University of California
Santa Barbara

# Efficient Natural Language Processing with Limited Data and Resources

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Hong Wang

Committee in charge:

Professor Xifeng Yan, Chair
Professor Tao Yang
Professor Shiyu Chang

September 2023

The Dissertation of Hong Wang is approved.

---

Professor Tao Yang

---

Professor Shiyu Chang

---

Professor Xifeng Yan, Committee Chair

June 2023

Efficient Natural Language Processing with Limited Data and Resources



by

Hong Wang

To my beloved wife, Yue, who has unwaveringly supported me throughout this journey.

## Acknowledgements

First and foremost, I extend my deepest appreciation to my advisor, Xifeng Yan, for his invaluable guidance and support throughout my research journey. His advice has not only contributed to the development of my work but has also been instrumental in shaping my personal growth. During challenging times, I am grateful for his steadfast support and trust. I have learned an immense amount from him, and I aspire to continue following his example as a role model.

I am also indebted to my committee members, Tao Yang and Shiyu Chang, for their valuable suggestions in this work, as well as their insights on presentation and future directions. Their expertise and feedback have been indispensable.

I extend my thanks to William Wang, who introduced me to the field of Natural Language Processing and provided invaluable guidance on conducting research. I am also grateful to Mo Yu, Xiao Xiao Guo, and Shiyu Chang for their valuable feedback and insightful discussions during the initial stages of my study. It has been a pleasure to collaborate and learn from them.

During my internship, I was fortunate to have wonderful mentors: Jingfei Du, Ves Stoyanov, Gengshen Fu, Tao Zhang, Xiao Bo, and Rajath Kumar. I am grateful for their guidance and support, which greatly contributed to my growth during this period. I would like to extend special thanks to Gengshen Fu for his support not only in technical matters but also in career development. I have truly gained invaluable knowledge from his mentorship.

I would also like to express my appreciation to my labmates, Wenhan Xiong, Jing Qian, Shiyang Li, Wenhu Chen, Zekun Li, and Weizhi Wang, for their insightful discussions and collaborative efforts. Their contributions have enriched my research experience.

Lastly, I want to express my heartfelt gratitude to my family for their unwavering

support. I am truly grateful to my parents, whose unwavering support and unconditional love have been a constant source of strength for me. I want to express my deepest appreciation for my wife, Yue. I consider myself incredibly fortunate to have her by my side. Without her unwavering encouragement and selfless sacrifices, I would never have embarked on this journey. Her presence and companionship have consistently provided me with the strength and motivation to overcome challenges and pursue my goals with resolute determination.

# Curriculum Vitæ

Hong Wang

## Education

2018-2023 Ph.D. in Computer Science, University of California, Santa Barbara.
Advisor: Prof. Xifeng Yan
Focus: Natural Language Processing, Deep Learning

2014-2018 B.S. in Computer Science, Nanjing University.
Advisor: Prof. Yang Yu
Focus: Reinforcement Learning, Optimization

## Publications

1. Jing Qian*, Hong Wang*, Zekun Li, Shiyang Li, Xifeng Yan, "Limitations of Language Models in Arithmetic and Symbolic Induction", in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023)*, Toronto, Canada.
2. Zekun Li, Wenhu Chen, Shiyang Li, Hong Wang, Jing Qian, Xifeng Yan, "Dialogic: Controllable Dialogue Simulation with In-Context Learning", in *Findings of the Conference on Empirical Methods in Natural Language Processing (Findings of EMNLP 2022)*, Abu Dhabi
3. Zekun Li*, Hong Wang*, Alon Albalak, Yingrui Yang, Jing Qian, Shiyang Li, Xifeng Yan, "Making Something out of Nothing: Building Robust Task-oriented Dialogue Systems from Scratch", in *Proceedings of Alexa Prize TaskBot*.
4. Jing Qian, Hong Wang, Mai ElSherief, Xifeng Yan, "Lifelong Learning of Hate Speech Classification on Social Media", in *Proceedings of 2021 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2021)*, online.
5. Hong Wang*, Wenhan Xiong*, William Yang Wang, "Progressively Pretrained Dense Corpus Index for Open-Domain Question Answering", in *Proceedings of the 16th conference of the European Chapter of the Association for Computational Linguistics (EACL 2021)*, online.
6. Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, William Wang, "HybridQA: A Dataset of Multi-Hop Question Answering over Tabular and Textual Data", In *Findings of the Conference on Empirical Methods in Natural Language Processing (Findings of EMNLP 2020)*, online.
7. Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou and William Yang Wang, "TabFact: A Large-scale Dataset for Table-based Fact Verification", In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*, Addis Ababa, Ethiopia.

8. Hong Wang, Xin Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang and William Wang, "Self-Supervised Learning for Contextualized Extractive Summarization", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, Florence, Italy.
9. Wenhan Xiong, Jiawei Wu, Hong Wang, Vivek Kulkarni, Mo Yu, Xiaoxiao Guo, Shiyu Chang and William Wang, "TweetQA: A Social Media Focused Question Answering Dataset", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, Florence, Italy.
10. Hong Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang and William Wang, "Sentence Embedding Alignment for Lifelong Relation Extraction", in *Proceedings of the 17th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, Minneapolis, MN, USA.
11. Hong Wang, Hong Qian and Yang Yu, "Noisy derivative-free optimization with value suppression", in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, New Orleans, LA, USA.

**Preprints**

1. Hong Wang, Xuan Luo, Weizhi Wang and Xifeng Yan, "Bot or Human? Detecting ChatGPT Imposters with A Single Question", technical report, 2023
2. Shiyang Li, Jianshu Chen, Yelong Shen, Zhiyu Chen, Xinlu Zhang, Zekun Li, Hong Wang, Jing Qian, Baolin Peng, Yi Mao, Wenhu Chen, Xifeng Yan, "Explanations from Large Language Models Make Small Reasoners Better", technical report, 2022
3. Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Hong Wang, Shiyu Chang, Murray Campbell and William Wang, "Simple yet Effective Bridge Reasoning for Open-Domain Multi-Hop Question Answering", MRQA: Machine Reading for Question Answering workshop at EMNLP-IJCNLP 2019, Hong Kong, China.
4. Hong Wang, Christfried Focke, Rob Sylvester, Nilesh Mishra and William Wang, "Fine-tune Bert for DocRED with Two-step Process", technical report, 2019.
5. Hong Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang and William Wang, "Meta Reasoning over Knowledge Graphs", technical report, 2019.

**Patents**

1. Hong Wang, Gengshen Fu, Bo Xiao and Tao Zhang, "Unified wakeword and language understanding processing", patent submitted, 2021.

## Experience

| | |
|---|---|
| 09/2022-06/2023 | Team Lead, Amazon Alexa Grand SocialBot Challenge |
| 06/2022-09/2022 | Applied Scientist Intern, Amazon Alexa, Cambridge, MA |
| 09/2021-06/2022 | Team Lead, Amazon Alexa TaskBot Challenge |
| 06/2021-09/2021 | Applied Scientist Intern, Amazon Alexa, Cambridge, MA |
| 06/2020-09/2020 | Research Intern, Meta AI, Menlo Park, CA |
| 10/2017-01/2018 | Software Engineer Intern, Microsoft Research Asia, Beijing |

## Awards & Honors

| | |
|---|---|
| 2018, 2020, 2022 | Chancellor's Fellowship, University of California, Santa Barbara |
| 2018 | Outstanding Undergraduate Award, Nanjing University |
| 2018 | Excellent Bachelor Thesis Award, Nanjing University |
| 2017 | National Scholarship, Ministry of Education, China |
| 2014-2016 | National Endeavor Scholarship, Ministry of Education, China |

## Teaching

| | |
|---|---|
| Fall 2019 | Teaching assistant, CS165B, Machine Learning, UCSB |
| Spring 2020 | Teaching assistant, CS165B, Machine Learning, UCSB |
| Spring 2021 | Teaching assistant, CS165B, Machine Learning, UCSB |

## Talks

| | |
|---|---|
| 05/28/2019 | Sentence Embedding Alignment for Lifelong Relation Extraction, IBM invited online talk |

**Abstract**

Efficient Natural Language Processing with Limited Data and Resources

by

Hong Wang

Natural language processing (NLP) has long been regarded as the pinnacle of artificial intelligence, aiming to achieve a comprehensive understanding of human languages. In recent years, the field has experienced significant advancements with the transition from rule-based approaches to deep learning methodologies. However, the standard approaches often rely on vast amounts of data for learning, highlighting the necessity for more data-efficient techniques. Additionally, effectively utilizing available resources while addressing the challenges of frequent model updates and safeguarding against malicious attacks that exploit limited resources presents another significant problem in NLP.

This dissertation focuses on the development of efficient natural language processing (NLP) models under limited data and the effective utilization of available resources. In the first part, we address the challenge of learning models with limited data. For scenarios where only a few examples are available, we propose a meta-learning approach that leverages task-specific meta information to effectively learn new models. For cases with a moderate amount of data but still insufficient for more demanding tasks, we introduce self-supervised learning techniques to enhance performance by incorporating additional learning tasks from the available data. We also explore the limitations of even state-of-the-art language models, such as GPT-3, in handling out-of-distribution data shifts and propose a tutor-based learning approach that converts out-of-distribution problems into in-distribution ones through step-by-step demonstrations.

In the second part, we shift our focus to optimizing resource utilization in NLP. Given

the rapidly changing nature of the world, frequent updates of deployed models with new data are crucial. We present innovative approaches for effectively updating models in lifelong learning scenarios. As the adoption of large language models as backbone dialogue systems gains popularity, resource limitations become a significant concern. To counter malicious attacks, particularly Distributed Denial of Service (DDoS) attacks, we investigate the detection of bot imposters using a single question. By accurately distinguishing between human users and bots, our objective is to maximize resource allocation for real users and ensure uninterrupted service.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Language serves as an indispensable tool, playing a crucial role in human communication for countless generations. It enables the exchange and transmission of information, thoughts, ideas, emotions, and knowledge, bridging the gaps between individuals and fostering collaboration, shared experiences, and mutual understanding. Through language, societies have effectively preserved and shared accumulated wisdom, cultural heritage, and scientific breakthroughs, contributing to the continuous progress and development of civilizations.

However, in the era of information technology, machines have significantly evolved in their power and user-friendliness. Despite their capabilities, machines still rely on human input to understand task requirements and carry out operations. For example, human intervention is necessary when taking a customer's order at a restaurant. Consequently, human communication has become a bottleneck in our society, hindering overall efficiency. Unlike machines, humans cannot duplicate themselves, work round the clock, or instantaneously process information by reading documents or gathering data.

Fortunately, Natural Language Processing (NLP) has emerged as a promising solution to reduce the extensive need for human involvement in various processes. By harnessing the power of NLP, machines can directly comprehend and execute human language, thereby diminishing reliance on human intervention. This advancement enables seamless communication and interaction between humans and machines, streamlining tasks and enhancing overall efficiency. In the past, machine interactions primarily relied on user interfaces and machine-specific languages, resulting in certain limitations and complexities. However, the introduction of NLP has revolutionized this landscape, simplifying human-machine interactions. When combined with external tools such as robots, NLP offers a significant boost in capabilities, empowering machines to perform a broader range of tasks. Imagine a future scenario where everyone has access to a personalized robot assistant equipped with advanced NLP capabilities. These robot assistants become invaluable allies, capable of understanding and fulfilling various requirements. By delegating tasks to these assistants, individuals gain more time and freedom to focus on other meaningful endeavors, while the assistants efficiently handle intricate details and comprehend the needs of others.

In recent years, Natural Language Processing (NLP) has witnessed remarkable progress, driven in large part by the emergence of pre-trained language models like BERT [1] and GPT-2 [2]. These advancements have been pivotal in propelling the field forward. Building on these successes, subsequent developments, such as GPT-3 [3], demonstrated the transformative potential of scaling up model sizes, leading to significant performance improvements. Furthermore, ingenious techniques like instruction tuning and reinforcement learning from human feedback have enabled ChatGPT [4] to achieve astonishing results in text generation, empowering users in various downstream tasks such as summarization, question answering, and polished writing. Its exceptional capabilities quickly captivated the attention of 1 million users in just 5 days. However, despite its impres-

sive text generation abilities, ChatGPT remains limited to pattern matching solely on text-to-text interactions. Expanding on these foundations, GPT-4 [5] took a significant step forward by incorporating image inputs in addition to text, broadening the scope of possibilities. Moreover, ChatGPT Plugins [6] further extended the model's capabilities by integrating additional tools like OpenTable, allowing the model to directly book tables through its interface. This trend highlights the increasing utilization of Large Language Models, such as ChatGPT, as the core component capable of processing diverse input data, including text and visual inputs, and generating outputs in the form of text or performing operations on various software, tools, or even robots.

## 1.2 Challenges

After highlighting the significance and motivation behind NLP, it is essential to address some of the challenges posed by limited data and resources.

One major challenge arises when dealing with tasks that have only a limited number of examples. In such scenarios, there is a risk of the model overfitting to the available examples, leading to poor generalization to new cases. Additionally, certain situations may involve encountering out-of-distribution (OOD) data, where the distribution during training differs from that during inference. Consequently, the model struggles to generalize to the new distribution, even if it is a large language model, which presents a significant limitation.

Another crucial challenge is the constraint of resources. With limited resources at hand, it becomes essential to strategize how to maximize their usage while avoiding any misuse. As the world evolves rapidly, datasets and tasks continue to update, necessitating the constant need to train new models incorporating all the accumulated knowledge. This process can be resource-intensive. Lifelong learning seeks to address this concern

by focusing on learning new knowledge while ensuring the model maintains good performance on previously learned tasks. However, lifelong learning faces the challenge of catastrophic forgetting, where the model tends to easily forget the knowledge acquired from previous tasks.

## 1.3 Contributions

In the first part of this dissertation, our focus centers on addressing the challenges of learning with limited data. To tackle learning with few-shot examples, we introduce few-shot learning applied to the task of multi-hop reasoning over knowledge graphs. To facilitate this, we present two meticulously crafted datasets specifically designed for this purpose. Our proposed approach involves employing a meta-encoder, which encodes task-specific information to generate a superior task-dependent model for new tasks. In the context of multi-hop reasoning, we leverage neighbor encoders and path encoders to incorporate task-specific information, and our experiments demonstrate the effectiveness of the augmented meta-encoder in enhancing model performance.

Moving beyond learning solely from the available data, we delve into self-supervised learning for the summarization task. We explore the use of the entire document to learn contextualized sentence representations through self-supervision, without relying on any human annotations. We experiment with various self-supervised approaches for extractive summarization, and one of these approaches achieves state-of-the-art results using a basic hierarchical model. Benefiting from self-supervised pre-training, the summarization model demonstrates enhanced sample efficiency and faster convergence compared to models trained from scratch.

Subsequently, we investigate the problem of handling out-of-distribution (OOD) data with large language models (LMs). Specifically, we study the limitations of LMs in

arithmetic and symbolic induction tasks when facing OOD scenarios. Through rigorous experimentation, we identify a set of simple symbolic manipulation tasks that reveal the challenges LMs encounter in these domains. To address these limitations, we explore several potential techniques, including positional markers, fine-grained computation steps, and LMs with callable programs. While these techniques offer some mitigation, they do not fully solve the generalization problem. Finally, we demonstrate the effectiveness of LMs with tutor-based training, achieving 100% accuracy in OOD and repeated symbol situations. Our comprehensive analysis sheds light on new possibilities to overcome the limitations of LMs in symbolic manipulation tasks, providing valuable insights for future research in this domain.

In the second part of this dissertation, we shift our focus towards optimizing resource utilization in NLP, recognizing the importance of updating deployed models with new data in the ever-changing world. To address this, we present innovative approaches for effectively updating models in lifelong learning scenarios. First, we introduce the lifelong relation detection problem and construct lifelong relation detection benchmarks using two datasets with extensive relation vocabularies: SimpleQuestions [7] and FewRel [8]. In tackling this challenge, we propose a simple yet powerful memory replay approach, which outperforms current popular methods such as EWC and GEM. Additionally, we propose an alignment model designed to mitigate the catastrophic forgetting problem, which slows down the rapid changes in the embedding space during lifelong learning. This approach offers a promising solution to enhance model performance and preserve previously learned knowledge while adapting to new tasks.

As large language models increasingly serve as the backbone of dialogue systems, there arises a pressing concern regarding their potential misuse for malicious purposes. Among the most significant threats is the impersonation of human users by these models, enabling nefarious activities like fraud, spamming, or denial-of-service attacks. To

safeguard against such misuse, we propose a novel framework called FLAIR, designed for detecting conversational bots in an online environment. This approach effectively differentiates human users from bots by using questions that are easy for humans but difficult for bots, and vice versa. Through comprehensive experiments, we demonstrate the effectiveness of this framework and identify the strengths of different question types. By providing online service providers with a new tool to safeguard against fraudulent activities, FLAIR ensures that resources are utilized optimally for real humans.

# Part I

# Learning with Limited Data

# Chapter 2

# Task-Dependent Meta-Learning with Few-Shot Examples

## 2.1 Introduction

Knowledge Graphs [9, 10, 11] represent entities' relational knowledge in the form of triples, i.e., (subject, predicate, object), and has been proven to be essential and helpful in various downstream applications such as question answering [12, 13, 14, 15]. Since most existing KGs are highly incomplete, a lot of studies [16, 17, 18] have been done in automatically completing KGs, i.e., inferring missing triples. However, most of these studies only focus on frequent relations and ignore the relations with limited training samples. As a matter of fact, a large portion of KG relations are actually long-tail, i.e., they have very few instances. Therefore, it is important to consider the task of knowledge graph completion under few-shot learning setting, where limited instances are available for new tasks. Previous research [19] first propose a graph network based metric-learning framework for this problem but the metric is learned upon graph embeddings and their method does not provide reasoning rationales for the predictions.

In contrast, we propose a meta reasoning agent that learns to make predictions along with multi-hop reasoning chains, thus the prediction of our model is fully explainable. In this problem setting, each task corresponds to a particular relation and the goal is to infer the end entity given the start entity (i.e., the query). Following the recent work [20, 21, 22, 23, 24, 25] on meta-learning, we aim to learns a reasoning agent that can effectively adapt to new relations with only a few examples. This is quite challenging since the model must learn to leverage its prior learning experience for fast adaptation and at the same time avoid overfitting on the few-shot training examples. Model-agnostic meta-learning algorithm (MAML) [21] is a popular and general algorithm to solve this problem. It aims to learn an initial model that captures the common knowledge shared within the tasks so that it can adapt on the new task quickly. But one problem of MAML is that it only learns a single initial model, which can not fit the new task without training, and has limited power in the case of diverse tasks [26]. Another problem is that MAML only learns the common knowledge shared within the tasks without taking advantage of the relationship between them since no task-specific information is used when learning the initial model.

In order to learn the relationship between tasks, the model must be aware of the identity of the current task, such as the query relation in our problem. But simply using task identity will be a problem, since there is no way to initialize the identity of the new task except random initialization. We try to solve this problem via a meta-encoder that learns the task representation from meta-information which is available on the new task as well. Specifically, the meta-encoder is used to encode the task-specific information and generate the representation of the task as part of parameters. Through this way, different tasks will have different representations, thus different initial models. Also, since the presentation of the task is available, the model can leverage the relationship between different tasks. To apply this idea in our problem, we propose two meta-encoder to encode

two different kinds of task-specific information. One is to use the neighbor encoder to encode the start entity and the end entity, and then use the difference between the embedding of the start entity and the end entity as the task representation. But this take-specific information is not robust when the number of neighbors is small. Thus we propose another way for the case which encodes the path from the start entity to the end entity. On two constructed few-shot multi-hop reasoning datasets, we show that the augmented meta-encoder yields much better initial point and outperforms several few-shot learning baselines.

The main contributions of this work include:

• We introduce few-shot learning on the task of multi-hop reasoning over knowledge graph, and present two constructed datasets for this task.

• We propose to use meta-encoder to encode task-specific information so as to generate better task-dependent model for the new task.

• We apply neighbor encoder and path encoder to leverage the task-specific information in multi-hop reasoning task, and experiments verify the effectiveness of the augmented meta-encoder.

## 2.2 Related Work

**Reasoning over Knowledge Graphs** Knowledge graph reasoning aims to infer the existence of a query relation between two entities. There are two general approaches for knowledge graph reasoning. The embedding based approaches [27, 16, 28, 17, 29] learn the representations of the relations and entities in the KG with some heuristic self-supervised loss functions, while path search based approaches [18, 30, 31, 32, 33, 34, 35] solve this problem through multi-hop reasoning, i.e., finding the reasoning path between two entities. In spite of the superior performance of embedding-based methods, they can

not capture the complex reasoning patterns in the KG and are lack of explainability.

Due to its explainability, multi-hop reasoning has been investigated a lot in recent years. The Path-Ranking Algorithm (PRA) [18] is a primal approach that learns random walkers to leverage the complex path features. [36, 37] improves upon PRA by computing feature similarity in the vector space. Recursive random walk integrates the background KG and text [38]. There are also other methods using convolutional neural network [39] and recurrent neural networks [30]. More recently, [31] first applies reinforcement learning for learning relational paths. [32] proposes a more practical setting of predicting end entity given the query relation and the start entity. [34] reshapes the rewards using pre-trained embedding model. [35] uses Monte Carlo Tree Search to overcome the problem of sparse reward.

**Meta-learning** Meta-learning aims to achieve fast adaption on new tasks through meta-training on a set of tasks with abundant training examples. It has been widely applied in few-shot learning settings where limited samples are available [22, 23]. One important category of meta-learning approaches is initialization based methods, which aims to find a good initial model that can fast adapt to new tasks with limited samples [21, 40]. However, they only learn a single initial model and do not leverage the relationship between tasks. [41] proposes to learn a data-dependent latent generative representation of the model parameters and conduct gradient-based adaptation procedure in this latent space. Another related work is Relation Network [24], which consists of am embedding module to encode samples and a relation module to capture the relation between samples.

