos" from 1989
 to 1997. Saget is also known for his adult-oriented stand-up routine.
 He also provided the voice of the future Ted Mosby on the CBS sitcom
 "How I Met Your Mother" from 2005 to 2014."]
Answer  : Robert Lane "Bob" Saget
baseline: Ted Mosby
embsum  : Robert Lane
embcat  : Robert Lane " Bob " Saget
encsum  : Robert Lane
enccat  : Robert Lane " Bob " Saget
decloss : Robert Lane " Bob " Saget
--------------------
```

This question asks to compare two different entities of the same type. The baseline model answers with a person's name but not the correct one. Since there are lots of entities in this example, abstraction can be beneficial to simplify the context and select the correct person in the end.

Next, we present HotpotQA examples in which all our abstraction models incorrectly answered the question but not the baseline (no abstraction) model.

```
--------------------
Question: What city does Paul Clyne and David Soares have in common?
facts   : [
"Paul Clyne was the District Attorney of Albany County, New York from
 January 2001 through December 2004. A graduate of Albany Law School, he
 spent about 14 years as an assistant district attorney, before he
 was tapped by local politicians to replace the retiring District Attorney,
 Sol Greenberg. He was defeated for re-election by David Soares, first
 in the Democratic Party primary election in September 2004, and then in
 the general election in November 2004, in which he ran on an independent
 line. After a stint teaching at the New York Prosecutors Institute, he
 went into private practice as a criminal defense attorney in 2007, with an
 office in Albany, New York.",
"P. David Soares (born October 26, 1969, Brava, Cape Verde) is the Albany
 County, N.Y. District Attorney. He is a Democrat."]
Answer  : New York
baseline: New York
embsum  : Albany
embcat  : Albany
encsum  : Albany
enccat  : Albany
decloss : Albany
--------------------
```

In this example the abstraction models all answered 'Albany', which is also a valid answer. In fact, the question asks for a city and not a state. In addition, the second paragraph doesn't mention New York City, so Albany should probably have been the correct answer.

```
--------------------
Question: what producer of The Real Housewives of Orange County
          also hosts "Watch What Happens Live with Andy Cohen"?
facts   : [
"Andrew Joseph "Andy" Cohen (born June 2, 1968) is an American talk show
 and radio host, author and producer. Cohen hosts the Bravo nightly
```

```
 series "Watch What Happens Live with Andy Cohen". He is the first openly
 gay host of an American late-night talk show. After being head of
 development at Bravo for more than 10 years, Cohen resigned in November
 2013. He continues to serve as an executive producer of "The
 Real
 Housewives" franchise.",
"The eleventh season of "The Real Housewives of Orange County", an American
 reality television series, is broadcast on Bravo. It aired June 20, 2016,
 until November 21, 2016, and is primarily filmed in Orange County,
 California. Its executive producers are Adam Karpel, Alex Baskin, Douglas
 Ross, Gregory Stewart, Scott Dunlop, Stephanie Boyriven and Andy Cohen."]
Answer  : Andy Cohen
baseline: Andy Cohen
embsum  : Adam Karpel
embcat  : Adam Karpel
encsum  : Adam Karpel
enccat  : Adam Karpel
decloss : Adam Cohen
--------------------
```

In this example the abstraction models all answer with the first entity listed as being a producer of The Real Housewives, eventhough it is not the person who hosts What Happens Live. Note that this example contains the answer in the question, which may be confusing the abstraction models.

```
--------------------
Question: What Actor whose birth name was Charles Dennis Buchinsky,
          was part of the Leslie Nielsen comedy?
facts   : [
"Charles Bronson (born Charles Dennis Buchinsky; Lithuanian:
 "Karolis Dionyzas Bučinskis" ; November 3, 1921 - August 30, 2003)
 was an American actor.",
"Allan A. Goldstein (born May 23, 1949) is an American film director
 and screenwriter, perhaps best known for directing the Charles Bronson
 vehicle and the Leslie Nielsen comedy".]
Answer  : Charles Bronson
baseline: Charles Bronson
```

```
embsum  : Allan A. Goldstein
embcat  : Allan A. Goldstein
encsum  : Allan A. Goldstein
enccat  : Allan A. Goldstein
decloss : Allan A. Goldstein
--------------------
```

In this example the answer can be answered directly from the first paragraph since we are asking for the name of the actor born 'Charles Denis Buchinsky'. Here the abstraction models all answered with the person from the second paragraph. This is probably due to the fact that a two hop question, starting with entity 'Charles Dennis Buchinsky', and asking for a person would first link the first and second paragraph with the entity 'Charles Bronson', and then answer with the entity 'Allan A. Goldstein'.

APPENDIX D ARTICLE 2 / GEOQUERY RESULTS

In this section we report results of our different models on the GeoQuery [77] benchmark. In particular, we use two versions described by Shaw et al. [223], focusing on compositional generalization: a train/test split based on query length and a train/test split based on Target Maximum Compound Divergence (TMCD). Each train and test set contains 440 examples.

We trained our models on both of these splits and evaluated the exact match of the predicted output of our models. Below are our average results based on 3 random seeds:

	LENGTH split	**TMCD split**
no-abstraction	24.24% (+- 0.001)	**36.06%** (+- 0.001)
emb-sum	**27.05%** (+- 0.000)	34.47% (+- 0.001)
emb-cat	15.91% (+- 0.002)	18.56% (+- 0.001)
enc-sum	18.64% (+- 0.002)	35.08% (+- 0.001)
enc-cat	17.50% (+- 0.000)	27.58% (+- 0.001)
dec-loss	24.09% (+- 0.000)	30.38% (+- 0.001)

We can see that for the length split, one abstraction model performs better than the baseline (no-abstraction), but on the TMCD split, models with abstraction are not better than the baseline. However, we believe this dataset may not be a good candidate to showcase if abstraction is useful for 2 reasons:

First, all examples in the dataset have a few number of entities (2 maximum):

entities per example	**examples**	**percentage of the dataset**
0	207	23.5%
1	667	75.8%
2	6	0.7%
Total	**880**	**100%**

This makes each abstract input very similar to the original input. Thus the abstract sequence provides very little information to the model.

Second, the dataset as a whole contains a few number of entities. 12% of tokens per input are tagged as entities. This is a relatively small number compared to the other datasets as we can see below.

Fraction of tokens per input in each dataset identified as entities by spacy NER:

Data	**avg (+-std)**
CLUTRR	0.22 (+- 0.019)
Proofwriter	0.36 (+- 0.036)
HotpotQA	0.37 (+- 0.085)
CoQA	0.18 (+- 0.095)
Geoquery	0.12 (+- 0.080)

We can say from these results that in order for the abstraction technique to be useful, the dataset must contain at least more than 2 entities per input sequence.