## 2.3 Background

In this section, we will first introduce the multi-hop reasoning task. Then we will extend it to the meta-learning setting and introduce the popular framework (MAML) for few-shot learning.

### 2.3.1 Multi-hop Reasoning Problem

In this problem, there is a background graph $G$, and a set of query relations $R$. Each query relation has its own training and testing triple $(e_s, r, e_t)$, where $e_s$, and $e_t$ are the start entity and end entity in the KB, while $r$ is the query relation. Given the start entity $e_s$ and the query relation $r$, the task is to predict the end entity $e_t$, along with a support reasoning path from $e_s$ to $e_t$ in $G$. The length of the path is set to be fixed, and an additional *STOP* edge is added for each entity to point at itself so that the model is able to stay in the end entity.

We give an example to better explain this task. Consider the relation of *Nationality* with a training triple: *(Obama, Nationality, American)*. Given the start entity and the query relation, *(Obama, Nationality)*, the model is expected to find a path with a fixed length in $G$ from *Obama* to *American*. A general framework to solve this problem is to train an agent that predicts the next relation based on the current entity, the query relation, and the visited path at each step. In expectation, the agent should give the reasoning path *(BornIn, CityIn, ProvinceIn)*, and predict the end entity as *American*.

### 2.3.2 Meta-learning for Multi-hop Reasoning

For multi-hop reasoning problem, we define a task as the inference of a specific relation's end entity conditioned on the start entity. It is easy to see that each relation forms an individual task. In the meta-learning framework, the tasks are divided into

three disjoint sets called meta-training, meta-dev, and meta-test set respectively. The goal of meta-learning is to train an agent that can quickly adapt on the new tasks in meta-test set with limited data by leveraging prior learning experience.

Following standard meta-learning setting as in [21], our setting consists of two phases, the **meta-training** and **meta-test** phase. In the meta-training phase, the agent learns on a set of meta-training tasks $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_N\}$, where each task $\mathcal{T}_i$ has its own training and validation set denoted as $\{D_i^{train}, D_i^{valid}\}$. By learning on the meta-training tasks $\mathcal{T}$, the agent is expected to gain some knowledge about the reasoning process, which can help learn faster on new tasks.

In the meta-test phase, the trained agent will be evaluated on a set of new tasks in the meta-dev/meta-test task set $\mathcal{T}' = \{\mathcal{T}'_1, \mathcal{T}'_2, \cdots, \mathcal{T}'_{N'}\}$. Each task $\mathcal{T}'_i$ has its own training and testing set denoted as $\{D_i'^{train}, D_i'^{test}\}$, where $D_i'^{train}$ only has limited training samples. The agent will be fine-tuned on each task $\mathcal{T}'_i$ using $D_i'^{train}$ for fixed gradient steps, and be evaluated after each gradient step. The macro-average on all tasks in $\mathcal{T}'$ is reported as its performance of meta-learning. Note that the number of fine-tuning steps should be chosen according to the model's performance on meta-dev tasks, and use the fixed chosen steps on meta-test tasks directly, since there are only limited samples in the new task, which are not sufficient for choosing a feasible fine-tuning step.

### 2.3.3 MAML Framework

Let $f$ denotes the reasoning model in our setting that maps the observation to the action, i.e., next relation to be taken. The objective of MAML [21] is to find a good model initialization $f_\theta$ which can quickly adapt to the new tasks after a few adaptions. We will first introduce the objective function of MAML, and then illustrate how to optimize it in the following part.

Let $\theta$ denote the parameter of the current model, and $\theta'$ denote the updated parameter using samples from task $\mathcal{T}_i$. For example, suppose we use one gradient update on $\mathcal{T}_i$, then we have:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i} \left( f_\theta \right).$$

The meta-objective is to optimize the performance of $f_{\theta_i'}$ across tasks sampled from $p(\mathcal{T})$. More formal definition is as follows:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i} \left( f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)} \right)$$

To optimize this problem, we sample a batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$. For each task $\mathcal{T}_i$, two subsets ($D_i$ and $D_i'$) of training examples will be sampled independently. $D_i$ is used to compute the updated parameters $\theta'$. Then $\theta$ is optimized to minimize the objective function using $D_i'$. Formally, we have

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}^{D_i} \left( f_\theta \right).$$

$$\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}^{D_i'}(f_{\theta_i'})$$

The above optimization requires the computation of second-order gradient, which is computationally expensive. In practice, people usually use first-order update rule instead, which has similar performance but needs much less computation [21, 40]:

$$\theta = \theta - \beta \nabla_{\theta'} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}^{D'}(f_{\theta'})$$

## 2.4 Meta-Learning of Deep Reasoners

### 2.4.1 MAML with Task-specific Initialization

MAML learns a single initial model that does not depend on any task-specific information. It works by adapting the initial model through gradient update on the target task. In other words, the initial model learns some common knowledge shared by the tasks, so that it can adapt to new tasks quickly. However, MAML is not able to capture the relationship between different tasks because it is lack of task-specific information. One easy way to inject task information is to use task identity, such as the embedding of query relation in our KB reasoning problem. But this solution could incur two problems. First, the model will learn some knowledge that only applies to a specific task, which is hard to transfer when adapting to new tasks. Second, when there comes a new task, we can not easily initialize the task identity, e.g. the embedding of a new query. Therefore, we propose to use a meta-encoder to encode the task-specific information, which can not only enable the model to learn the relationship between different tasks but also allows the model adapt in the new task faster since the model can leverage the task-specific information of the new tasks as well.

Let $x$ and $\hat{x}$ denote the input data and task-specific information respectively. $g$ is the meta-encoder that encodes $\hat{x}$, and $f$ is the model which takes both $x$ and $g(\hat{x})$ as inputs to predict the outputs, i.e., $f(x, g(\hat{x}))$ is used for prediction. Note that we hope $g(\hat{x})$ can encode the information about the whole target task instead of just $x$ itself so that $g(\hat{x})$ can also benefit other instances $x'$ within the same task $\mathcal{T}_i$, i.e., $f(x', g(\hat{x}))$ should perform well for any $x' \in \mathcal{T}_i$. This is because the task-specific information may not be available for the testing sample. For example, the end entity we use as the task-specific information is not available in new testing samples. To achieve this goal, we apply meta-gradient methods which is similar to MAML. Given a task $\mathcal{T}_i$, we will sample two subsets

of instances $D_i$ and $D'_i$. The updated parameter is computed using $D_i$:

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i} \left( f_\theta(D_i, g_\theta(\hat{D}_i)) \right).$$

Then meta-gradient is computed using $\hat{D}_i$ and $D'_i$, where $\hat{D}_i$ is used for initialization.

$$\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}(D'_i, g_{\theta'_i}(\hat{D}_i)))$$

The first order update rule can be written as:

$$\theta = \theta - \beta \nabla_{\theta'_i} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}(D'_i, g_{\theta'_i}(\hat{D}_i)))$$

The details are shown in Algorithm 1. At first, a batch of tasks will be sampled. For each task $\mathcal{T}_i$, we sample two subsets of instances $(D_i, D'_i)$, and compute the meta information $\hat{D}_i$ based on $D_i$, which is the neighbor of start and end entity or the reasoning path between them for the multi-hop reasoning problem. In the following procedure, the updated parameters $\theta'_i$ will be computed for each task (line 7-9). In meta-update step (line 11), we update $\theta$ to minimize the loss of $\theta_i$ using new instances $D'_i$ and the task representation $\hat{D}_i$.

For testing on a new task $\mathcal{T}'_i$, we obtain the task representation $g(\hat{x})$ based on the few-shot samples $x \in D_i^{\prime train}$. Then we fine-tune $f$ and $g$ using the data $D_i^{\prime train}$. The model makes prediction on testing samples $x\prime \in D_i^{\prime test}$ using $f(x\prime, g(\hat{x}))$.

### 2.4.2 Model

The general framework of our model is shown in Figure 2.1. The original reasoning agent takes start entity and query relation as inputs, and output the reasoning path and

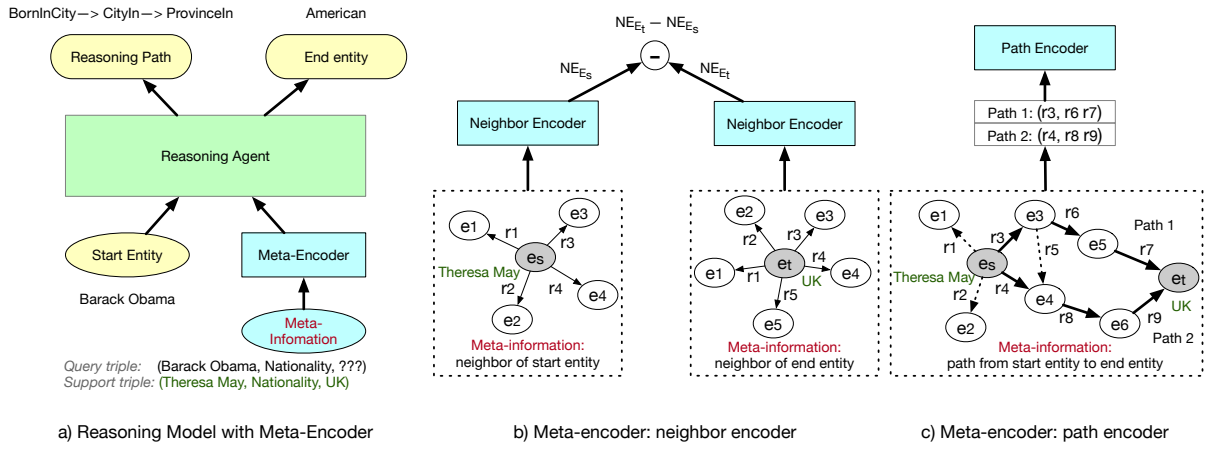



Figure 2.1: The model we use for meta-reasoning over knowledge graph. a) is the general framework of the model. b) and c) are our neighbor encoder and path encoder respectively.

**Algorithm 1** MAML with Meta-Encoder

**Require:**

$p(\mathcal{T})$: the distribution of tasks
$\alpha, \beta$: learning rates for adaptation and meta-update
$k$: the number of adaptations
$f, g$: the reasoning model and meta-encoder

Randomly initialize $\theta$
**for** step $= 0$ : M-1 **do**
  **for** batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$ **do**
    Sample task instances $(D_i, D_i')$ from $\mathcal{T}_i$
    Compute task specific information $\hat{D}_i$
    Set $\theta_i' = \theta$
    **for** i $= 0$ : k **do**
      $\theta_i' \leftarrow \theta_i' - \alpha \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}(D_i, g_{\theta_i'}(\hat{D}_i)))$
    **end for**
  **end for**
  $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}(D_i', g_{\theta_i'}(\hat{D}_i)))$
**end for**

end entity. But this agent will not work well under meta-learning setting, where the embedding of the new query relation is hard to be initialized. Our method replaces the query relation with a meta-encoder that encodes some meta information about the task, which is available on a new task. In the following parts, we will introduce more about

the reasoning agent and meta-encoder.

### Reasoning Agent

We use the policy proposed in [32], which is called MINERVA. They formulated this problem as a reinforcement learning problem. The state is defined as the combination of the query, the answer, and the current location (an entity in KB). But the answer is not observed, so the observation only includes the query and the current location. The actions are defined as the outgoing edges of the current location. The reward is +1 is reaching the answer, otherwise, it is 0.

The policy uses LSTM to encode the history information, i.e. the visited path.

$$\mathbf{h_t} = \text{LSTM}\left(\mathbf{h_{t-1}}, [\mathbf{a_{t-1}}; \mathbf{o_t}]\right)$$

where $\mathbf{h_{t-1}}$ is previous hidden state, $\mathbf{a_{t-1}}$ is the embedding for the chosen relation at time $t-1$, and $\mathbf{o_t}$ is the embedding of the current entity. The hidden state of the LSTM, $\mathbf{h_t}$ is then concatenated with the embedding of the current entity $\mathbf{o_t}$ and the query relation $\mathbf{r_q}$. The action distribution $\mathbf{d_t}$ is computed by applying softmax on the matching score between the action embedding and the projection of the concatenated embedding, i.e.,

$$\mathbf{d_t} = \text{softmax}\left(\mathbf{A_t}\left(\mathbf{W}_2\,\text{ReLU}\left(\mathbf{W_1}\left[\mathbf{h_t}; \mathbf{o_t}; \mathbf{r_q}\right]\right)\right)\right).$$

The model structure is the same as proposed in [32], which uses two linear layers ($\mathbf{W_1}$ and $\mathbf{W_2}$) to encode the observation. Next action is sampled from the action distribution $\mathbf{d_t}$.

### Meta-encoder

We can regard the embedding of query relation used in the above MINERVA model as the task identity. But when there comes a new task, there is no good way to find an initial embedding for the new query relation that fits into the reasoning model well. Therefore, we need another meta-encoder that leverage some meta-information about the new task and generate the embedding of query relation, based on which the model will be able to make reasonable outputs. Here we introduce two task-specific encoders to achieve this, neighbor encoder and path encoder.

**Neighbor Encoder** Given an instance, i.e., a triple $(e_s, r, e_t)$, we use the difference between the embedding of start entity $e_s$ and end entity $e_t$ as an representation of the query relation $r$ [16]. To better represent the entity, we borrow the idea of neighbor encoder from [19]. Let $\mathcal{N}_e$ denotes the neighbor of entity $e$. For each relation-entity pair $(r_i, e_i) \in \mathcal{N}_e$, We compute the feature representation $C_{r_i,e_i}$ as

$$C_{r_i,e_i} = W_c \left(v_{r_i} \oplus v_{e_i}\right) + b_c,$$

where $v_{r_i}$ and $v_{e_i}$ are the embedding for $r_i$ and $e_i$ respectively, $\oplus$ denotes concatenation, and $W_c$ and $b_c$ are parameters of a linear layer. Then the neighbor embedding of the given entity $e$ is computed as the average of the feature representations of all neighbors, i.e.,

$$\mathrm{NE}_e = \sigma\left(\frac{1}{|\mathcal{N}_e|} \sum_{(r_i,e_i)\in\mathcal{N}_e} C_{r_i,e_i}\right),$$

where $\sigma = \tanh$ is the activation function. Then the representation of the query relation is defined as the difference between the neighbor embedding of $e_s$ and $e_t$ like TransE [16]:

$$R_r = \mathrm{NE}_{e_t} - \mathrm{NE}_{e_s}.$$

| Dataset | # Entities | # Relations | # Triples | # Tasks | # Degree | |
|---|---|---|---|---|---|---|
| | | | | | average | median |
| FB15K-237 | 14505 | 237 | 239266 | 237 | 20.00 | 14 |
| NELL | 68272 | 358 | 181109 | 67 | 3.99 | 1 |

Table 2.1: Statistics of the datasets. # Entities, # Relations, # Triples, # Tasks denotes the number of entities, relations, triples, tasks in the corresponding dataset respectively. In the column of # Degree, average and median denote the average and median outgoing degree of each entity respectively.

**Path Encoder** The neighbor encoder needs to encode the neighbor as the representation for the start and end entity, and it will not work well when the number of neighbors is small. Thus we propose another encoder for this case called path encoder. Path encoder takes into consideration of the successful path in the graph, i.e., the reasoning path from start entity to end entity for a given query relation. Since not all the paths from start entity to end entity are meaningful, this path encoder is noisier than the neighbor encoder.

Let $\mathcal{P}_e$ denotes all the paths from start entity $e_s$ to end entity $e_t$. For any path $p_i \in \mathcal{P}_e$, we have $p_i = \{r_i^1, \cdots, r_i^n\}$, where $r_i^j$ is the selected relation at step $j$ in path $p_i$, and $n$ is the max length of reasoning path. We use LSTM [42] to encode each path:

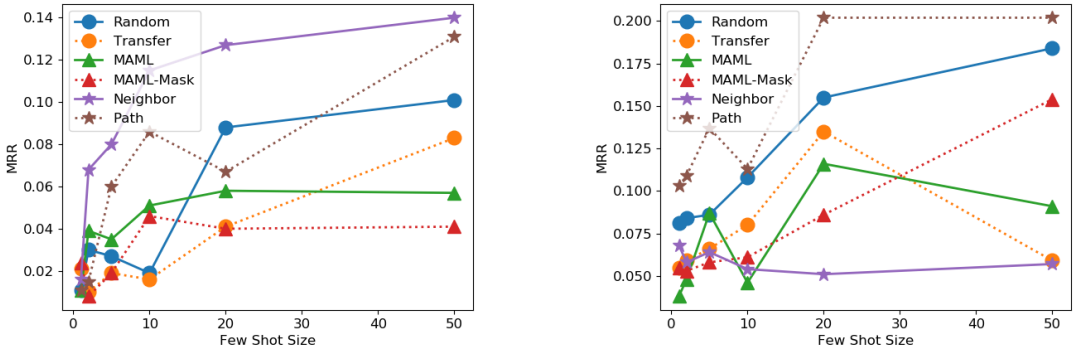
$$\mathbf{h_t} = \text{LSTM}(\mathbf{h_{t-1}}, \mathbf{r_i^j}),$$

where $\mathbf{h_t}$ is the hidden state of the LSTM at step $t$, and $\mathbf{r_i^j}$ is the embedding for relation $r_i^j$. The last hidden state $\mathbf{h_n}$ is used as the embedding $C_{p_i}$ for path $p_i$, i.e., $C_{p_i} = \mathbf{h_n}$. The final path embedding $\text{PE}_e$ for the given triple $(e_s, e, e_t)$ is average embedding of all the paths, i.e.,

$$\text{PE}_e = \frac{1}{|\mathcal{P}_e|} \sum_{p_i \in \mathcal{P}_e} C_{p_i}.$$

| Setting | Method | FB15K-237 | | | | NELL | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Hits@1 | Hits@3 | Hits@10 | MRR | Hits@1 | Hits@3 | Hits@10 | MRR |
| Full Data | MINERVA | .124 | .146 | .187 | .142 | .137 | .176 | .202 | .163 |
| Best | | | | | *Baselines* | | | | |
| | Random | .017 | .028 | .043 | .027 | .047 | .100 | .165 | .086 |
| | Transfer | .010 | .012 | .054 | .019 | .041 | .070 | .128 | .066 |
| | MAML | .021 | .041 | .052 | .035 | .067 | .086 | .139 | .087 |
| | MAML-Mask | .009 | .023 | .045 | .019 | .032 | .054 | .080 | .058 |
| | | | | | *Ours* | | | | |
| | Neighbor | **.065** | **.073** | **.128** | **.080** | .045 | .066 | .106 | .064 |
| | Path | .041 | .067 | .101 | .060 | **.108** | **.141** | **.200** | **.137** |
| Initial | | | | | *Baselines* | | | | |
| | Random | .000 | .000 | .005 | .002 | .021 | .074 | .105 | .056 |
| | Transfer | .000 | .005 | .023 | .006 | .037 | .055 | .077 | .051 |
| | MAML | .005 | .005 | .023 | .010 | .017 | .031 | .054 | .032 |
| | MAML-Mask | .000 | .014 | .045 | .012 | .021 | .050 | .081 | .043 |
| | | | | | *Ours* | | | | |
| | Neighbor | **.043** | **.054** | **.092** | **.056** | .026 | .047 | .091 | .045 |
| | Path | .000 | .005 | .058 | .012 | **.082** | **.109** | **.164** | **.104** |

Table 2.2: The results on 5-shot experiments. We also report the performance of MINERVA on these tasks using full data for better comparison. Full Data denotes using MINERVA algorithm on these tasks with full training data. Best denotes the best performance for each method after fine-tuning, and Initial denotes the performance of method at the initial point. We report the average performance on the meta-test tasks. Best result for each evaluation matrix is marked in bold.




(a) FB15K-237 (b) NELL

Figure 2.2: The change of the performance with the size of few-shot samples for each method. Here we choose the size to be $1, 2, 5, 10, 20, 50$. MRR of each model after fine-tuning is reported.

## 2.5 Experiments

To verify the effectiveness of the proposed methods, we compare it with several baselines on two knowledge completion datasets, FB15K-237 [39], and NELL [43]. In the following part, we will introduce how we construct the meta-learning setting for knowledge graph reasoning and the baselines we use, then we will show the main results and other analytic experiments.

### 2.5.1 Datasets and Settings

We construct the meta-learning setting from two well-known knowledge completion datasets: FB15K-237 [39] and NELL [43]. FB15K-237 is created from original FB15K by removing various sources of test leakage. Every relation in the training set of FB15K-237 is regarded as an individual task. For the NELL dataset, we use the modified version from [19], which chooses relations with more than 50 triples, and less than 500 triples as one-shot tasks. Here we used those selected tasks as meta-learning tasks. The statistics of the two datasets are shown in Table 2.1.

Let $D_{\text{train}}$, $D_{\text{dev}}$, and $D_{\text{test}}$ denotes the training data, validation data and test data in original dataset such as FB15K-237. We choose some tasks with positive transfer (task that has better performance when training together with other tasks than training solely) as meta-dev and meta-test tasks. More specifically, we choose task with at least 0.04 and 0.02 positive transfer on FB15K-237 and NELL dataset respectively, from which we only keep tasks with more than 20 samples in the dev set. Note that 0.04 and 0.02 are carefully chosen threshold so that we can get enough tasks with reasonable positive transfer. Through this way, we get 5/10 and 4/7 relations for meta-dev/meta-test on FB15K-237 and NELL respectively, and other relations left are used for meta-training. We denote the partitioned relation set as $R_{\text{meta-train}}/R_{\text{meta-dev}}/R_{\text{meta-test}}$, and each relation

has its own training/test data.

### 2.5.2 Baselines and Hyper-parameters

We compare our methods with the following baselines. **Random** method trains a separate model for each task from random initialization. **Transfer** method will learn an initial model by using samples from $D_{\text{train}}^{\text{meta-train}}$. **MAML** uses the training framework of MAML to learn an initial point, and the task identity (the query relation) is given. **MAML-Mask** uses the same training framework as MAML, the difference is that we mask the task identity by setting the query relation for all tasks to be 0. **Neighbor** and **Path** method means we use the neighbor encoder and path encoder to encode the task-specific information respectively.

We tuned the hyper-parameters for all the baselines and our methods, and they are set as follows. For Transfer, the batch size in the pre-training phase is set to be 128. For MAML, MAML-Mask, Neighbor, and Path, the batch size is set to be 5. For Path, 1 adaption step is applied to compute the updated parameters, and $\alpha = 0.01, \beta = 0.001$. For Neighbor, MAML, and MAML-Mask, 5 and 1 adaption steps are applied on FB15K-237 and NELL respectively, and $\alpha = 0.01$ when the number of adaption step $k = 1$, $\alpha = 0.001$ when $k = 5$, and $\beta = 0.001$. Other parameters are set as default as in [32].

### 2.5.3 Results

We conduct our experiments under 5-shot learning setting, i.e., there are 5 training samples for each task in $R_{\text{meta-dev}}$ and $R_{\text{meta-test}}$. We use the mean reciprocal rank (MRR) and Hits@K to evaluate each model. For each method, we will first fine-tune and test the initial model on meta-dev tasks, through which we choose the number of fine-tune steps and fix it on meta-test tasks. For example, if a model has the best performance after 5

fine-tune steps on meta-dev tasks, then the model will be tested after 5 fine-tune steps on meta-test tasks. We report the best performance on meta-test tasks for each method in Table 2.2 as Best group. We also list the results using full data for better comparison. From the results, we can see that neighbor encoder and path encoder achieves the best performance on FB15K-237 and NELL dataset respectively. It is reasonable that neighbor encoder does not perform well on NELL dataset since the median outgoing degree on this dataset is only 1. We also note that path encoder outperforms other baselines on FB15K-237, which verify the consistent effectiveness of the task-specific encoder. While other baselines do not show much difference as the simple Random baseline, sometimes they even underperform Random baseline.

In order to show that our model can have better initial point than others, we report the performance of the initial point without any training in Table 2.2 as Initial group. We notice that the baselines have very poor initial performances on FB15K-237, which is reasonable since the model has never seen the new relation. From the results, we can see that the neighbor encoder and path encoder achieves much better initial point than other baselines in FB15K-237 and NELL respectively. The path encoder has a fair performance which is similar to the best of the baselines MAML-Mask, we think the reason that path encoder does not perform very well is the path encoder is noisier than neighbor encoder as we mentioned before.

### 2.5.4 Few-shot Size

To investigate the impact of the few-shot size on the performance of the model, we evaluate the model using various few-shot size: $1, 2, 5, 10, 20, 50$. The results are shown in Figure 2.2. From the results, we can see that for MAML and MAML-Mask, their performances remain nearly the same after the size reaches 10 on FB15K-237 dataset.

| Setting | FB15K-237 | | | |
|---|---|---|---|---|
| | Hits@1 | Hits@3 | Hits@10 | MRR |
| Encoder-1-shot | .047 | .058 | .117 | .064 |
| Encoder-50-shot | **.049** | **.070** | **.128** | **.069** |
| No-encoder | .008 | .035 | .084 | .032 |

Table 2.3: The comparison of performance for model with different initialization on FB15K-237 dataset. Encoder-1-shot and Encoder-50-shot denotes using neighbor encoder with 1 and 50 samples. No-encoder means using a random initialization. We report the average performance on meta-test tasks. Best result for each evaluation matrix is marked in bold.

The performance of MAML is not stable on NELL dataset, while MAML-Mask keeps increasing. Both methods underperform the Random baseline when the size increases. For Transfer method, its performance increases with the few shot size on FB15K-237, but there is a huge drop on NELL when the size is 50, which indicates it is not stable enough, and sensitive to the noise in the data. The neighbor encoder has the best performance on FB15K-237 dataset, but not well on NELL due to the small neighbor size. Path encoder seems to be less stable compared with neighbor encoder since there is performance drop once on both datasets, but it achieves the best performance on NELL and second-best performance when size is larger than 5 except 20.

### 2.5.5 Ablation Study

To verify the effectiveness of the encoder, we compare the model using task-specific initialization with the model using random initialization at the initial point. We choose the neighbor encoder on FB15K-237 dataset to conduct the ablation study. The comparison results are shown in Table 2.3. The three models in the table use the same reasoning model, the only difference is the task representation. Encoder-1-shot and Encoder-50-shot apply neighbor encoder to generate the task representation using 1 and 50 samples respectively, while No-encoder uses a randomly initialized representation. By comparing

Encoder-1-shot with No-encoder, we can see that the model can achieve much better performance through the way of encoding task-related information, even using only one sample, which also indicates the generated task representations are meaningful. Also, better initialization can be achieved when using more samples, since the performance of Encoder-50-shot is better than that of Encoder-1-shot.

## 2.6 Conclusion

In this chapter, we consider multi-hop reasoning over knowledge graphs under few-shot learning setting, where limited samples are available on new tasks. We improve upon MAML by using a meta-encoder to encode task-specific information. Through this way, our method can create a task-dependent initial model that better fits the target task. Neighbor encoder and path encoder are proposed for our problem. Experiments on FB15K-237 and NELL under meta-learning setting show that our task-specific meta-encoder yields a better initial point and outperforms other baselines.

# Chapter 3

# Self-Supervised Learning on Document Level Structure

## 3.1 Introduction

Extractive summarization aims at shortening the original article while retaining the key information through the way of selection sentences from the original articles. This paradigm has been proven effective by many previous systems [44, 45, 46, 47]. In order to decide whether to choose a particular sentence, the system should have a global view of the document context, e.g., the subject and structure of the document. However, previous works [48, 49, 50, 51] usually directly build an end-to-end training system to learn to choose sentences without explicitly modeling the document context, counting on that the system can automatically learn the document-level context.

We argue that it is hard for these end-to-end systems to learn to leverage the document context from scratch due to the challenges of this task, and a well pre-trained embedding model that incorporates document context should help on this task. In recent years, extensive works [52, 53, 54, 55, 56, 57, 58, 59, 60] have been done in learning the word

| | |
|---|---|
| Masked Paragraph | Last week, I went to attend a one-day meeting.<br>I booked the flight in advanced.<br>[masked sentence]<br>The earliest next flight will be a few days later.<br>I had to use the online discussion instead. |
| Candidate Sentences | But the flight was cancelled due to the weather.<br>But I lost my passport.<br>The meeting was cancelled.<br>The weather is good today. |

Figure 3.1: An example for the *Mask* pre-training task. A sentence is masked in the original paragraph, and the model is required to predicted the missing sentence from the candidate sentences.

or sentence representations, but most of them only use a sentence or a few sentences when learning the representation, and the document context can hardly be included in the representation. Hence, we introduce new pre-training methods that take the whole document into consideration to learn the contextualized sentence representation with self-supervision.

Self-supervised learning [61, 62, 63, 64] is a newly emerged paradigm, which aims to learn from the intrinsic structure of the raw data. The general framework is to construct training signals directly from the structured raw data, and use it to train the model. The structure information learned through the process can then be easily transformed and benefit other tasks. Thus self-supervised learning has been widely applied in structured data like text [65, 66, 55, 56, 67] and images [62, 63, 64, 68]. Since documents are well organized and structured, it is intuitive to employ the power of self-supervised learning to learn the intrinsic structure of the document and model the document-level context for the summarization task.

In this chapter, we propose three self-supervised tasks (*Mask*, *Replace* and *Switch*), where the model is required to learn the document-level structure and context. The knowledge learned about the document during the pre-training process will be transferred and benefit on the summarization task. Particularly, The *Mask* task randomly masks

some sentences and predicts the missing sentence from a candidate pool; The *Replace* task randomly replaces some sentences with sentences from other documents and predicts if a sentence is replaced. The *Switch* task switches some sentences within the same document and predicts if a sentence is switched. An illustrating example is shown in Figure 3.1, where the model is required to take into account the document context in order to predict the missing sentence. To verify the effectiveness of the proposed methods, we conduct experiments on the CNN/DM dataset [69, 70] based on a hierarchical model. We demonstrate that all of the three pre-training tasks perform better and converge faster than the basic model, one of which even outperforms the state-of-the-art extractive method NeuSum [50].

The contributions of this work include:

• To the best of our knowledge, we are the first to consider using the whole document to learn contextualized sentence representations with self-supervision and without any human annotations.

• We introduce and experiment with various self-supervised approaches for extractive summarization, one of which achieves the new state-of-the-art results with a basic hierarchical model.

• Benefiting from the self-supervised pre-training, the summarization model is more sample efficient and converges much faster than those trained from scratch.

## 3.2 Model and Pre-training Methods

### 3.2.1 Basic Model

As shown in Figure 3.2, our basic model for extractive summarization is mainly composed of two parts: a sentence encoder and a document-level self-attention module. The

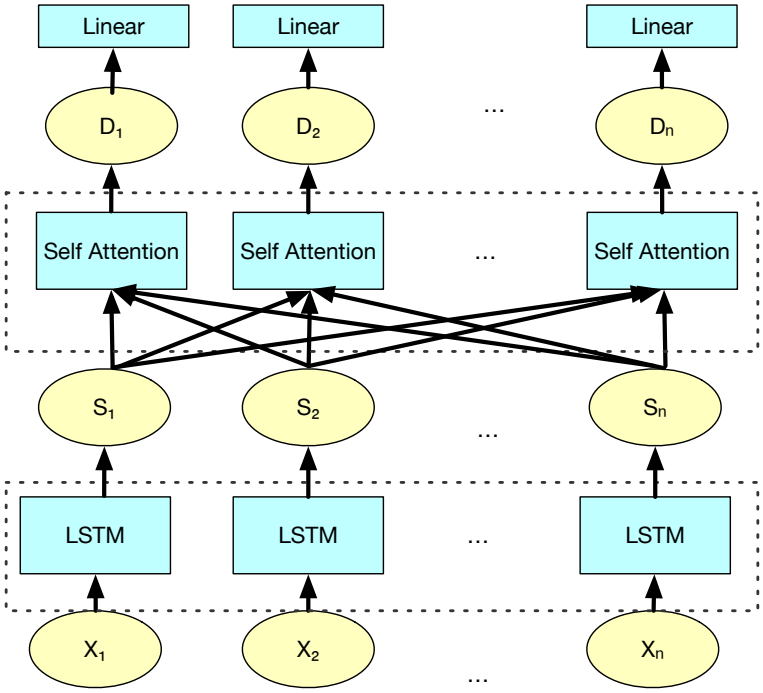



Figure 3.2: The structure of the Basic Model. We use LSTM and self-attention module to encode the sentence and document respectively. $X_i$ represent the word embedding for sentence $i$. $S_i$ and $D_i$ represent the independent and document involved sentence embedding for sentence $i$ respectively.

sentence encoder is a bidirectional LSTM [42], which encodes each individual sentence $X_i$ (a sequence of words) and whose output vector at the last step is viewed as the sentence representation $S_i$. Given the representations of all the sentences, a self-attention module [71] is employed to incorporate document-level context and learn the contextualized sentence representation $D_i$ for each sentence.[1] Finally, a linear layer is applied to predict whether to choose the sentence to form the summary.

### 3.2.2 Self-supervised Pre-training Methods

In this section, we will describe three self-supervised pre-training approaches. Through solving each pre-training task, the model is expected to learn the document-level contextualized sentence embedding model from the raw documents, which will then be used to solve the downstream summarization task. Note that we are only pretraining the sentence encoder and document-level self-attention module of the basic model for extractive summarization.

**Mask** Similar to the task of predicting missing word, the *Mask* task is to predict the masked sentence from a candidate pool. Specifically, we first mask some sentences within a document with the probability $P_m$ and put these masked sentences $(\mathbf{x_1^m}, \mathbf{x_2^m}, \cdots, \mathbf{x_t^m})$ into a candidate pool $T^m$. The model is required to predict the correct sentence from the pool for each masked position $i$. We replace the sentence in the masked position $i$ with a special token $\langle unk \rangle$ and compute its document contextualized sentence embedding $D_i$. We use the same sentence encoder in the basic model to obtain the sentence embedding $S^m$ for these candidate sentences in $T^m$. We score each candidate sentence $j$ in $T^m$ by

[1] We leave the combination of different architectures such as replacing the self-attention module with LSTM for future work.

using the cosine similarity:

$$\Theta(i, j) = \cos(D_i, S_j^m)$$

To train the model, we adopt a ranking loss to maximize the margin between the gold sentence and other sentences:

$$\ell_m = \max\{0, \gamma - \Theta(i, j) + \Theta(i, k)\}$$

where $\gamma$ is a tuned hyper-parameter, $j$ points to the gold sentence in $T^m$ for the masked position $i$, and $k$ points to another non-target sentence in $T^m$.

**Replace** The *Replace* task is to randomly replace some sentences (with probability $P_r$) in the document with sentences from other documents, and then predict if a sentence is replaced. Particularly, we use sentences from $10,000$ randomly chosen documents to form a candidate pool $T^r$. Each sentence in the document will be replaced with probability $P_r$ by a random sentence in $T^r$. Let $C_r$ be the set of positions where sentences are replaced. We use a linear layer $f_r$ to predict if the sentence is replaced based on the document embedding $D$, and minimize the MSE loss:

$$\ell_r = \text{MSE}(f_r(D_i),\ y_i^r)$$

where $y_i^r = 1$ if $i \in C_r$ (i.e., the sentence in position $i$ has been replaced), otherwise $y_i^r = 0$.

**Switch** The *Switch* task is similar to the *Replace* task. Instead of filling these selected sentences with sentences out of the document, this task chooses to use sentences within the same document by switching these selected sentences, i.e., each selected sentence will be put in another position within the same document. Let $C_s$ be the set of positions

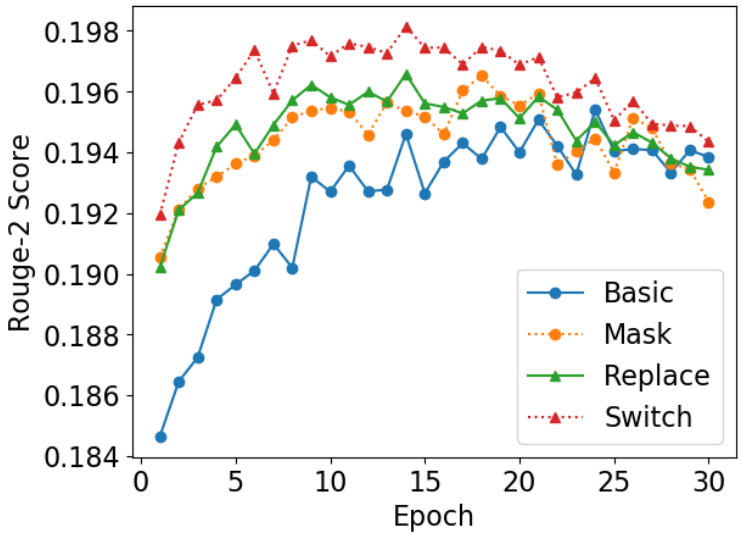



Figure 3.3: This figure shows the Rouge-2 score for each pre-training method and the basic model on the development set during the training process.

where the sentences are switched. Similarly, we use a linear layer $f_s$ to predict if a sentence is switched and minimize the MSE loss:

$$\ell_s = \text{MSE}(f_s(D_i),\ y_i^s)$$

where $y_i^s = 1$ if $i \in C_s$, otherwise $y_i^s = 0$.

## 3.3 Experiment

To show the effectiveness of the pre-training method (**Mask**, **Replace** and **Switch**), we conduct experiments on the commonly used dataset CNN/DM [69, 70], and compare them with a popular baseline **Lead3** [72], which selects first three sentences as the summary, and the state-of-the-art extractive summarization method **NeuSum** [50], which jointly scores and selects sentences using pointer network.

### 3.3.1 On CNN/DM Dataset

**Model and training details** We use the rule-based system from [50] to label sentences in a document, e.g., sentences to be extracted will be labeled as 1. Rouge score[2] [73] is used to evaluate the performance of the model, and we report Rouge-1, Rouge-2, and Rouge-L as in prior work. We use the pre-trained glove embedding [52] with 100 dimensions to initialize the word embedding. A one-layer bidirectional LSTM [42] is used as the sentence encoder, and the size of hidden state is 200. A 5-layer Transformer encoder [71] with 4 heads is used as the document-level self-attention module. A linear classification layer is used to predict whether to choose the sentence.

The training process consists of two phrases. First, we use the pre-training task to pre-train the basic model using the raw article from the CNN/DM dataset without labels. Second, we fine-tune the pre-trained model for the extractive summarization task using the sentence labels. The learning rate is set as 0.0001 in the pre-training phase and 0.00001 in the fine-tune phase. We train each pre-training task until it is converged or the number of training epochs reaches the upper bound 30. We set the probability to mask, replace or switch sentences as 0.25.

**Results** We show the Rouge score on the development set during the training process in Figure 3.3, and present the best Rouge score for each method in Table 3.1. All pre-training methods improve the performance compared with the Basic model. Especially, Switch method achieves the best result on all the three evaluations compared with other pre-training methods, and is even better than the state-of-the-art extractive model NeuSum[3].

In the terms of convergence, the Mask, Replace and Switch task takes $21, 24, 17$ epochs

[2]We use PyRouge `https://pypi.org/project/pyrouge/` to compute the Rouge score.

[3]We use code from `https://github.com/magic282/NeuSum` to train the model, and evaluate it using our evaluation script.

| Method | Rouge-1 | Rouge-2 | Rouge-L |
|---|---|---|---|
| Basic | 41.07 | 18.95 | 37.56 |
| LEAD3 | 39.93 | 17.62 | 36.21 |
| NeuSum | 41.18* | 18.84 | 37.61 |
| Mask | 41.15* | 19.06* | 37.65* |
| Replace | 41.21* | 19.08* | 37.73* |
| Switch | **41.36** | **19.20** | **37.86** |
| SentEnc | 41.17* | 19.04* | 37.69* |
| Switch 0.15 | 41.35* | 19.18* | 37.85* |
| Switch 0.35 | 41.27* | 19.12* | 37.77* |

Table 3.1: The Rouge [73] scores for the basic model, baselines, pre-training methods, and analytic experiments. All of our Rouge scores have a 95% confidence interval of at most ±0.25 as reported by the official ROUGE script. The best result is marked in bold, and those that are not significantly worse than the best are marked with *.

in the training phase respectively, and $18, 13, 9$ epochs to achieve the best performance in the fine-tune phase. The basic model takes 24 epochs to obtain the best result. From Figure 3.3, we can see that the *Switch* task converges much faster than the basic model. Even adding on the epochs taken in the pre-training phase, Switch method (26 epochs) takes roughly the same time as the Basic model (24 epochs) to achieve the best performance.

### 3.3.2 Ablation Study

**Reuse only the sentence encoder** Our basic model has mainly two components: a sentence encoder and a document-level self-attention module. The sentence encoder focuses on each sentence, while document-level self-attention module incorporates more document information. To investigate the role of the document-level self-attention module, we only reuse the sentence encoder of the pre-train model, and randomly initialize the document-level self-attention module. The results is shown in Table 3.1 as SentEnc. We can see that using the whole pre-training model (Switch 0.25) can achieve better

performance, which indicates the model learn some useful document-level information from the pre-training task. We notice that only using the sentence encoder also get some improvement over the basic model, which means that the pre-training task may also help to learn the independent sentence representation.

**On the sensitivity of hyper-parameter** In this part, we investigate the sensitivity of the model to the important hyper-parameter $P_w$, i.e., the probability to switch sentences. In the previous experiment, we switch sentences with probability 0.25. We further try the probability of 0.15 and 0.35, and show the results in Table 3.1 as Switch 0.15 and Switch 0.35. We can see Switch 0.15 achieve basically the same result as Switch 0.25, and Switch 0.35 is slightly worse. So the model is not so sensitive to the hyper-parameter of the probability to switch sentences, and probability between 0.15 and 0.25 should be able to work well.

## 3.4 Conclusion

In this chapter, we propose three self-supervised tasks to force the model to learn about the document context, which will benefit the summarization task. Experiments on the CNN/DM verify that through the way of pre-training on our proposed tasks, the model can perform better and converge faster when learning on the summarization task. Especially, through the Switch pre-training task, the model even outperforms the state-of-the-art method NeuSum [50]. Further analytic experiments show that the document context learned by the document-level self-attention module will benefit the model in summarization task, and the model is not so sensitive to the hyper-parameter of the probability to switch sentences.

# Chapter 4

# Learning with Tutor for Out-of-Distribution Problem

## 4.1 Introduction

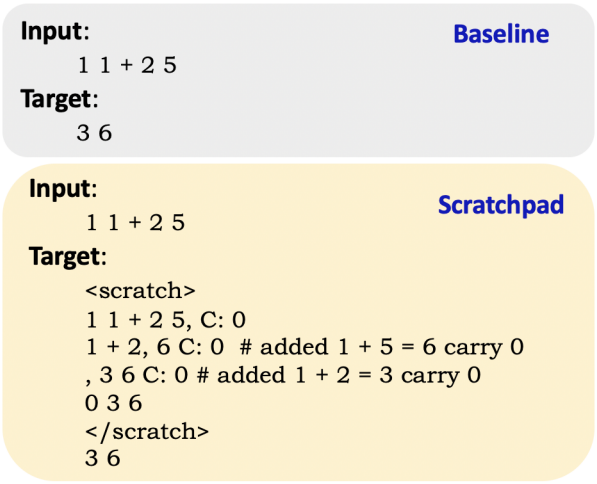



Figure 4.1: Examples of addition: the baseline setting (top) and Scratchpad [74] with intermediate computation steps (bottom). A similar method with more detailed demonstration is introduced in [75].

Transformer-based large pretrained Language Models, such as GPT3 and T5 [76, 3,

77], have been widely used as few-shot learners in many NLP tasks. Recent work even finds these models can achieve state-of-the-art performance in arithmetic and symbolic reasoning [74, 78]. Although these models exhibit surprisingly impressive capabilities in complex arithmetic reasoning tasks, such as MultiArith [79] and GSM8k [80], it has also been pointed out that they tend to make certain calculation errors and perform significantly worse when the number of math operations increases in equations [78]. GPT3 [3] displays strong proficiency in 2-digit arithmetic addition, but struggles in arithmetic addition on numbers with more than three digits. Previous research [81] also observe that the fine-tuned T5 model can not correctly add or subtract arbitrarily long numbers. Larger models might perform better on the testing data, but worse on numbers that are longer than the training data (out-of-distribution, OOD) [81]. However, even with the largest T5 model they experimented, the out-of-distribution (OOD) accuracy is not as high as the in-distribution accuracy, and increasing the training data does not improve OOD generalization beyond a critical amount.

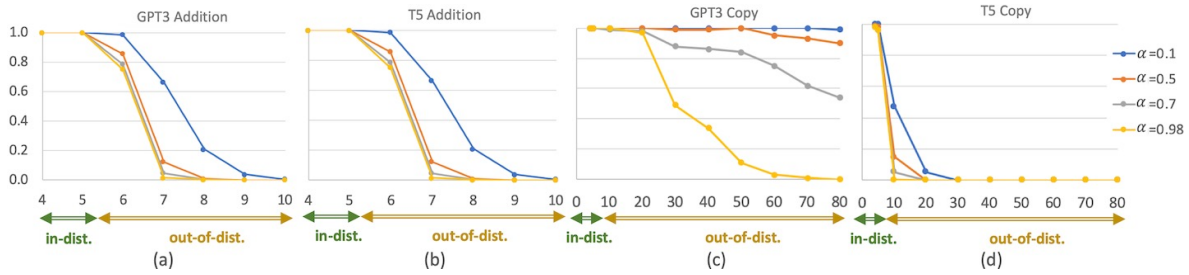



Figure 4.2: The horizontal axis is the number of digits and the vertical axis is the accuracy. The prompts for GPT3 consist of 4 examples. The T5 models are trained on 1-5 digits of up to 2,000 examples and each training example consists of random numbers in the format of 2 4 1. In-dist: in-distribution. Out-of-dist.: out-of-distribution (OOD). In-distribution refers to training on up to k-digit numbers and testing on up to k-digit numbers while out-of- distribution refers to training on up to k-digit numbers and testing on numbers with more digits. $\alpha$ indicates the repetition level of the examples. An example $x_1 \cdots x_n$ with $n$ digits are sampled with the next digit probability $p(x_{i+1}|x_i) = \alpha$, when $x_{i+1} = x_i$; otherwise, $(1 - \alpha)/9$. Larger $\alpha$ indicates a higher repetition level.

Figure 4.1 shows two possible addition exemplars for LMs. Addition can be considered

as a basic arithmetic operation and a simple symbolic manipulation task. The scratchpad version gives more details on how humans do basic arithmetic. Previous research [74] shows that with more fine-grained demonstrations, the accuracy of addition can be improved dramatically with fine-tuning. Yet, it still can not achieve 100% on OOD data, even with thousands of training data points provided. Figure 4.2 shows the performance of GPT-3 and T5 on addition using the scratchpad version of training data. The problem becomes more severe when there are repeating digits in the addition operands.

As the performance drops with repeating digits, we suspect that LMs might not handle the repeating symbols well. Figure 4.2 illustrates the performance of GPT-3 and T5 on the copy task, one of the simplest symbolic manipulation operations. GPT-3 and T5 still can not perform well on OOD. We further do a preliminary experiment where a T5 model is fine-tuned using the data containing repeating numbers of up to 80 digits, T5 still can not achieve 100% in-distribution accuracy on long repeating digits. The results indicate that there are two problems intervening: Transformers are not good at handling repeating symbols and OOD generalization. The repeating symbols can also be a problem even for in-distribution data.

Why do large pretrained LMs that can do complex language generation fail on a simple symbolic manipulation task?

In this chapter, we investigate the potential causes behind this phenomenon. We believe that overcoming the aforementioned limitations is of critical importance for the future application of Transformer-based LMs to reasoning-intensive tasks. What are the necessary steps to take to significantly improve or even approach 100% accuracy on these simple but fundamentally important induction tasks? We examine a set of possible mitigation solutions including fine-grained computation steps, positional markers, and LMs with callable programs.

Since incorporating computation steps improves the OOD generalization in arithmetic

addition [74], one possible direction is to provide more fine-grained computation steps in the fine-tuning data or the few-shot prompt. However, it may not be sufficient to alleviate the problem of repeating numbers. When a human does addition, the position of each digit is used to differentiate the repeating digits. However, the self-attention mechanism in the Transformer may not tell which "1" is referred to in the input. This prompts us to explore using positional markers to differentiate the important tokens. Using these two methods to augment the reasoning process, we find that the performance of pretrained LMs still can not reach satisfying results. Then we resort to a method where the copy operation is implemented as a primitive function and explore whether the LM can further boost its performance.

We experiment with three symbolic manipulation tasks: copying, reversing, and addition. Experimental results show that although generalization in these symbolic manipulation tasks is straightforward for humans, it is still challenging for LMs, and none of these mitigation methods fully solves the problems. In the end, we introduce LMs with tutor which demonstrates every single step of teaching, pinpointing where these digits come from. LMs with tutor is able to deliver 100% accuracy in situations of OOD and repeated symbols. In this design, LMs are used to generate actions that mimic operations in multiple tape Turing machines, rather than the intermediate results. These actions generate the intermediate results on tapes. We hope this could shed light on the capability of Transformer-based LMs in addition to providing large training datasets or scaling up the size of these models.

To conclude, our main contributions are:

- We identify a set of simple symbolic manipulation tasks and uncover the limitations of the LMs in arithmetic and symbolic induction.

- We examine a set of potential techniques including positional markers, fine-grained

computation steps, and LMs with callable programs. Though these techniques could mitigate the limitations of the LMs, none of them can completely solve the generalization problem.

- Finally, we demonstrate that LMs with tutor is able to deliver 100% accuracy in situations of OOD and repeated symbols. Our analysis could inspire new thoughts to overcome the limitation of LMs in symbolic manipulation.

## 4.2 Related Work

Previous research in using LMs for symbolic induction improves the model performance in the following three directions.

**Large Pretrained Language Models**: GPT3 [3] exhibits strong proficiency on 2-digit addition and subtraction using simply few-shot prompting, without any task-specific training. Furthermore, the larger the LM, the better the performance. Following GPT3, PaLM [82] further scale the Transformer-based LMs to a 540-billion parameter model, called Pathways Language Model (PaLM). Same as GPT-3 [3], PaLM [82] find that scaling the LMs consistently results in better arithmetic reasoning ability with few-shot prompting. However, the reasoning ability of the large LMs is still limited. GPT3 struggles with 3-digit arithmetic and with direct prompting, even 540B PaLM can not achieve high performance on complex tasks requiring multi-step reasoning. Therefore CoT [78] propose to augment the large pretrained LMs with the following prompting method.

**Chain-of-Thought Prompting**: This prompting method provides a few chain-of-thought demonstrations, which is a series of intermediate reasoning steps, as exemplars in the prompting. Therefore, given a complex reasoning task, the model is allowed to calculate the intermediate results step-by-step before generating the final answer. With

chain-of-thought prompting, a complex reasoning task is decomposed into a list of simple operations and LMs can derive these operations one by one. Previous research [83] adopt faithful explanations that accurately represent the reasoning process behind solving a math word problem. CoT [78] show that combining chain-of-thought prompting and a sufficiently large LM, 540B PaLM, can significantly improve the LMs' reasoning ability on complex tasks, such as math word problems.

**Fine-tuning with Large Training Datasets**: Instead of few-shot prompting, another direction is to fine-tune large LMs with a sufficient amount of training data. In [81], they fine-tune T5 with different ways of representing numbers, but even with the best-performing representation, the fine-tuned model can not achieve as good accuracy on out-of-distribution testing examples as in-distribution testing examples. In [74], they propose to use Scratchpad to improve the out-of-distribution accuracy. Scratchpad combines step-by-step reasoning with fine-tuning. The training examples include the intermediate steps of an algorithm in target, so the model is trained to generate not only the final answer, but also the intermediate steps, which is similar to chain-of-thought, but requires more training data. In [74], they show that using the training data augmented with intermediate steps significantly improves the model performance, but even with 100k augmented training examples for the addition task, the fine-tuned 1B LM still does not perform well on out-of-distribution addition.

Our work is also related to [84], which extends the capabilities of Recurrent Neural Networks to two simple symbolic manipulation tasks, copy and sort, by augmenting the model with external memory resources. Instead of using hundreds of thousands of training examples, we focus on large pretrained LMs with few-shot prompting or fine-tuning settings in this work.

## 4.3 Observations

We first analyze the difficulty of generalizing the copy operation, one of the most fundamental, simplest symbolic manipulation operations. We start by copying random numbers. For GPT3, we augment each testing example with the few-shot prompt as shown in Figure 4.3.

```
copy: 8 3 2 2
result: 8 3 2 2
copy: 7 7 7 7
result: 7 7 7 7
copy: 3 9 4 3 2
result: 3 9 4 3 2
copy: 6 6 6 6 6
result: 6 6 6 6 6
```

Figure 4.3: The prompt for GPT3 on the copy task.

We also fine-tune a T5 model for copying. The training data follows the same format as above and consists of random numbers of up to 5 digits. We first evaluate the prompted GPT3 and fine-tuned T5 on copying random numbers of up to 80 digits ($\alpha = 0.1$ in Figure 4.2). GPT3 achieves nearly 100% accuracy on all the testing examples of up to 80 digits when $\alpha$ is 0.1. The finetuned T5 does not generalize well beyond 7 digits, and it achieves nearly 100% accuracy on all the testing examples of 7 digits, except for a few error cases as follows:

‘‘input’’: copy: ··· 9 8 9 8 9 4 ···

‘‘pred’’: ··· 9 8 9 4 ···

‘‘input’’: copy: ··· 6 0 6 0 6 5 ···

‘‘pred’’: ··· 6 0 6 5 ···

A common feature of these error cases is that they all contain consecutive repeating numbers and the model tends to mistakenly skip part of them or over-replicate them.

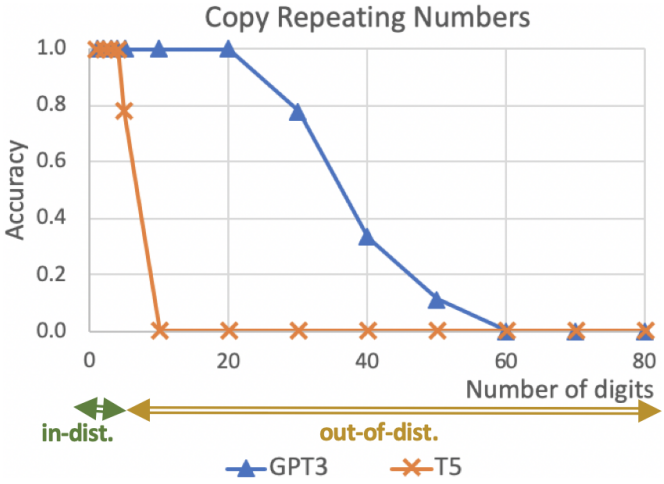



Figure 4.4: An illustration of GPT3 and T5 performance on copying repeating numbers.

Therefore we further do a copying task where the testing examples consist only of repeated digits, such as `copy: 2 2 2 2 2`. The results are shown in Figure 4.4. Although T5 performs well at copying random numbers of up to 7 digits, its accuracy at copying 5-digit repeating numbers drops below 80%. Similarly, the prompted GPT3 can not generalize to copying more than 30 repeating digits. Both GPT3 and T5 are Transformer-based LMs, which use the self-attention mechanism. When copying numbers, the models are required to use self-attention to locate the next digit to copy. When copying random numbers without repeated digits, it usually would be sufficient to locate the next digit by comparing the previous few digits. However, when copying repeated digits, this mechanism no longer works since all the previous digits are the same. Instead, the model needs to locate the next digit either by counting the repetitive digits or by remembering the previous position. Therefore, the results suggest that the Transformer-based LMs, such as GPT3 and T5, might have limited capability of locating in symbolic manipulation.

## 4.4 Mitigation Methods

In order to mitigate the limitations, we examine a few potential solutions.

### 4.4.1 Positional Markers

We first explore possible methods to mitigate the problem of repeating numbers. We introduce two types of positional markers: implicit positional markers and explicit ones.

As stated above, LMs tend to make mistakes when the input contains repeating numbers. When humans deal with repeating numbers in basic arithmetic, we usually use indices to distinguish those digits in the input. Positional encoding in large LMs is closely related to this human practice. Most Transformer-based LMs encode the positional information into positional vectors and add each of them to the corresponding word vector. Although large LMs have already incorporated positional encoding in the model architecture (Figure 4.5), results in Figure 4.2 indicate that the positional encoding commonly used in large LMs may not be sufficient to locate each repeating digit effectively.

Instead of representing each token by the sum of its contextual token embedding and the position embedding, DeBERTa [85] represents each token with a token embedding and a position embedding, respectively, and the attention weights are computed using disentangled matrices based on both embeddings, respectively (Figure 4.5). In other words, the self-attention in DeBERTa is disentangled. With the disentangled relative position embeddings, the attention scores between tokens depend not only on the content but also on the relative position between the tokens, so the disentangled relative position embeddings act as implicit position markers within DeBERTa, which might make it easier for the model to learn the latent position relationship in the training data of the symbolic manipulation tasks.

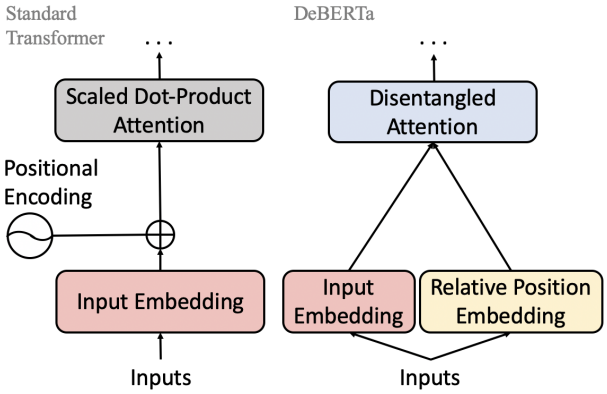



Figure 4.5: An illustration of standard Transformer attention (left) and DeBERTa disentangled attention (right).

Although DeBERTa uses disentangled attention mechanism, it was not originally introduced to enhance the locating capability of LMs, so no pretraining task was specifically proposed for training the position embeddings in DeBERTa. This may potentially lead to limited generalization ability of DeBERTa on the induction tasks requiring accurate locating.

Rather than relying on implicit positional markers, another, more straightforward approach is to add explicit positional markers in the input for the model. For example, the input string `2 2 2` is augmented with positional markers `A, B, C, ⋯`. We explore two methods of adding explicit positional markers:

**Ordered marker**: The markers are inserted into the input in order. `2 2 2` → `A 2 B 2 C 2`

**Random marker**: The markers are inserted into the input in random order. `2 2 2` → `E 2 X 2 J 2`

With the explicit positional markers, each repeating `2` becomes different for the model. When doing symbolic manipulation, the Transformer-based LMs can easily locate the digit by recognizing the explicit positional markers. Essentially, adding explicit positional markers breaks the repeating numbers into a non-repeating input sequence. This method

is also related to pointer networks [86], which uses attention as a pointer to select the position indexes of the input tokens as the output. A hybrid pointer-generator network can also be leveraged to copy number from the source text, while retaining the ability to produce new numbers through the generator [87]. Compared with implicit markers, explicit markers provide more direct and clearer location information in text format. However, similar to the implicit positional markers, whether using the explicit positional markers can generalize to arbitrary length or unseen markers is still questionable.

### 4.4.2 Fine-grained Computation Steps

We then explore possible methods to alleviate the OOD generalization problem. One observation is that the complexity of addition with long digits is larger than that of the 1-digit addition. Thus, the model should be given more computation time on the task when the numbers are large. The fine-tuned T5 and prompted GPT3 mentioned above, however, is required to generate the answer with a fixed amount of computation, so one possible direction to mitigate this limitation is to allow the model to operate step-by-step instead of generating the answer in one forward pass. For example, in k-digit addition, the model is allowed to break it down into k simple 1-digit addition and the model is allowed to generate k intermediate addition results to get the final answer.

Generating fine-grained computation steps can potentially alleviate the generalization problem, but may not contribute to the locating capability of the Transformer-based LMs. To mitigate the locating problem, we add positional markers to scratchpad [74] (Figure 4.6):

We also experiment a more comprehensive scheme where each number in the demonstration is associated with an explicit positional marker or reference marker. A reference marker refers the positional marker where the following number is copied from as shown

```
question: 1 1 + 2 5
solution:
convert 1 1 into ☞ 1, ☛ 1.
convert 2 5 into ☞ 2, ☛ 5.
☛ 1 5, carry 0, so 1 + 5 + 0 = 6. carry 0, step result 6.
combine 6 and result, get result 6.
☞ 1 2, carry 0, so 1 + 2 + 0 = 3. carry 0, step result 3.
combine 3 and result 6, get result 3 6.
carry 0, combine 0 and result 3 6, final result 3 6.
```

Figure 4.6: The prompt for GPT3 on the addition task. We use 👉 and 👉 to denote optional different markers as described in Section 4.4.1 if they are applied.

in Figure 4.7.

```
question: question: S[B] 1 S[A] 1 + T[B] 2 T[A] 5
solution:
S[A] 1 + T[A] 5 + Z[A] 0 = R[A] 6, Z[B] 0
S[B] 1 + T[B] 2 + Z[B] 0 = R[B] 3, Z[C] 0
result: Z[C] 0 R[B] 3 R[A] 6
```

Figure 4.7: The demonstration of comprehensive scheme for addition problem, where position marker is marked red and reference marker is marked in green.

Through our experiments, we found that although these markers can help achieve higher accuracy for in domain data. It does not help much for OOD data. It clearly indicates the limitation of Transformers and pre-trained language models in induction. In the following discussion, we will shed some insights on how to eliminate such limitation.

### 4.4.3 LM with callable Programs

If both implicit positional markers and explicit positional markers do not generalize well in a symbolic reasoning task, then an alternative is to combine LMs with callable programs to replace the basic symbolic operations when possible, since callable programs do not have the generalization problem. For example, when combined with the fine-grained computation steps in the addition task, the convert, add, or combine operations can be considered callable programs. When the LM generates the text sequence `add(1,5)`, the

callable function `add` will be invoked and return the result in text: `carry C: 0, result 6`.

Following the example in Section 4.4.2, with callable functions, the prompt format is as follows:

```
question: 1 1 + 2 5
solution:
call convert (1 1, 2 5), return ☞ (1 2), ☛ (1 5).
☛ (1 5), call add (1, 5), return carry C: 0, result 6.
call combine (6, ), return 6.
☞ (1 2), call add (C: 0, 1, 2), return carry C: 0, result 3.
call combine (3, 6), return 3 6.
call combine (C: 0, 3 6), return 3 6, final result 3 6.
```

Figure 4.8: The prompt for GPT3 on the addition task with callable programs. ☞ and ☛ are positional markers. Different callable programs (convert, add and combine) are marked in different colors, and the results they returned are underlined with the corresponding color.

Given a testing example, the prompted GPT3 first generates the solution step by step. During the process, the results of the function calls will be appended to the generated result to be used in the following steps.

Callable programs can be viewed as decomposing a complex task to smaller, simpler jobs. The remaing issue is to learn chaining these smaller jobs together to complete the task.

Callable programs can guarantee the correctness of output given correct input for a given job. However, even augmented with callable programs, LMs may still suffer from the locating problem since the callable programs rely on LMs to decide which token to copy (Figure 4.9). Unfortunately, LMs cannot guarantee the correctness of this copy action.

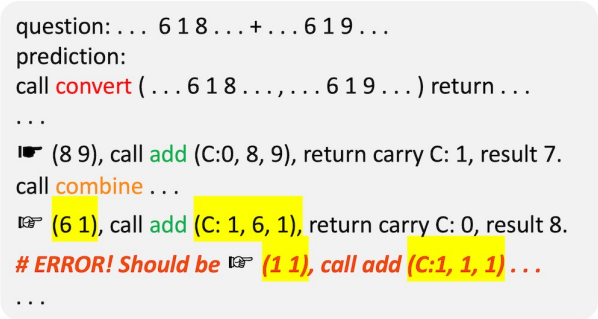



Figure 4.9: An error example of GPT3 with callable functions. The error is highlighted.

### 4.4.4 LM with Tutor

Scratchpad [74] ignores the visual process when an elementary school tutor visually illustrates how to perform addition step by step: Pinpointing where each digit in the output sequence comes from, adding single digits together and iterating. It turns out that these details and abstractions are important in order to simplify the learning process and help kids learn how to do addition in a few shots.

A tutor shows every single step visually and sometimes calls an already learned submodule to complete a task. In this way, the hypothesis space between two consecutive steps can be dramatically simplified; hence the chance of learning a correct model can be improved.

Taking copy as an example. Instead of providing a training example: `copy: 1 1 1 2 2 2 result: 1 1 1 2 2 2`, we need to demonstrate where the first 1, the second 1, the third 1 in the output sequence comes from, which exactly imitates the finest action a human could do to perform such an operation. Suppose there is a cursor placed at the beginning of the input sequence, a "rmov" operation moves the cursor one token to the right. A "cpy" operation copies a single digit to the output sequence. An "end" operation checks if the marker reaches the end of the sequence. "T" and "F" represent true and false respectively. We assume all these actions are unitary and have been learned.

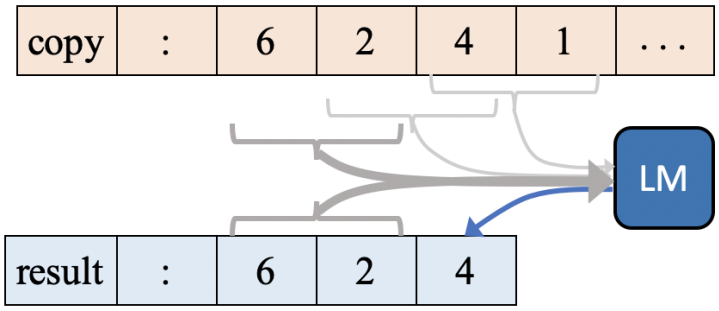



Figure 4.10: An illustration of doing copy with pattern matching.

Then a possible action sequence to complete the copy operation is as follows:

`rmov, end=F, cpy, rmov, end=F, cpy, . . . , rmov, end=T.`

This fine-grained action sequence accurately describes the whole copy operation. Certainly, there are other ways to perform copying. For example, instead of using a cursor, one can use a pattern match to perform the copy operation (Figure 4.10). We suspect that the copy operation learned from Transformer is following this pattern-matching approach, which is error-prone when the pattern has repeating symbols and when the long pattern is out-of-distribution. Positional markers do not help either as it seems unable to handle the OOD generalization problem.

If we take the action sequence "rmov, end=F, . . . " to train a Transformer for copying, the hypothesis space is simplified, thus making it possible to find the simplest model that can simulate the whole action sequence. This is related to imitation learning [88, 89]. Although there is no guarantee that Transformer can definitely find the correct model, the chance is much higher. One can also relate the setting with a multiple tape Turing machine where state transition is conducted among the positions of tape heads and read/write operations. The Transformer is trained to learn such state transition, thus completing the programming of a Turing machine.

As for the addition operation, a similar action sequence can be obtained to simulate how humans tutor kids do addition at an early age (Figure 4.11). Let "lmov" denote

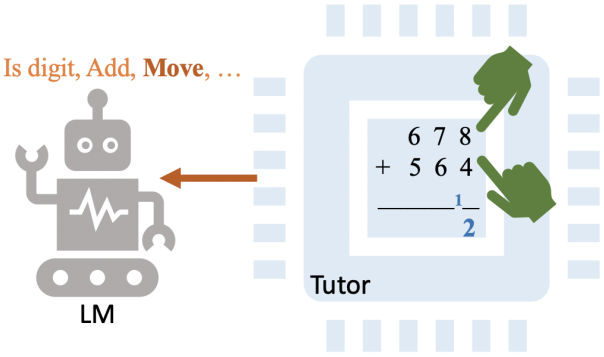



Figure 4.11: An illustration of the LM with Tutor method. With the tutor (right), the LM (left), or just a transformer generates an action sequence that simulates how humans do arithmetic addition.

moving the cursor one token to the left. The "add" operation adds three single digits together, one from each of the two operands and the third one from the carry digit, appends the result to the output, and updates the carry digit. Assume "add" is a callable program as kids have learned how to do single digits addition. Suppose the cursor starts from the end of the operands. The entire action sequence looks like the following.

`lmov, end=F, add, lmov, end=F, add, . . . , lmov, end=T.`

The main difference between the tutor and the Scratchpad method [74] is the abstract callable function and detailed action sequence. The action sequence includes all the state transitions needed to complete the task. It perfectly overcomes the OOD issue and does not require many training examples in order to achieve 100% accuracy.

While there is a great effort to enlarge Transformer-based LMs such as PALM [82] and Minerva [90], to improve the performance in symbolic and logical reasoning, our result reveals that it might be necessary to demonstrate the action sequence with reasonable abstraction to the Transformer to leverage its full strength.

The action sequence "rmov, end=F, ..." can also be viewed as an algorithm execution instance. It is well known that if one step is missing in an algorithm, it will most likely not produce the expected output. In order to fill the gap in case one step is missed during

demonstration, we have to rely on training examples of that step and learn it, which will likely incur errors. As the number of missing steps increases, errors will accumulate and eventually make learning more difficult and hard to generalize. To make learning in symbolic reasoning easier, it will be important to have a detailed action sequence as we have demonstrated in Figure 4.11.

## 4.5 Experiments

In this section, we conduct experiments on three different problems including copying, addition, and another basic symbolic manipulation operation, reverse. We illustrate the limitation of LMs in symbolic and arithmetic induction and the improvement that could be achieved by the introduced mitigation methods.

### 4.5.1 Experiment configuration

For fine-tuning the T5-base and DeBERTa model, we use the learning rate 5e-5, batch size 16, training epochs 200. The maximum generation length is set to 512. The checkpoints are evaluated every 1000 optimization steps. The random seed is fixed to 42. We use the implementation for HuggingFace [91]. For GPT3, we set temperature=0, top_p=1, frequency_penalty=0, and presence_penalty=0. All the experiments are conducted on NVIDIA RTX A6000 GPUs.

### 4.5.2 Copy Operation

Copying is the most basic operation used in symbolic manipulation and arithmetic reasoning. We experiment with the following methods:

**GPT3**: We use the prompt as shown in Section 4.3.

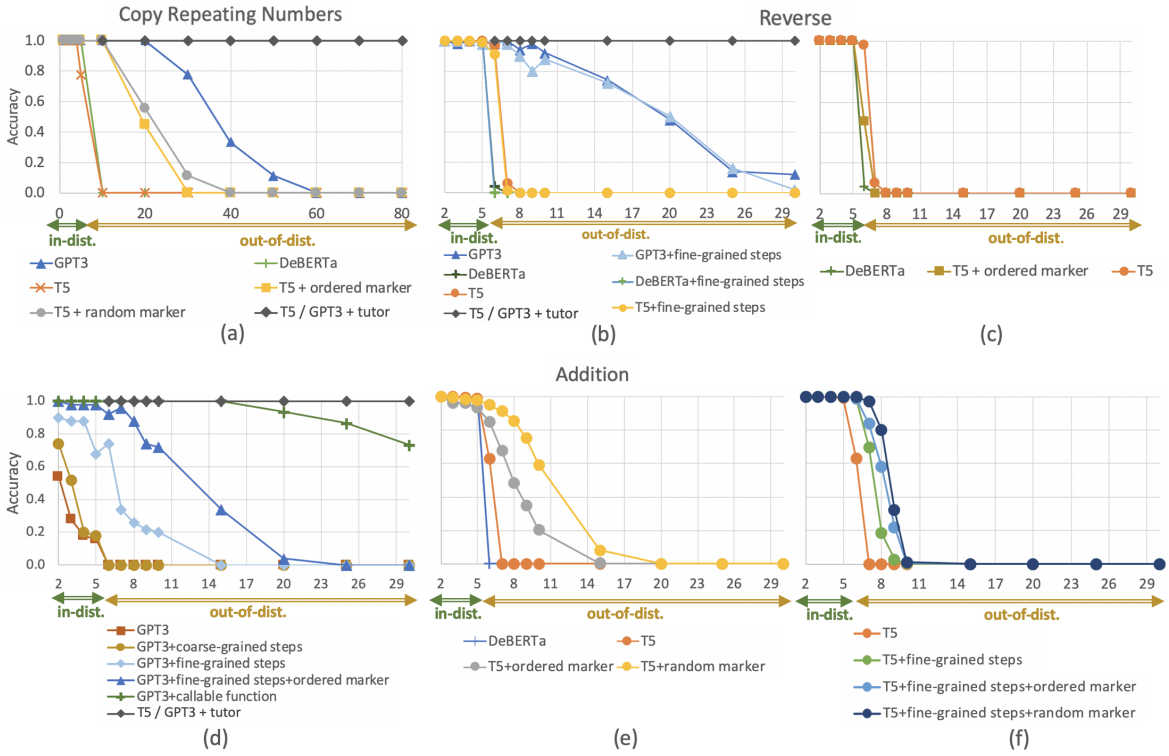



Figure 4.12: Experimental results. (a): results of copying repeating numbers. (b)(c): results of reversing the list. (d)(e)(f): results on arithmetic addition. The x-axis is the number of digits or number of items.

**DeBERTa / T5**: The training example is as follows: `copy: 1 2 3 4 result: 1 2 3 4`

**T5 + ordered marker**: The training data is augmented with explicit positional markers. `copy: A 1 B 2 C 3 result: A 1 B 2 C 3`

**T5 + random marker**: Same as above, but the augmented positional markers are in random order. `copy: E 1 A 2 F 3 result: E 1 A 2 F 3`

**T5 / GPT3 + tutor**: The training and testing examples are as described in Section 4.4.4.

We experiment with the T5-base (220M) model, DeBERTa-base (140M) model, and GPT3 text-davinci-002. The models are initiated with the pretrained parameters and further fine-tuned on the training data. For GPT3 or T5 with tutor, the training data consists of 15 examples of up to 5 digits. For all the other T5 models and DeBERTa, the

training data consists of 2,000 random numbers of up to 5 digits. We evaluate all the models on copying repeating numbers of up to 80 digits. The results are illustrated in Figure 4.12(a).

As shown in Figure 4.12(a), GPT3 achieves 100% accuracy on the in-distribution testing data (1-5 digits) but the fine-tuned T5 achieves 78% accuracy on the 5-digit repeating numbers although they are in-distribution. Augmented with random or ordered positional markers, the T5 models achieve 100% in-distribution accuracy, and so does using implicit positional markers (DeBERTa). This suggests that both implicit positional markers and explicit positional markers may help with the locating capability of LMs. However, using explicit positional markers, either ordered or random, the model exhibits significantly better generalization to OOD testing data whereas DeBERTa fails on OOD data. GPT3 exhibits better OOD generalization than T5 with positional markers but it does not generalize well beyond 30 digits. Both *T5 + tutor* and *GPT3 + tutor* keeps 100% accuracy when the number of digits increases.

```
question: 1 1 + 2 5
result: 3 6
question: 5 0 2 + 7 0 3
result: 1 2 0 5
question: 1 9 2 7 + 4 2 1 8
result: 6 1 4 5
question: 3 1 3 9 8 + 4 7 2 7 1
result: 7 8 6 6
```

Figure 4.13: The prompt for GPT3 on the addition task without intermediate steps.

### 4.5.3 Addition

For arithmetic addition, we experiment with the following methods:

**GPT3**: The exemplars are shown in Figure 4.13.

**GPT3 + coarse-grained steps**: The exemplar is similar to that in Figure 4.6, but the instructions for the result combination and the computation of the carry digit and step result are omitted.

**GPT3 + fine-grained steps (+ ordered marker)**: The exemplar we use is as shown in Figure 4.6.

**GPT3 + callable programs**: The exemplar is shown in Figure 4.8.

**DeBERTa / T5**: The training data follows the format of the exemplar for GPT3.

**DeBERTa / T5 + fine-grained steps**: The training data used in this setting follow the format as the exemplar in *GPT3 + fine-grained steps*.

**T5 + ordered / random marker**: The training example is augmented with ordered or random markers. For example, `question: G 1 C 1 + G 2 C 5 result: G 3 C 6`.

**T5 + fine-grained steps + ordered / random marker**: The training data in this setting follow a similar format as the exemplar in *GPT3 + fine-grained steps + ordered marker*, but the positional markers can be in random order.

**T5 / GPT3 + tutor**: The training and testing examples are as described in Section 4.4.4.

The model settings are the same as in the above copy experiments. For LMs with tutor, the training data or prompt consists of 15 examples of up to 5 digits. In other settings, the training data consists of 1,000 examples of 1-5 digit addition and for GPT3, the prompt includes 4 examples. We evaluate all the models on the addition of up to 30 digits. The results are shown in Figure 4.12(d)(e)(f).

As shown in Figure 4.12(d), both coarse-grained and fine-grained computation steps contribute to the in-distribution performance of GPT3, and using finer-grained steps achieves larger performance gains on both in-distribution data and OOD data. The performance is further boosted with explicit positional markers. Experiments on T5 (Figure 4.12(e)(f)) also show the effectiveness of using explicit positional markers, with or with-

out fine-grained computation steps, indicating that the explicit positional markers might make it easier for LMs to learn the induction in the arithmetic reasoning tasks. Similar to the results on the copying task, both DeBERTa and *DeBERTa + fine-grained steps* achieve near 100% in-distribution accuracy but 0% OOD accuracy, suggesting that the relative position embedding of DeBERTa might have limited OOD generalization ability. On T5, incorporating fine-grained computation steps does not improve the OOD performance as significantly as on GPT3 (Figure 4.12(f)). The reason might be that fine-tuning T5 tends to overfit more easily than prompting GPT3. Unsurprisingly, *GPT3 + callable programs* achieves much better OOD generalization. However, its OOD performance still degrades as the number of digits increases. Same as in the copy experiments, *LMs + tutor* keeps 100% accuracy on all the experimented numbers of digits.

### 4.5.4 Reverse List

Besides copying and addition, we also experiment with reversing. Reversing is similar to copying. Both require replicating the items in the input, but reversing might be more challenging than copying in the terms of locating. In copying, the distance between each source digit and the replicated digit is the same for each digit in the number. However, when reversing, the distance between the source item and the replicated item keeps increasing during the generation. For this problem, we experiment with the following methods:

**GPT3**: The prompt without any intermediate steps is used, as shown in Figure 4.14.

**DeBERTa / T5**: `reverse the list: bike, apple, book result: bike, cat, pen`

**GPT3 / DeBERTa / T5 + fine-grained steps**: The training example for T5 and the exemplar for GPT3 are shown in Figure 4.15.

**T5 + ordered marker**: The list items are augmented with the ordered positional

```
reverse the list: bike, cat, pen
result: pen, cat, bike
reverse the list: chair, bike, apple, book
result: book, apple, bike, chair
reverse the list: book, phone, fish, orange, fish
result: fish, orange, fish, phone, book
```

Figure 4.14: The prompt for GPT3 on the reverse task without intermediate steps.

```
reverse the list: bike, cat, pen
solution:
A is bike. B is cat. C is pen.
Now to reverse, change the order to:
C is pen. B is cat. A is bike.
Result: pen, cat, bike
```

Figure 4.15: The prompt for GPT3 on the reverse task with fine-grained steps.

markers in the input. `reverse the list: A bike, B cat, C pen result: pen, cat, bike`.

**T5 / GPT3 + tutor**: The training and testing examples are very similar to that for the copy task. The only difference is the direction of the move operation. "rmov" in the copy task is replaced by "lmov" here.

The model settings are the same as in the above experiments and the training data consists of examples of 1-5 items, which are randomly sampled from a predefined list of single-token nouns. For LMs with tutor, the training data or prompt consists of 15 examples of up to 5 items. In other settings, the training data consists of 1,000 examples for T5, and for GPT3, each prompt includes 4 examples. We evaluate all the models on reversing the list of up to 30 items. The results are shown in Figure 4.12(b)(c).

Although GPT3 can generalize to 80 digits on copying random numbers (Figure 4.2), it does not generalize well beyond 20 items on reversing, which suggests that reversing might require stronger locating capability than copying. This problem also occurs on

DeBERTa and T5. When tested on the OOD data, the models tends to generate only a sublist of the input. Using fine-grained steps (Figure 4.12(b)) or positional markers, whether implicit or explicit (Figure 4.12(c)), does not significantly improve the generalization of the experimented models. The reason might be the increasing distance between the source item and the replicated item as stated above. Again, *LMs + tutor* maintains 100% accuracy throughout the experiments.

### 4.5.5 Reference marker

As shown in Figure 4.7, we apply two different markers in the demonstration. The positional marker is used to define the value stored in the marker, while reference marker is used to explicitly copy the value from the positional marker with the same name. Each number in this demonstration is uniquely marked with positional or reference marker. For the positional marker, the model needs to generate both the marker and its value. For the reference marker, the model only needs to generate the marker and the value will be explicitly copied from its corresponding positional marker.

Similar to previous experiments on the addition problem, we train the model on 1-5 digits and test its performance on both in-domain (1-5 digits) and out-of-domain (6-10 digits) settings. The experimental results show that the model is able to achieve 100% accuracy on in-domain data, but get 0% accuracy on out-of-domain data. We also tried to extend the in-domain to 10 digits and get the same results that the model can solve in-domain problems, but fail to generalize to out-of-domain.

We show one error case of this model in Figure 4.16, where the error step is highlighted in yellow. On this 6-digit addition problem, the model skipped the last digit and directly jump to the result, which causes the error. The problem is the model doesn't learn to how to generalize from 1-5 digits to 6 digits. Instead, it is overfitting to the training

```
question: S[F] 5 S[E] 2 S[D] 8 S[C] 1 S[B] 7 S[A] 1 +
          T[F] 6 T[E] 5 T[D] 0 T[C] 2 T[B] 4 T[A] 5
solution:
S[A] 1 + T[A] 5 + Z[A] 0 = R[A] 6, Z[B] 0.
S[B] 7 + T[B] 4 + Z[B] 0 = R[B] 1, Z[C] 1.
S[C] 1 + T[C] 2 + Z[C] 1 = R[C] 4, Z[D] 0.
S[D] 8 + T[D] 0 + Z[D] 0 = R[D] 8, Z[E] 0.
S[E] 2 + T[E] 5 + Z[E] 0 = R[E] 7, Z[F] 0.
result: Z[F] 0 R[E] 7 R[D] 8 R[C] 4 R[B] 1 R[A] 6
```

Figure 4.16: Error case for T5 model with positional and reference marker on addition problem.

data, which makes it directly output the results after adding 5 digits. How to reduce the hypothesis space and force the model to learn to generalize to out-of-domain data would be one future research direction to solve this problem.

### 4.5.6 Discussion

From the experimental results, we observe that fine-grained computation steps may improve the LM's induction ability on the arithmetic reasoning tasks and the granularity of the steps has an impact on the performance improvement. Finer-grained computation steps may contribute to larger performance improvement.

Positional markers, whether implicit or explicit, improves LMs' in-distribution performance on all the symbolic manipulation tasks in our experiments. However, We find that augmented with the relative position embeddings, DeBERTa tends to face more severe over-fitting than T5 during fine-tuning. In the reversing experiment, using the T5 model without pretrained parameters, the fine-tuned model can not achieve a good in-distribution performance after 200k optimization steps. However, the DeBERTa model without pretrained parameters achieves 100% in-distribution accuracy within only 2k optimization steps while the OOD accuracy drops, indicating that it has overfitted within 2k optimization steps. In other words, the relative position embeddings in DeBERTa

significantly improve the model's capacity of positions, which improves in-distribution performance on simple symbolic manipulation tasks, but may not generalize well on OOD data. Compared with the implicit positional markers (relative position embeddings in DeBERTa), explicit positional markers might have better OOD generalization ability. However, incorporating symbolic manipulation tasks in the LM pretraining stage might alleviate this problem, so incorporating implicit positional markers can still be a possible direction of improving the LM's performance on reasoning tasks requiring locating ability.

Using LM with callable programs exhibits strong OOD performance on addition, suggesting that the LMs' ability to perform simple symbolic operations, such as copying, splitting, and combining, can be critical for improving their performance on reasoning tasks. How to further improve the LMs' performance on more complex reasoning tasks in this direction is left for future work.

## 4.6 Conclusion

In this work, we explore the limitations of pretrained LMs on arithmetic reasoning and symbolic manipulation. We experiment with three simple symbolic manipulation tasks and show that improving the locating and induction capability of LMs can be important for further improving their performance on induction tasks. Our method that combines abstraction and finest-grained step-by-step tutoring demonstrates its potential to generalize correctly, shedding light on possible directions orthogonal to scaling up LMs for future work in this area.

# Part II

# Optimizing Resource Utilization

# Chapter 5

# Effective Model Updates through Lifelong Learning

## 5.1 Introduction

The task of relation detection/extraction aims to recognize entity pairs' relationship from given contexts. As an essential component for structured information extraction, it has been widely used in downstream tasks such as automatic knowledge-based completion [92] and question answering [93, 94].

Existing relation detection methods always assume a closed set of relations and perform once-and-for-all training on a fixed dataset. While making the evaluation straightforward, this setting clearly limits the usage of these methods in realistic applications, where new relations keep emerging over time. To build an evolving system which automatically keeps up with the dynamic data, we consider a more practical lifelong learning setting (also called *continual learning*) [95, 96, 97], where a learning agent learns from a sequence of tasks, where each of them includes a different set of relations. In such scenarios, it is often infeasible to combine the new data with all previous data and re-train

the model using the combined dataset, especially when the training set for each task is huge.

To enable efficient learning in such scenarios, recent lifelong learning research [98, 99] propose to learn the tasks incrementally, while at the same time preventing catastrophic forgetting [100, 101, 102, 103], i.e., the model abruptly forgets knowledge learned on previous tasks when learning on the new task. Current lifelong learning approaches address such challenge by either preserving the training loss on previously learned tasks (GEM) [99], or selectively dimming the updates on important model parameters (EWC) [98]. These methods usually involve adding additional constraints on the model's parameters or the updates of parameters by utilizing stored samples. Despite the effectiveness of these methods on simple image classification tasks, there is little research validating the practical usage of these methods in realistic NLP tasks. In fact, when applying these methods to our relation extraction task, we observe that they underperform a simple baseline that updates the model parameters (i.e., learning by SGD) with a mix of stored samples from previous tasks and new samples from the incoming task. We further test this simple baseline on commonly used continual learning benchmarks and get similar observations.

In this work, we thoroughly investigate two existing continual learning algorithms on the proposed lifelong relation extraction task. We observe that recent lifelong learning methods only operate on the models' parameter space or gradient space, and do not explicitly constraint the feature or embedding space of neural models. As we train the model on the new task, the embedding space might be distorted a lot, and become infeasible for previous tasks. We argue that the embedding space should not be distorted much in order to let the model work consistently on previous tasks. To achieve this, we propose an alignment model that explicitly anchors the sentence embeddings derived by the neural model. Specifically, the alignment model treats the saved data from previous

tasks as anchor points and minimizes the distortion of the anchor points in the embedding space in the lifelong relation extraction. The aligned embedding space is then utilized for relation extraction. Experiment results show that our method outperforms the state-of-the-art significantly in accuracy while remaining efficient.

The main contributions of this work include:

• We introduce the lifelong relation detection problem and construct lifelong relation detection benchmarks from two datasets with large relation vocabularies: SimpleQuestions [7] and FewRel [8].

• We propose a simple memory replay approach and find that current popular methods such as EWC and GEM underperform this method.

• We propose an alignment model which aims to alleviate the catastrophic forgetting problem by slowing down the fast changes in the embedding space for lifelong learning.

## 5.2 Problem Definition

**Generic definition of lifelong learning problems** In lifelong learning, there is a sequence of $K$ tasks $\{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots, \mathcal{T}^{(K)}\}$. Each task $\mathcal{T}^{(k)}$ is a conventional supervised task, with its own label set $L^{(k)}$ and training/validation/testing data $(T^{(k)}_{\text{train}}, T^{(k)}_{\text{valid}}, T^{(k)}_{\text{test}})$, each of which is a set of labeled instances $\{(x^{(k)}, y^{(k)})\}$. Note that $x^{(k)}$ is the input data of the context and candidate relations, and $y^{(k)}$ is the ground-truth label. The goal of lifelong learning is to learn a classification model $f$. At each step $k$, $f$ observes the task $\mathcal{T}^{(k)}$, and optimizes the loss function on its training data with a loss function $\ell(f(x), y)$. At the same time, we require the model $f$ learned after step $k$ could still perform well on the previous $k-1$ tasks. That is, we evaluate the model by using the average accuracy of $k$ tasks at each step as $\frac{1}{k}\sum_{j=1}^{k} acc_{f,j}$.

To make $f$ perform well on the previous tasks, during the lifelong learning process,

we usually allow the learner to maintain and observe a memory $\mathcal{M}$ of samples from the previous tasks. Practically, with the growth of the number of tasks, it is difficult to store all the task data[1]. Therefore, in lifelong learning research, the learner is usually constrained on the memory size, denoted as a constant $B$. Thus at each step $k$, the learner is allowed to keep training samples from $\{\mathcal{T}^{(j)}|j = 1, \ldots, k-1\}$ with size less or equal to $B$.

**Lifelong relation detection** In this chapter we introduce a new problem, *lifelong relation detection*. Relation detection is an important task that aims to detect whether a relation exists between a pair of entities in a paragraph. In many real-world scenarios, relation detection naturally forms a lifelong learning problem because new relation types emerge as new knowledge is constantly being discovered in various domains. For example, in the Wikidata [104] knowledge graph, the numbers of new items and properties are constantly increasing[2]. So we need to keep collecting data and updating the model over time in order to handle newly added relations.

The problem of lifelong relation detection has the same definition as above with only one difference: during prediction time, we hope to know whether an input paragraph contains any relation observed before. Therefore at time $k$, given an input $x$ from task $j'<k$, instead of predicting an $y \in L^{(j')}$, we predict $y^{(k)} \in \bigcup_{j=1}^{k} L^{(j)}$. That says, the candidate label set is expanding as the learner observes more tasks, and the difficulty of each previous task is increasing over time as well.

[1]Even the data can be stored, it is unrealistic to make full usage of the stored data. For example, random sampling from all previous task data (e.g., for the methods in Section 5.4) will become statistically inefficient.

[2]`https://www.wikidata.org/wiki/Wikidata:News`

## 5.3 Evaluation Benchmarks for Lifelong Learning

### 5.3.1 Previous non-NLP Benchmarks

**Lifelong MNIST** MNIST is a dataset of handwriting ten digits [105], where the input for each sample is an image, and the label is the digit the image represents. Two variants of the MNIST dataset were proposed for lifelong learning evaluation. One is MNIST *Permutations* [98], where a task is created by rearranging pixels according to a fixed permutation. $K$ different permutations are used to generate $K$ tasks. Another variant is MNIST *Rotations* [99], where each task is created by rotating digits by a fixed angle. $K$ angles are chosen for creating $K$ tasks. In our experiments, we follow [99] to have $K = 20$ tasks for each benchmark.

**Lifelong CIFAR** CIFAR [106] is a dataset used for object recognition, where the input is an image, and the label is the object the image contains. Lifelong CIFAR100 [107] is a variant of CIFAR-100 (CIFAR with 100 classes) by dividing 100 classes into $K$ disjoint subsets. Each task contains samples from $\frac{100}{K}$ classes in one subset. Following [99], we have $K = 20$ tasks, where each of them has 5 labels.

### 5.3.2 The Proposed Lifelong Relation Detection Benchmarks

**Lifelong FewRel** FewRel [8] is a recently proposed dataset for few-shot relation detection. There are 80 relations in this dataset. We choose to create a lifelong benchmark based on FewRel because there are a sufficient number of relation labels. We extract the sentence-relation pairs from FewRel and build our lifelong FewRel benchmark as follows. Each sample contains a sentence with the ground-truth relation it refers, and a set of 10 randomly chosen false relations from all the whole relations set. The model is required to distinguish the right relation from the candidates. We apply K-Means over the averaged

word embeddings of the relation names and divide 80 relations into 10 disjoint clusters. This results in 10 tasks in this benchmark, and each task contains relations from one cluster. Candidate relations will be masked if they do not appear in the history tasks.

**Lifelong SimpleQuestions** SimpleQuestions is a KB-QA dataset containing single-relation questions [7]. [94] created a relation detection dataset from SimpleQuestions that contains samples of question-relation pairs. For each sample, a candidate set of relations is also provided. Similar to lifelong FewRel, we divide relations into 20 disjoint clusters by using K-Means. This results in 20 tasks, and each task contains relations from one cluster.

## 5.4 Simple Episodic Memory Replay Algorithm for Lifelong Learning

Catastrophic forgetting is one of the biggest obstacles in lifelong learning. The problem is particularly severe in neural network models, because the learned knowledge of previous tasks is stored as network weights, while a slight change of weights when learning on the new task could have an unexpected effect on the behavior of the models on the previous tasks [103].

Currently, the memory-based lifelong learning approaches, which maintain a working memory of training examples from previous tasks, are proved to be one of the best solutions to the catastrophic forgetting problem. In this section, we first propose a memory-based lifelong learning approach, namely Episodic Memory Replay (EMR), which uses the working memory by sampling stored samples to replay in each iteration of the new task learning. Surprisingly, such a straightforward approach with a clear motivation was never used in previous research. We first compare EMR with the state-of-the-art

memory-based algorithm Gradient Episodic Memory (GEM). We also show that the EMR outperforms GEM on many benchmarks, suggesting that it is likely to be among the top-performed lifelong learning algorithms, and it should never be ignored for comparison when developing new lifelong learning algorithms.

### 5.4.1 Episodic Memory Replay (EMR)

EMR is a modification over stochastic gradient descent algorithms. It replays randomly sampled data from memory while training on a new task, so the knowledge of previous tasks could be retained in the model. After training on each task $k$, EMR selects several training examples to store in the memory $\mathcal{M}$, denoted as $\mathcal{M} \bigcap T^{(k)}_{\text{train}}$.[3]

To handle the scalability, EMR stochastically replays the memory. Specifically, when training on task $k$ with each mini-batch $D^{(k)}_{\text{train}} \subset T^{(k)}_{\text{train}}$, EMR samples from the memory $\mathcal{M}$ to form a second mini-batch $D^{(k)}_{\text{replay}} \subset \mathcal{M}$. Then two gradient steps are taken on the two mini-batches of $D^{(k)}_{\text{train}}$ and $D^{(k)}_{\text{replay}}$. Note that EMR could work with any stochastic gradient optimization algorithm, such as SGD, Adagrad, AdaDelta, and Adam, to optimize the model $f$ with the mixed mini-batches.

We try two variations of $D^{(k)}_{\text{replay}}$ sampling: first, *task-level sampling*, which samples from one previous task $j$ each time, i.e., $D^{(k)}_{\text{replay}} \subset \mathcal{M} \bigcap T^{(j)}_{\text{train}}$. Second, *sample-level sampling*, which samples all over the memory, i.e., $D^{(k)}_{\text{replay}} \subset \mathcal{M}$.

The two approaches differ in the task instance sampling probability. The task-level approach assumes a uniform distribution over tasks, while the sample-level approach has a marginal distribution on tasks that is proportional to the number of their training data in $\mathcal{M}$.[4] When tasks are balanced like MNIST and CIFAR, or when the stored data in

[3]Previous research [108] propose to dynamically change the size of memory set for each task during training. The followup work and this chapter all use fixed sets, and we will investigate the usage of dynamic sets in future work.

[4]The two approaches hence favor different evaluation metrics – the former fits macro averaging better

the memory for different tasks are balanced, the two approaches become equivalent.

However, the sample-level strategy could sometimes make the code implementation more difficult: for some lifelong learning benchmarks such as MNIST Rotation, MNIST Permutation, and CIFAR-100 used in [99], the tasks could differ from each other in the input or output distribution, leading to different computation graphs for different training examples. From our preliminary study, the task-level approach could always give results as good as those of the sample-level approach on our lifelong relation detection benchmarks (see Table 5.1) , so in our experiments in Section 5.6 we always use the task-level approach.

### 5.4.2 Comparing EMR with State-of-the-art Memory-based Lifelong Algorithm

In this part, we will first thoroughly introduce a state-of-the-art memory-based lifelong learning algorithm called Gradient Episodic Memory (GEM) [99], and then compare EMR with it in both time complexity and experimental results on several benchmarks.

**Gradient Episodic Memory (GEM)** The key idea of GEM [99] is to constrain the new task learning with previous task data stored in memory. Specifically, it constrains the gradients during training with the following operation. When training on task $k$, for each mini-batch $D^{(k)}_{\text{train}} \subset T^{(k)}_{\text{train}}$, it first computes the gradient $g^{(k)}_{\text{train}}$ on $D^{(k)}_{\text{train}}$, and the average gradients on the stored data of each previous task $j$, denoted as $g^{(j)}_{\text{task}}$. More concretely, we define

$$g^{(j)}_{\text{task}} = \frac{\sum_{i'} \nabla \ell(f(x^{(j)}_{i'}), y^{(j)}_{i'})}{|\mathcal{M} \bigcap T^{(j)}_{\text{train}}|},$$

and the latter fits micro averaging better.

where $j<k$, $\ell(\cdot)$ is the loss function, and $(x_{i'}^{(j)}, y_{i'}^{(j)}) \in \mathcal{M} \bigcap T_{\text{train}}^{(j)}$, i.e. $(x_{i'}^{(j)}, y_{i'}^{(j)})$ is a training instance in $\mathcal{T}^{(j)}$ that was stored in memory $\mathcal{M}$. Then the model $f$ is updated along the gradient $\tilde{g}$ that solves the following problem:

$$\begin{aligned} &\min_{\tilde{g}} \quad ||\tilde{g} - g_{\text{train}}^{(k)}||^2 \\ &\text{s.t.} \quad \langle \tilde{g}, g_{\text{task}}^{(j)} \rangle \geq 0, \ j = 1, \ldots, k-1. \end{aligned}$$

$\tilde{g}$ is the closest gradient to the gradient on the current training mini-batch, $g_{\text{train}}^{(k)}$, without decreasing performance on previous tasks much since the angle between $\tilde{g}$ and $g_{\text{task}}^{(j)}$ is smaller than 90°.

**Time Complexity** One difference between EMR and GEM is that EMR deals with unconstrained optimization and does not require the gradient projection, i.e., solving $\tilde{g}$. But since the model $f$ is deep networks, empirically the time complexity is mainly dominated by the computation of forward and backward passes. We analyze the time complexity as below:

In task $k$, suppose the mini-batch size is $|D|$ and the memory replay size is $m$, our EMR takes $|D| + m$ forward/backward passes in each training batch. Note that $m$ is a fixed number and set to be equal to the number of instances stored for each previous task in our experiments. While for GEM, it needs to compute the gradient of all the data stored in the memory $\mathcal{M}$, thus $|D| + |\mathcal{M}|$ forward/backward passes are taken. Its complexity is largely dominated by the size $|\mathcal{M}|$ (upper bounded by the budget $B$). When the budget $B$ is large, with the number of previous tasks increases, $\mathcal{M}$ grows linearly, and GEM will become infeasible.

**Superior Empirical Results of EMR** The EMR algorithm is much simpler compared to the GEM. However, one interesting finding of this chapter is that the state-of-

| Task | EMR | | GEM |
|---|---|---|---|
| | sample | task | |
| MNIST Rotation | – | 0.828 | **0.860** |
| MNIST Permutation | – | 0.824 | **0.826** |
| CIFAR-100 | – | **0.675** | **0.675** |
| FewRel | 0.606 | **0.620** | 0.598 |
| SimpleQuestions | 0.804 | **0.808** | 0.796 |

Table 5.1: The average accuracy across all the tasks at last time step for EMR and GEM on both non-NLP and our lifelong relation detection benchmarks. For the experiments on MNIST and CIFAR, we follow the setting in [99] (see section 5.6.1 for details). For the experiments on FewRel and SimpleQuestions, we use the same setting in Section 5.6. We only implement task-level EMR for MNIST and CIFAR because of the relatively easy implementation.

the-art GEM is unnecessarily more complex and more inefficient, because EMR, a simple stochastic gradient method with memory replay, outperforms it on several benchmarks.

The results are shown in Table 5.1. The numbers are the average accuracy, i.e. $\frac{1}{k}\sum_{j=1}^{k} acc_{f,j}$, at last time step. For both algorithms, the training data is randomly sampled to store in the memory, following [99]. On lifelong relation detection, the EMR outperforms GEM on both of our created benchmarks. To further show its generalizability, we apply the EMR to previous lifelong MNIST and CIFAR benchmarks and compare to the results in [99] with all the hyperparameters set as the same. Still, EMR performs similarly to GEM except for the MNIST Rotation benchmark.[5]

From the above results, we learned the lesson that previous lifelong learning approaches actually fail to show improvement compared to doing memory replay in a stochastic manner. We hypothesise that GEM performs worse when there is positive transfer among tasks, making the gradient projection an inefficient way to use gradients computed from memory data. Therefore, in the next section, we start with the basic EMR and focus on more efficient usage of the historical data.

[5]Even on MNIST Rotation, it has achieved a competitive result, since the conventional training on shuffled data from all the tasks in this benchmark gives $\sim 0.83$ according to [99].

## 5.5 Embedding Aligned EMR (EA-EMR)

Based on our basic EMR, this section proposes our solution to lifelong relation detection. We improve the basic EMR with two motivations: (1) previous lifelong learning approaches work on the parameter space. However, the number of parameters in a deep network is usually huge. Also, deep networks are highly non-linear models, and the parameter dimensions have complex interactions, making the Euclidean space of parameters not a proper delegate of model behavior [103]. That is, a slight change in parameter space could affect the model prediction unexpectedly. The above two reasons make it hard to maintain deep network behaviors on previous tasks with constraints or Fisher information. Therefore, we propose to alleviate catastrophic forgetting in the hidden space (i.e., the sentence embedding space). (2) for each task, we want to select the most informative samples to store in the memory, instead of random sampling like in [99]. Therefore the budget of memory can be better utilized.

### 5.5.1 Embedding Alignment for Lifelong Learning

This section introduces our approach which performs lifelong learning in the embedding space, i.e., the Embedding Aligned EMR (EA-EMR).

In EA-EMR, for each task $k$, besides storing the original training data $(x^{(k)}, y^{(k)})$ in the memory $\mathcal{M}$, we also store the embeddings of $x^{(k)}$. In the future after a new task is trained, the model parameters are changed thus the embeddings for the same $(x^{(k)}, y^{(k)})$ would be different. Intuitively, a lifelong learning algorithm should allow such parameter changes but ensure the changes do not distort the previous embedding spaces too much.

Our EA-EMR alleviates the distortion of embedding space with the following idea: if the embedding spaces at different steps are not distorted much, there should exist a simple enough transformation $a$ (e.g., a linear transformation in our case) that could

transform the newly learned embeddings to the original embedding space, without much performance degeneration on the stored instances. So we propose to add a transformation $a$ on the top of the original embedding and learn the basic model $f$ and the transformation $a$ automatically. Specifically, at the $k$-th task, we start with the model $f^{(k-1)}$, and the transformation $a^{(k-1)}$, that trained on the previous $k-1$ tasks. We want to learn the basic model $f$ and the transformation $a$ such that the performance on the new task and stored instances are optimized without distorting the previous embedding spaces much.

$$\min_{f(\cdot),a(\cdot)} \sum_{(x,y)\in D^{(k)}_{\text{train}}} \ell(a(f(x)),y)+ \\ \sum_{(x,y)\in D^{(k)}_{\text{replay}}} \left(\ell(a(f(x)),y) + \|a(f(x)) - a^{(k-1)}(f^{(k-1)}(x))\|^2\right)$$

We propose to minimize the above objective through two steps. In the first step, we optimize the basic model $f$ by:

$$\min_{f(\cdot)} \sum_{(x,y)\in D^{(k)}_{\text{train}} \bigcup D^{(k)}_{\text{replay}}} \ell\left(a^{(k-1)}(f(x)),y\right)$$

This step mainly focuses on learning the new task without performance drop on the stored samples.

In second step, we optimize $a$ to keep the embedding space of the current task and restore the previous embedding space of all stored samples:

$$\min_{a(\cdot)} \sum_{(x,y)\in D^{(k)}_{\text{train}}} \|a(f(x)) - a^{(k-1)}(f(x))\|^2 \\ + \sum_{(x,y)\in D^{(k)}_{\text{replay}}} \|a(f(x)) - a^{(k-1)}(f^{(k-1)}(x))\|^2$$

**Embedding Alignment on Relation Detection Model** We introduce how to add embedding alignment to relation detection models. The basic model we use is a ranking model that is similar to HR-BiLSTM [94]. Two BiLSTMs [42] are used to encode the

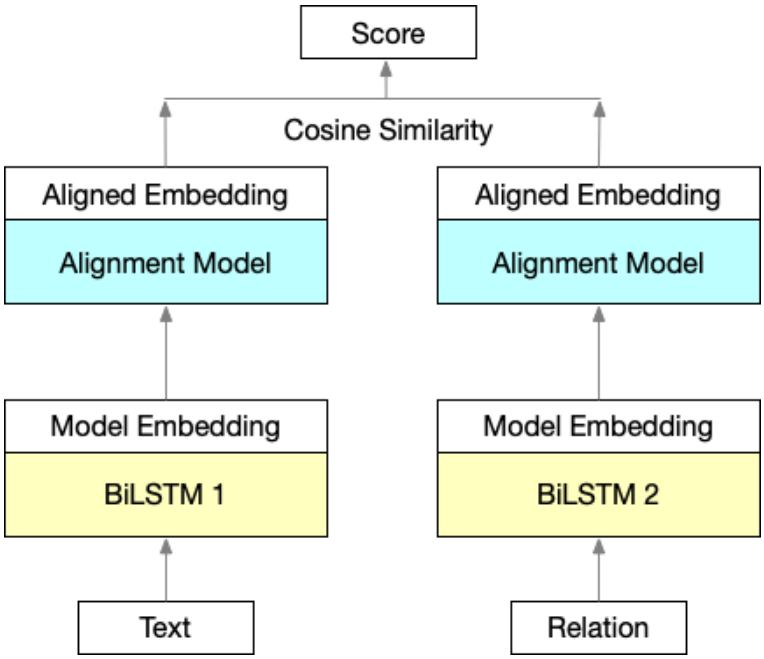



Figure 5.1: This figure shows how we add the alignment model (a linear model in our case) on the basic relation detection model, where two BiLSTMs are used to encode the text and relation, and cosine similarity between their embeddings are computed as the score.

sentence and relation respectively given their GloVe word embedding [52]. Cosine similarity between the sentence and relation embedding is computed as the score. Relation with maximum score is predicted by the model for the sentence. Ranking loss is used to train the model[6]. This base model is our model $f$, which is trained on a new task $k$ at each step and results in an updated model $f^{(k)}$. Our proposed approach (Figure 5.1) inserts an alignment model $a$ to explicitly align to embedding space for stored instances and maintain the embedding space of the current task. Note that the label $y$ (the relation here) also has embedding, so it needs to pass through the alignment model $a$ as well.

[6]Though the basic model is simple, it achieves reasonable results on the two datasets when training with all the data, i.e., 0.837 on FewRel and 0.927 on SimpleQuestions.

### 5.5.2 Selective Storing Samples in Memory

When the budget of memory is relatively smaller, how to select previous samples will greatly affect the performance. Ideally, in order to make the memory best represents a previous task, we hope to choose diverse samples that best approximate the distribution of task data. However, distribution approximation itself is a hard problem and will be inefficient due to its combinatorial optimization nature. Therefore, many recent works such as GEM ignore this step and randomly select samples from each task to store in the memory.

In [108], they proposed to select exemplars that best approximate the mean of the distribution. This simplest distribution approximation does not give an improvement in our experiments because of the huge information loss. Therefore, we propose a better approach of sample selection by clustering over the embedding space from the model, and choose one representative from each cluster to store in the memory. More specifically, The embedding after alignment model is used to represent the input because the model makes prediction based on that. Then we apply K-Means (the number of clusters equals the budget given to the specific task) to cluster all the samples of the task. For each cluster, we select the sample closest to the centroid to store in the memory.

We leave more advanced approaches of representative sample selection and their empirical comparison to future work.

## 5.6 Experiments

### 5.6.1 Experiment setting

**On MNIST and CIFAR** Following the setting in [99], the size of memory for each task is set to be 256. The learning rate is set to be 0.1. The epoch for training the model

on each task is set to be 1. Plain SGD and minibatch of 10 samples are used. For the MNIST dataset, each task has 1000 samples of 10 classes. For the CIFAR dataset, each task has 2500 samples of 5 classes.

**On relation extraction benchmarks** We conduct experiments on our lifelong benchmarks: lifelong SimpleQuestions [7] and lifelong FewRel [8] to compare our proposed methods EA-EMR, EA-EMR without Selection (EA-EMR_NoSel), EA-EMR without Alignment (EA-EMR_noAlign), and EMR with the following baselines.

• **Origin**, which simply trains on new tasks based on the previous model.

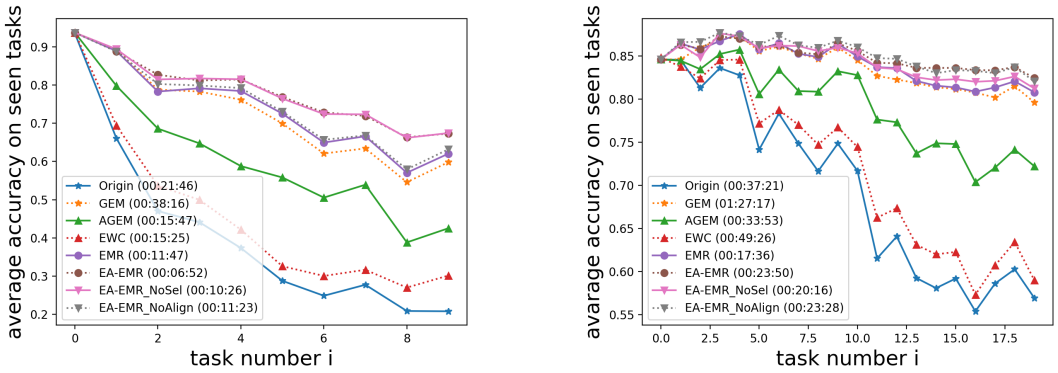
• **EWC** [98], which slows down updates on important parameters by adding $L_2$ regularization of parameter changes to the loss.

• **GEM** [99], which projects the gradient to benefit all the tasks so far by keeping a constraint for each previous task.

• **AGEM** [109], which only uses one constraint that the projected gradient should decrease the average loss on previous tasks.

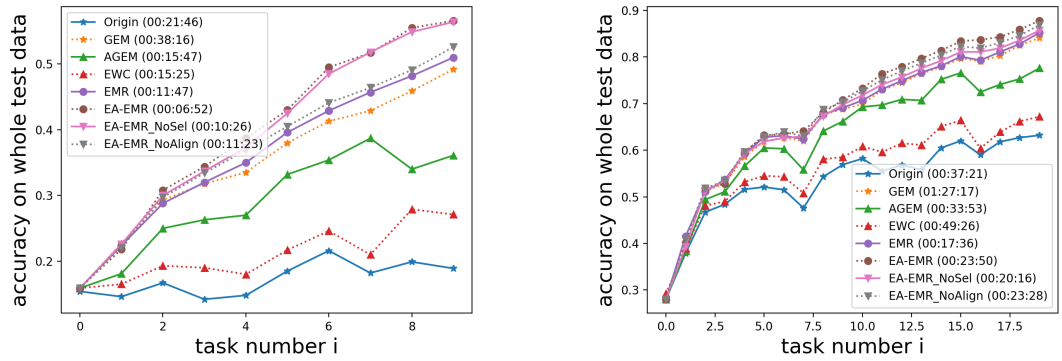
On both FewRel and SimpleQuestions, the epoch to train on each task is set to be 3. Learning rate for the basic model is set to be 0.001. The hidden size of LSTM is set to be 200. The batch size is set to be 50. For each sample in the memory, 10 candidate relations is randomly chosen from all observed relations to alleviate the problem that new relations are emerging incessantly.

Parameters for our model and baselines are set as follows. For EA-EMR and EA-EMR_NoSel, when training the alignment model, the learning rate is set to be 0.0001, and the training epoch is set to be 20 and 10 for FewRel and SimpleQuestions respectively. For AGEM, 100 samples are randomly chosen from all the previous tasks to form a constraint. For EWC, we set the balancing parameter $\alpha = 100$. For GEM and EMR related methods, memory size of each task is set to be 50.




(a) FewRel (b) SimpleQuestions

Figure 5.2: This figure shows the average accuracy of all the observed tasks on the benchmarks of lifelong FewRel and lifelong SimpleQuestions during the lifelong learning process. The average performance of 5 runs is reported, and the **average running time** is shown in the brackets.




(a) FewRel (b) SimpleQuestions

Figure 5.3: This figure shows the accuracy on the whole testing data on the benchmark of lifelong FewRel and lifelong SimpleQuestions during the lifelong learning process. The average performance of 5 runs is reported, and the **average running time** is shown in the brackets.

### 5.6.2 Lifelong Relation Detection Results

**Evaluation Metrics** We use two metrics to evaluate the performance of the model:

• Average performance on all seen tasks after time step $k$, which highlights the catastrophic problem:

$$\text{ACC}_{\text{avg}} = \frac{1}{k}\sum_{i=1}^{k} acc_{f,i}$$

| Method | FewRel | | SimpleQuestions | |
|---|---|---|---|---|
| | Whole | Avg | Whole | Avg |
| Origin | 0.189 | 0.208 | 0.632 | 0.569 |
| *Baselines* | | | | |
| GEM | 0.492 | 0.598 | 0.841 | 0.796 |
| AGEM | 0.361 | 0.425 | 0.776 | 0.722 |
| EWC | 0.271 | 0.302 | 0.672 | 0.590 |
| *Ours* | | | | |
| Full EA-EMR | **0.566** | 0.673 | **0.878** | **0.824** |
| w/o Selection | 0.564 | **0.674** | 0.857 | 0.812 |
| w/o Alignment | 0.526 | 0.632 | 0.869 | 0.820 |
| w/o Alignment but keep the architecture | 0.545 | 0.655 | 0.871 | 0.813 |
| EMR Only | 0.510 | 0.620 | 0.852 | 0.808 |

Table 5.2: This table shows the accuracy on the whole testing data ("Whole" column), and average accuracy on all observed tasks ("Avg" column) after the last time step. The average performance of 5 runs are listed here and the best result on each dataset is marked in bold.

• Accuracy on the whole testing data of all tasks:

$$\mathrm{ACC}_{\mathrm{whole}} = acc_{f, D_{\mathrm{test}}}$$

**Results on FewRel and SimpleQuestions** We run each experiment 5 times independently by shuffling sequence of tasks, and the average performance is reported. The average accuracy over all observed tasks during the whole lifelong learning process is presented in Figure 5.2, and the accuracy on the whole testing data during the process is shown in Figure 5.3. We also list the result at last step in Table 5.2. From the results, we can see that EWC and GEM are better than the Origin baseline on both two datasets, which indicates that they are able to reduce the catastrophic forgetting problem. However, our EA-EMR perform significantly better than these previous state-of-the-arts. The proposed EMR method itself achieves better results than all baselines on both datasets. The ablation study shows that both the *selection* and the *alignment* modules help on both tasks.

**The Effect of Embedding Alignment** To investigate the effect of our embedding alignment approach, we conduct two ablation studies as below: First, we remove both the alignment loss in equation 5.5.1, as well as the alignment module $a$, which results in significant drop on most of the cases (the line "w/o Alignment" in Table 5.2). Second, to make sure that our good results do not come from introducing a deeper model with the module $a$, we propose to only remove the embedding alignment loss, but keep everything else unchanged. That means, we still keep the module $a$ and the training steps, with the only change on replacing the loss in step 2 with the one in step 1 (the line "w/o Alignment but keep the architecture" in Table 5.2). We can see that this decreases the performance a lot. The above results indicate that by explicitly doing embedding alignment, the performance of the model can be improved by alleviating the distortion of previous embedding space.

**Comparison of Different Sample Selection Strategies** Here we compare different selection methods on lifelong FewRel and SimpleQuestions. EMR Only randomly choose samples. [108] propose to choose samples that can best approximate the mean of the distribution. We compare their sampling strategy (denoted as iCaRL) with our proposed method (K-Means) which encourages to choose diverse samples by choosing the central sample of the cluster in the embedding space. From the results in Table 5.3, we can see that our method outperforms iCaRL and the random baseline. While iCaRL is not significantly different from the random baseline.

## 5.7 Related Work

**Lifelong Learning without Catastrophic Forgetting** Recent lifelong learning research mainly focuses on overcoming the *catastrophic forgetting* phenomenon [103, 100,

| Method | FewRel | | SimpleQuestions | |
|---|---|---|---|---|
| | Whole | Avg | Whole | Avg |
| EMR Only | 0.510 | 0.620 | 0.852 | 0.808 |
| + K-Means | **0.526** | **0.632** | **0.869** | **0.820** |
| + iCaRL | 0.501 | 0.615 | 0.854 | 0.806 |

Table 5.3: Comparison of different methods to select data for EMR. The accuracy on the whole testing data ("Whole" column), and average accuracy on all observed tasks ("Avg" column) is reported. We run each method 5 times, and give their average results.

102, 101], i.e., knowledge of previous tasks is abruptly forgotten when learning on a new task.

Existing research mainly follow two directions: the first one is *memory-based approach* [99, 109], which saves some previous samples and optimizes a new task with a forgetting cost defined on the saved samples. These methods have shown strength in alleviating catastrophic forgetting, but the computational cost grows rapidly with the number of previous tasks. The second direction is to *consolidate parameters that are important to previous tasks* [98, 110, 111, 112]. For example, Elastic Weight Consolidation (EWC) [98] slows down learning on weights that are important to previous tasks. These methods usually do not need to save any previous data and only train on each task once. But their abilities to overcome catastrophic forgetting are limited.

**Lifelong Learning with Dynamic Model Architecture** There is another related direction on dynamically changing the model structure (i.e., adding new modules) in order to learn the new task without interfering learned knowledge for previous tasks, such as [113, 114, 115]. These approaches could successfully prevent forgetting. However, they do not suit many lifelong settings in NLP. First, it cannot benefit from the positive transfer between tasks. Second, the size of the model grows dramatically with the number of observed tasks, which makes it infeasible for real-world problems where there are a lot of

tasks.

**Remark** It is worth noting that the term lifelong learning is also widely used in [116, 117, 118, 119], which mainly focus on how to represent, reserve and extract knowledge of previous tasks. These works belong to a research direction different from lifelong learning without catastrophic forgetting.

## 5.8 Conclusion

In this chapter, we introduce lifelong learning into relation detection, and find that two state-of-the-art lifelong learning algorithms, GEM and EWC, are outperformed by a simple memory replay method EMR on many benchmarks. Based on EMR, we further propose to use embedding alignment to alleviate the problem of embedding space distortion, which we think is one reason that causes catastrophic forgetting. Also, we propose to choose diverse samples to store in the memory by conducting K-Means in the model embedding space. Experiments verify that our proposed methods significantly outperform other baselines.

# Chapter 6

# Maximizing Resource Allocation for Real User Services

## 6.1 Introduction

Recently, the development of Large Language Models (LLMs) such as GPT-3 [3], PaLM [120], and ChatGPT [4] has brought significant advances in natural language processing, enabling superior performance in downstream tasks such as language understanding [121, 120], question answering [122, 123], and dialogue systems [124, 125]. These models have shown remarkable abilities in understanding and generating human-like language, providing benefits across various domains, including healthcare, education, finance, and entertainment. With their ability to learn from massive amounts of data, these models are constantly improving and becoming more versatile, paving the way for new and exciting applications.

However, with the proliferation of these models, concerns have emerged regarding their potential misuse for malicious purposes. One of the most significant threats is the use of large language models to impersonate human users and engage in nefarious

activities, such as fraud, spamming, or denial-of-service attacks. For instance, ChatGPT agents could be used by hackers to occupy all customer service channels of various corporations, such as e-commerce, airlines, and banks. Moreover, with the help of text-to-speech (TTS) techniques, machine-generated voices could even occupy public service lines like 911, leading to severe public crises [126]. These attacks could cause significant harm to online service providers and their users, eroding the trust and integrity of online interactions.

Differentiating between human users and malicious bots is a long-standing topic of interest for both industry and academia. Conventional techniques, such as the use of CAPTCHAs [127], have been developed to determine whether a user is a human or a bot in order to prevent bot spamming and raiding. A commonly used CAPTCHA method involves asking users to recognize distorted letters and digits. However, these approaches face significant challenges when it comes to detecting chatbots involving text and voice only. This is where the emergence of large language models such as GPT-3 and ChatGPT has further complicated the problem of chatbot detection, as they are capable of generating high-quality human-like text and mimicking human behavior to a considerable extent. Although recent studies such as DetectGPT have proposed methods to classify if text is generated by ChatGPT or not, they focus on the offline setting. A recent study [128] shows that these detectors are not reliable under paraphrasing attacks, where a light paraphraser is applied on top of the generative text model. This limitation highlights the need for more robust and accurate methods to differentiate large language models from human users and detect their presence in online chat interactions.

In this chapter, we propose a novel framework named **FLAIR**, Finding LLM Authenticity with a Single Inquiry and Response, to take full advantage of the strength and weakness of LLMs for LLM-based conversational bot detection. Specifically, we introduce a set of carefully designed questions that induce distinct responses between bots

| | Humans good at | Humans not good at |
|---|---|---|
| Bots good at | × | √ memorization<br>computation |
| Bots not good at | √ symbolic manipulation<br>noise filtering<br>randomness<br>graphical understanding | × |

Table 6.1: Leveraging tasks that Bots and Humans are (not) good at

and humans. These questions are tailored to exploit the differences in the way that bots and humans process and generate language. As shown in Table 6.1, certain questions in the fields of symbolic manipulation, noise filtering, and graphical understanding are difficult for bots but relatively easy for humans. Examples include the counting, substitution, positioning, noise injection, and ASCII art. On the other hand, memorization and computation was relatively easy for bots but difficult for humans. Specifically, counting requires users to count the number of times a particular character appears in a string. Substitution involves substituting the characters of a random string according to a given rule. In positioning, users are asked to identify the $k$-th character after the $j$-th appearance of a given character $c$ in a random string. Random editing involves performing random operations, such as dropping, inserting, swapping, and substituting characters, on a randomly generated string and producing three distinct outputs. Noise injection presents common-sense questions that have been modified with additional characters. ASCII art require users to identify what is being depicted by ASCII characters. Memorization asks questions that demand the enumeration of items within a category or domain-specific knowledge that are challenging for humans to recall. Computation requires users to provide answers to complex math questions, such as calculating the product of two randomly sampled four-digit numbers. Our experimental results demonstrate that FLAIR is effective in practice and offers a viable alternative to traditional

CAPTCHAs. The proposed approach shows promise in developing more robust and accurate methods to detect bots quickly and protect online interactions.

The rest of the chapter is organized as follows. Section 6.2 shows some related work in this field. Sections 6.3 and 6.4 present our proposed approaches to detect LLMs by leveraging their weaknesses and strengths respectively. Section 6.5 shows the experimental results. Finally, Section 6.6 concludes the chapter.

## 6.2 Related Works

### 6.2.1 CAPTCHAs

CAPTCHA [127] is a common technique used to block malicious applications like dictionary attacks, E-mail spamming, web crawlers, phishing attacks, etc. There are different types of CAPTCHAs. Text-Based CAPTCHAs require the users to recognize letters and digits in distortion form [129, 130, 131], while Image-Based CAPTCHAs [132] require users to choose images that have similar properties such as traffic lights. Video-Based CAPTCHAs [133] require the user to choose three words that describe a video, and Audio-Based CAPTCHAs [134] ask the user to listen to an audio and submit the mentioned word [135]. Puzzle CAPTCHAs [136] require the user to combine segments to form a complete picture. These techniques are used to differentiate between human users and bots, preventing malicious activities.

### 6.2.2 Large Language Models

The emergence of Large Language Models significantly advances the state-of-the-art (SOTA) for NLP tasks and benchmarks. Enabled by the paradigm of large-scale self-supervised pre-training and task-specific tuning, the early LLMs [1, 137, 138, 139, 2, 77]

demonstrate incredible capability in resolving multiple downstream tasks like text classification, natural language inference, machine translation, reading comprehension, etc. Additionally, via fusing multimodal knowledge like tables, images, and graphs into language modeling process, LLMs have been endowed with the ability to handle multimodal tasks beyond text, including entity typing [140], node classification [141], visual question-answering [142], multimodal commonsense reasoning [143], etc. Furthermore, through the scaling up on both model parameters and context length, LLMs like GPT-3 [3] can even work as few-shot learners and resolve unseen tasks via in-context learning with several demonstrations. Then, to further facilitate LLMs to follow a user's intent, InstructGPT [144] is proposed to aligning LLMs with human intent instructions via deep reinforcement learning on human feedback. Later, ChatGPT [4] and GPT-4 [5], as latest variants of InstructGPT, ignites the enthusiasm of AI-Generated-Content (AIGC). Through collaboration with vector databases and plugin tools [6], GPT-4 currently supports thousands of downstream applications.

### 6.2.3 ChatGPT offline detection

Since its introduction, ChatGPT has become widely used and raised public concerns about potential misuse. For instance, students may use ChatGPT to complete written assignments, making it difficult for instructors to accurately assess student learning. As a result, there is a growing need to detect whether a piece of text was written by ChatGPT. To tackle this problem, DetectGPT [145] proposes a solution by comparing the log probabilities of the original passage with that of the perturbations of the same passage. The hypothesis behind this method is that minor rewrites of text generated by the model would likely result in lower log probabilities compared to the original sample, while minor rewrites of text written by humans may result in either higher or lower log

probabilities. Another line of study model this problem as binary classification problem and fine-tune another model using supervised data [146]. Most recently, [147] fine-tunes a Transformer-based model and uses it to make predictions, which are then explained using SHAP [148]. Another area of research focuses on adding watermarks to AI-generated text in order to facilitate their identification, which involves imprinting specific patterns on the text to make it easier to detect [149]. Soft watermarking, as proposed by [150], involves dividing tokens into green and red lists in order to create these patterns. When generating text, a watermarked LLM is more likely to select a token from the green list, which is determined by the prefix token. These watermarks are often subtle and difficult for humans to notice.

However, as demonstrated in [151], a range of detection methods, including watermarking schemes, neural network-based detectors, and zero-shot classifiers, can be easily defeated by paraphrasing attacks. These attacks involve applying a light paraphraser to text generated by a language model. Furthermore, a theoretical analysis suggests that even the best possible detector can only perform marginally better than a random classifier when dealing with a sufficiently good language model. This highlights the fundamental challenge in offline detection of text generated by advanced language models, which can produce writing that is virtually indistinguishable from human-written text. Thus, it is more meaningful and crucial to shift the focus to online detection settings where users engage in live chat interactions with the system.

In the following discussion, we will present our FLAIR framework for detecting Large Language Models, such as ChatGPT, in an online setting where users engage in real-time conversations with the system. Our method involves asking users questions and distinguish bots from humans based on the correctness of their answers. We have categorized the FLAIRs into two types. The first type includes questions that are difficult for LLMs but easy for humans. These types of questions typically require skills

such as symbolic manipulation, noise filtering, and graphical understanding, which are not strong points of LLMs [152]. It is possible to bypass these tests by fine-tuning LLMs with failed testing cases or using plug-ins of specific routines. We argue that these tests show the fundamental weaknesses inside LLMs. The second type includes questions that are easy for LLMs but difficult for humans. These questions typically require memorization and computation, which is not a strong suit of humans. In the following sections, we will discuss each of them in detail.

## 6.3 Leveraging the Weakness of LLM

### 6.3.1 Counting

State-of-the-art LLMs cannot accurately count characters in a string, while humans can do so with ease. This limitation of LLMs has inspired the design of a counting FLAIR to differentiate humans and LLMs. Participants are asked to count the number of times a specific character appears in a given string. To increase the difficulty of the problem, the appearance of the character is explicitly increased in the string. The generated questions are used as prompts for LLMs [3, 144]. Here's an example question-answering prompt and the generated results from GPT-3, ChatGPT, and a human:

> Q: Please count the number of t in eeooeotetto
>
> GPT-3: There are 5 t's in eeooeotetto.
>
> ChatGPT: There are a total of 5 t's in the string "eeooeotetto".
>
> Human: 3

As we can see from the example, both GPT-3 and ChatGPT are unable to accurately

count the number of times the given character appears in the string, whereas humans can easily give the correct answer.

### 6.3.2 Substitution

It is known that LLMs often output contents that are inconsistent with context [153, 154]. It is a shared weakness of current LLMs. We ask LLMs to spell a random word under a given substitution rule, testing if they can follow the rule consistently. The random word has a length between five to ten characters, which is randomly sampled from a dictionary. Here is an example used to test both LLMs and humans:

> Q: Use m to substitute p, a to substitute e, n to substitute a, g to substitute c, o to substitute h, how to spell peach under this rule?
>
> GPT-3: mogno
>
> ChatGPT: Using the given rule to substitute letters, "peach" would be spelled as "enmog".
>
> Human: mango

Despite the fact that LLMs have been trained to learn the rule of "substitution" during pretraining, they still struggle with applying this rule consistently and repeatedly as in the above substitution task. This testing concept could be further generalized to encryption scheme where a string is transformed based rules.

### 6.3.3 Positioning

Following the speculation about why LLMs cannot count, we further explored whether LLMs have the ability to position. For each question, a random string is generated under

the same rule described in subsection 2.1. LLMs should output the $k$-th character in the string after the $j$-th appearance of a given character $c$, where $k$ and $j$ are random integers and $c$ is a random character chosen form the set of characters in the string. The question below is an example:

Q: Please output the 4th character after the second s in the string rjsjuubrjjsjsucuj

GPT-3: u

ChatGPT: The second "s" in the string "rjsjuubrjjsjsucuj" is at index 6 (counting from 0), so the 4th character after it would be at index 10. Therefore, the output is the letter "j".

Humans: c

In this example, both ChatGPT and GPT3 are unable to correctly locate the character. ChatGPT failed on finding the position of the second s.

### 6.3.4 Random Editing

Random editing is a technique used to evaluate the robustness of natural language processing models against noisy inputs. The technique involves presenting the model with a string of characters (sampled from the set of $[0, 1]$) and randomly dropping, inserting, swapping, or substituting some characters. To verify that the model is capable of performing these random operations, it is required to generate three different outputs under the same conditions. By ensuring that the three outputs are different, we can confirm that the model is applying the random operation.

For random drop, the model is asked to randomly remove $k$ occurrences of a given character $c$ from the string, where $k$ is a random number and $c$ is a random character

that appears in the string. For random insert, the model is asked to randomly insert $k$ occurrences of a given character $c$ into the string at random positions. For random swap, the model is asked to randomly swap $k$ pairs of characters $c$ and $d$ in the string, where $c$ and $d$ are different characters that appear in the string. Finally, for random substitution, the model is asked to randomly replace $k$ occurrences of character $c$ with another character $d$ in the string. Below we show an example of LLMs and humans output on the random drop task:

> Q: randomly drop two 1 from the string: 0110010011. Give me three different outputs.
>
> GPT-3: 0111010011, 0110001011, 0110011011
>
> ChatGPT: 010010011, 011000011, 0010010011
>
> Human: 00010011, 01000011, 01100001

As we can seen from the above example, both GPT-3 and ChatGPT failed on this task, while humans can easily solve this problem.

### 6.3.5 Noise Injection

We first generate a set of common sense questions using the ChatGPT model with the following prompt: Ask 100 common sense questions and answer them, the answers should be a single word. Below is an example question and answer generated by ChatGPT:

> Q: Is water wet or dry?
>
> A: wet

To introduce noise into the questions, we appended a random uppercase word to

each word. Although these uppercase letters may be easily ignored by humans, they can confuse large language models (LLMs) that rely on subword tokens. For instance, we transformed the original question "is water wet or dry? into "isCURIOSITY waterARCANE wetTURBULENT orILLUSION drySAUNA?", and below are the prompt and outputs from GPT-3 and ChatGPT on this task:

> Q: isCURIOSITY waterARCANE wetTURBULENT orILLUSION drySAUNA?
>
> GPT-3: Sauna is dry.
>
> ChatGPT: I'm sorry, but I'm not sure what you're asking. The terms "CURIOSITY," "waterARCANE," "wetTURBULENT," "orILLUSION," and "drySAUNA" don't seem to form a coherent question or statement. Can you please provide more context or clarify your question?
>
> Human: wet.

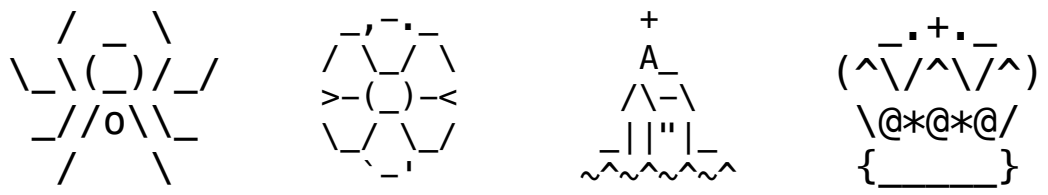
The example question we generated demonstrates the confusion caused by the added noise in our approach. This noise, which involves appending uppercase letters to each word, can significantly impact the judgement of large language models (LLMs) and thus, could be used to construct FLAIRs against current LLMs.

### 6.3.6 ASCII Art

Understanding ASCII arts requires a visual abstraction capability, which is lacking in language models. We collected ASCII arts from ASCII Art Archive[1]. Here are a few

[1]ASCII Art Archive: `www.asciiart.eu`

examples of ASCII art:



(a) Spider (b) Flower (c) House (d) Crown

Figure 6.1: Examples of ASCII arts.

We fed this ASCII art with the same format shown in Figure 6.3.6 to GPT-3, ChatGPT and human, here are the answers:

GPT-3: Unknown

ChatGPT: The ASCII art depicts a person sitting cross-legged with their arms raised above their head in a meditative or yoga position. The forward slashes represent the person's raised legs, the underscores represent the person's torso, and the backslashes represent the person's arms. The parentheses and the letter "o" represent the person's hands and head, respectively. Overall, the ASCII art conveys a sense of peace and serenity.

Human: spider

Graphical understanding is still a challenge for LLMs. Although ChatGPT provided lots of analysis to try to understand ASCII arts, it cannot globally process the characters to give the correct answer. All of the analysis provided by ChatGPT is based on locating character groups.

## 6.4 Leveraging the Strength of LLM

In this section, we will discuss the methods that capitalize on the strengths of LLMs. These questions are typically challenging for humans, but are relatively easy for LLMs due to their ability to memorize vast amounts of information or perform complex computation, thanks to their large number of parameters and pre-training on massive corpora. On the other hand, memorization and complex computation are not typically a strong suit for humans.

### 6.4.1 Memorization

The primary idea behind this type of questions is to ask users to enumerate items within a given category. Below is and example question and answers from GPT-3 (we only list a few items from the output):

> Q: List the capitals of all the states in US:
>
> GPT-3: 1. Montgomery, Alabama 2. Juneau, Alaska 3. Phoenix, Arizona ... 48. Charleston, West Virginia 49. Madison, Wisconsin 50. Cheyenne, Wyoming

For humans, this type of question is not easy as it require a good memorization. There are several choices when designing enumerating questions. The first option is to contain many items to make it more challenging for humans to remember, such as all the countries in the world. The second option is to include relatively old information that people may not have encountered, such as all the movies in 1970s. The third option is domain-specific information that people are unlikely to know, such as the names of all Intel CPU series. We can determine the likelihood of the answer coming from a bot by

verifying the overlap between the given answer and the ground truth. If the overlap is greater than a threshold, then it is more likely that the answer comes from a bot.

In addition to the questions of enumerating all the items in a given category, we also propose domain-specific questions that may not be encountered in daily life. Humans may not have a strong motivation to remember such information, whereas LLMs have been pre-trained on large corpora that are likely to contain such long-tailed and infrequently used information. Below are two example questions and the answer generated by GPT-3:

> What is the fist 50 digits of $\pi$?
>
> GPT-3: The first 50 digits of $\pi$ are
>
> 3.14159265358979323846264338327950288419716939937 51.
>
> What is the cabin volume of a typical Boeing 737?
>
> GPT-3: The cabin volume of a typical Boeing 737 is 1,858 cubic feet.

The answers generated by GPT3 for the provided questions were quite accurate and precise, indicating that LLMs are well-equipped to answer long-tail questions that are not commonly known or encountered in daily life. On the other hand, humans may not have the required knowledge or motivation to remember such specific information, which may make it challenging for them to answer such questions accurately. Therefore, incorporating questions that require specialized knowledge not commonly held by humans can be an effective strategy for distinguishing between LLMs and humans.

### 6.4.2 Computation

Performing complex calculations, such as multiplication, without the aid of a computer or external notes is difficult for humans due to the challenges of recalling interme-

diate steps. In contrast, LLMs excel in remembering the results of common equations, such as the square of $\pi$. For instance, below is an example question and the answer generated by GPT-3:

> What is the square of $\pi$?
>
> GPT-3: The square of $\pi$ is $\pi^2$, which is equal to 9.8696.

Moreover, by utilizing external tools, such as Wolfram, ChatGPT Plugins can even solve more intricate problems, including Algebra. However, for equations that are uncommon, GPT-3 may hallucinate a false answer, as demonstrated by its response to the question

> What is the result of 3256*354?
>
> GPT-3: 1153664

where it provided 1153664 instead of the true answer 1152624. This behavior can be leveraged to distinguish LLMs from humans, as humans are unlikely to fabricate answers and will typically respond by admitting they do not know the solution.

## 6.5 Experiments

In this section, we present the experimental results of our proposed single questions for distinguishing between humans and LLMs. We curate a dataset for each category of the proposed questions, which is used to evaluate the performance of both humans and LLMs. By contrasting the accuracy of responses, we aim to differentiate between the two. In the subsequent subsections, we describe the process of constructing the datasets and report our findings.

### 6.5.1 Datasets

To evaluate the performance of both LLMs and humans, we constructed a dataset for each category of questions and open-sourced it on `https://github.com/hongwang600/FLAIR`.

**Counting** We used the entire alphabet set as the candidate character set. First, we randomly selected a character as the target to count. Then we randomly sampled the target count $k$ from the interval $[10, 20]$. We set the total length of the string to be 30, consisting of $k$ target characters and $30 - k$ characters sampled from the rest. For the generated string, the ground truth answer would be $k$.

**Substitution** To create our dataset, we began by collecting the top 1500 nouns from the Talk English web site[2]. We then filtered the words to include only those with a length between 5 and 10 characters. Next, we randomly generated 100 pairs of words, each with a corresponding substitution map that could transform one word into the other. To ensure the validity of our pairs, we excluded any that would require one character to be mapped to more than one character, which would result in a conflict. The resulting questions presented to participants included the substitution rule and the original word, with the answer requiring another word produced through the substitution.

**Positioning** We use the full alphabet set as candidate character set. We randomly generate a string of length 30 from this set and two random position index $j$ and $k$. Also, we randomly sample another character $c$. The question presented to the participant is "what is the $k$-th character after the $j$-th appearance of the character $c$ in the string". The ground truth of this question can be calculated through a program and we ensure $j$

[2]website URL: `https://www.talkenglish.com/vocabulary/top-1500-nouns.aspx`

and $k$ are random integer within feasible interval, e.g., character $c$ has appeared at least $j$ times.

**Random Editing** For random editing, we support four different operations including drop, insert, swap and substitute as described in Section 6.3.4. For each operation, we generate a random string of length 20 from the $[0, 1]$ set to make it easier to read. We randomly sample the parameters such as interested character and operation count. Then the participant is asked to give three different outputs after the random operation. To check the correctness of the output, we will first check the correctness of each output by comparing the output with the original one from left to right. Then we will check if the three output differs from each other. Only when each individual output is correct and three output are different will the answer be counted as correct.

**Noise Injection** We constructed a dataset of 80 common sense questions using ChatGPT with the following prompt: Ask 80 common sense questions and answer them, the answers should be a single word. To increase the difficulty level and add noise to the questions, we append a random uppercase word to each word. In order to do so, we randomly created a set of 400 random words. For each word in the question, we will randomly select one word from the list and append it to the particular word within the sentence. The ground truth answer to each noisy question is the same as the answer to the original question.

**ASCII Art** We curated a dataset of 50 ASCII arts from the website ASCII Art Archive. Each ASCII art was assigned a ground truth label. Participants were presented with an ASCII art and asked to provide the corresponding label as the answer.

**Memorization** We used a set of questions that required the user has a good memorization. There are two types of question under this category including enumerating and domain-specific questions. For enumerating, the user is asked list items within a given category. We manually collected 100 categories containing more than 50 items or those that were difficult for humans to know with the help of ChatGPT. The question asked users to list the items within the given category, and we calculated the coverage of the response against the ground truth. If the coverage exceeded the threshold of 95%, we considered the answer to have been generated by an LLM. For domain specific questions, we manually collected a set of 100 questions whose answers are difficult for people to recall or access, such as "What is the weight of the world's heaviest lemon in grams?". Although these questions may be challenging for humans to answer, they are relatively easy for large language models (LLMs) due to their pre-training on large corpora that includes these questions.

**Computation** To create the computation dataset, we selected the problem of four-digit multiplication. Specifically, we randomly sampled 100 pairs of four-digit numbers and calculated their product as the ground truth. Participants were asked to solve these multiplication problems and were considered correct if absolute difference between their answer and the ground truth was within 10%. For humans, it can be difficult to accurately calculate these multiplications without the aid of notes or a calculator, leading them to often respond with "I don't know". In contrast, large language models (LLMs) have seen many similar equations during pre-training and tend to provide a guess that is often close to, or the same as, the ground truth. This testing can be further extended to any complicated computation like division, exponents, etc.

### 6.5.2 Main Results

We conducted experiments using various OpenAI models, including GPT-3 [3], and ChatGPT [4]. Additionally, we evaluated recent open-sourced models, such as LLaMA [155], Alpaca [156], and Vicuna [157]. For each model, we evaluated its performance on each proposed category of questions and recorded its accuracy. Furthermore, we asked a group of human-workers to answer the questions and reported their accuracy as well (they are required to answer each question in 10 seconds).

| | Count | Substitution | Positioning | Random Edit | Noise Injection | ASCII Art | Memorization | Computation |
|---|---|---|---|---|---|---|---|---|
| Humans | 100% | 100% | 100% | 100% | 100% | 94% | 6% | 2% |
| GPT3 | 13% | 2% | 15% | 0% | 0% | 0% | 94% | 95% |
| ChatGPT | 17% | 3% | 23% | 2% | 0% | 8% | 99% | 98% |
| LLaMA | 4% | 0% | 16% | 0% | 0% | 0% | 91% | 91% |
| Alpaca | 8% | 0% | 12% | 1% | 0% | 0% | 85% | 99% |
| Vicuna | 15% | 1% | 15% | 0% | 0% | 0% | 93% | 100% |

Table 6.2: The comparison between LLMs and Human on different FLAIRs. The left part are the questions that are easy for humans while hard for LLMs. The right part are the questions that are hard for humans while easy for LLMs.

The results are presented in Table 6.2. In the left section, we compare the performance of humans and LLMs on questions that are considered easy for humans but difficult for LLMs. The results show that humans achieved perfect scores (100%) on all tasks except for ASCII Art, where their accuracy was 94%. On the other hand, most LLMs had difficulty with tasks such as substitution, random edit, noise injection, and ASCII Art, with their accuracy being nearly 0%. However, the LLMs performed better on the count and positioning tasks, as the solution space for these problems is much smaller compared to other tasks, making it easier for the models to guess the correct answer.

In the right section, we compared human performance with LLMs on questions that are difficult for humans but easy for LLMs. The results indicate that humans performed poorly on these questions, as they require good memorization or computation abilities. In contrast, the performance of LLMs was excellent, with some models achieving almost

100% accuracy.

## 6.6 Conclusion

In conclusion, this chapter proposes a new framework called FLAIR for detecting conversational bots in an online environment. The proposed approach targets a single question scenario that can effectively differentiate human users from bots by using questions that are easy for humans but difficult for bots, and vice versa. Our experiments demonstrate the effectiveness of this approach and show the strengths of different types of questions. This framework provides online service providers with a new way to protect themselves against fraudulent activities and ensure that they are serving real users. Furthermore, we have open-sourced our dataset on GitHub and welcome further contributions to enlarge the detection datasets.

# Chapter 7

# Conclusion

In conclusion, this dissertation has delved into critical challenges and presented innovative solutions in the field of Natural Language Processing (NLP). Our work has focused on two fundamental areas that are vital for the advancement of NLP: learning with limited data and optimizing resource utilization. By addressing these challenges, we have made significant contributions to the development of robust and efficient NLP models.

In the first part, we tackled the essential concern of learning with limited data, where the scarcity of annotated examples demands effective generalization of models. Our approach to few-shot learning for multi-hop reasoning over knowledge graphs introduced meticulously constructed datasets, and the meta-encoder method proved its efficacy in generating task-dependent models for improved multi-hop reasoning performance. Moreover, our exploration of self-supervised learning for summarization showcased promising results in learning contextualized sentence representations, offering enhanced sample efficiency and faster convergence. Additionally, our investigation into the limitations of large language models when encountering out-of-distribution data shifts has laid the groundwork for potential mitigating techniques. The use of LMs with tutor-based training exhibited impressive results, inspiring new avenues of research in symbolic manipulation

tasks.

In the second part of the dissertation, we focused on optimizing resource utilization in NLP. Recognizing the rapidly changing nature of the world, we presented innovative approaches for updating models in lifelong learning scenarios, ensuring continuous adaptation to new data. Our proposed memory replay approach and alignment model effectively addressed the catastrophic forgetting problem, improving model performance in lifelong learning. Considering the resource constraints and the growing adoption of large language models in dialogue systems, we addressed the challenge of safeguarding against malicious Distributed Denial of Service (DDoS) attacks. The FLAIR framework provided a powerful tool to detect conversational bots, enabling optimal resource allocation for real users and protecting online service providers against fraudulent activities.

Throughout this dissertation, we demonstrated the value of innovative approaches and extensive experimentation in overcoming NLP challenges. Our contributions not only advance the field of NLP but also hold great potential in real-world applications, impacting dialogue systems, summarization, and security measures against malicious bots. As we conclude this work, we acknowledge that the journey of NLP research is ongoing and ever-evolving. The challenges and opportunities that lie ahead are vast, from handling limited data and resource constraints to exploring novel techniques for better language understanding and generation. We hope that our contributions inspire further exploration, collaboration, and progress in the field of NLP, ultimately leading to more intelligent, versatile, and efficient language models that positively impact various domains of human life.

# Bibliography


[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, *arXiv preprint arXiv:1810.04805* (2018).

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et. al.*, *Language models are unsupervised multitask learners*, *OpenAI blog* **1** (2019), no. 8 9.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language models are few-shot learners*, in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.

[4] OpenAI, "Introducing chatgpt." `https://openai.com/blog/chatgpt`, 2022. Accessed on November 30, 2022.

[5] OpenAI, "Gpt-4." `https://openai.com/research/gpt-4`, 2023. Accessed on March 14, 2023.

[6] OpenAI, "Chatgpt-plugins." `https://openai.com/blog/chatgpt-plugins`, 2023. Accessed on March 23, 2023.

[7] A. Bordes, N. Usunier, S. Chopra, and J. Weston, *Large-scale simple question answering with memory networks*, *CoRR* **abs/1506.02075** (2015) [arXiv:1506.0207].

[8] X. Han, H. Zhu, P. Yu, Z. Wang, Y. Yao, Z. Liu, and M. Sun, *Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 4803–4809, 2018.

[9] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, *Dbpedia: A nucleus for a web of open data*, in *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.* (K. Aberer, K. Choi, N. F. Noy, D. Allemang, K. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, eds.), vol. 4825 of *Lecture Notes in Computer Science*, pp. 722–735, Springer, 2007.

[10] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, *Freebase: a collaboratively created graph database for structuring human knowledge*, in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008* (J. T. Wang, ed.), pp. 1247–1250, ACM, 2008.

[11] D. Vrandecic and M. Krötzsch, *Wikidata: a free collaborative knowledgebase*, *Commun. ACM* **57** (2014), no. 10 78–85.

[12] X. Yao and B. V. Durme, *Information extraction over structured data: Question answering with freebase*, in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pp. 956–966, The Association for Computer Linguistics, 2014.

[13] A. Bordes, N. Usunier, S. Chopra, and J. Weston, *Large-scale simple question answering with memory networks*, *CoRR* **abs/1506.02075** (2015) [arXiv:1506.0207].

[14] W. Yih, M. Chang, X. He, and J. Gao, *Semantic parsing via staged query graph generation: Question answering with knowledge base*, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 1321–1331, The Association for Computer Linguistics, 2015.

[15] M. Yu, W. Yin, K. S. Hasan, C. N. dos Santos, B. Xiang, and B. Zhou, *Improved neural relation detection for knowledge base question answering*, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers* (R. Barzilay and M. Kan, eds.), pp. 571–581, Association for Computational Linguistics, 2017.

[16] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, *Translating embeddings for modeling multi-relational data*, in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural*

*Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.* (C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2787–2795, 2013.

[17] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, *Complex embeddings for simple link prediction*, in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016* (M. Balcan and K. Q. Weinberger, eds.), vol. 48 of *JMLR Workshop and Conference Proceedings*, pp. 2071–2080, JMLR.org, 2016.

[18] N. Lao and W. W. Cohen, *Relational retrieval using a combination of path-constrained random walks*, *Machine Learning* **81** (2010), no. 1 53–67.

[19] W. Xiong, M. Yu, S. Chang, X. Guo, and W. Y. Wang, *One-shot relational learning for knowledge graphs*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018* (E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, eds.), pp. 1980–1990, Association for Computational Linguistics, 2018.

[20] S. Ravi and H. Larochelle, *Optimization as a model for few-shot learning*, in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[21] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, PMLR, 2017.

[22] J. Gu, Y. Wang, Y. Chen, V. O. K. Li, and K. Cho, *Meta-learning for low-resource neural machine translation*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018* (E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, eds.), pp. 3622–3631, Association for Computational Linguistics, 2018.

[23] P. Huang, C. Wang, R. Singh, W. Yih, and X. He, *Natural language to structured query generation via meta-learning*, in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)* (M. A. Walker, H. Ji, and A. Stent, eds.), pp. 732–738, Association for Computational Linguistics, 2018.

[24] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, *Learning to compare: Relation network for few-shot learning*, in *2018 IEEE*

*Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 1199–1208, IEEE Computer Society, 2018.

[25] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, *A simple neural attentive meta-learner*, in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[26] W. Chen, Y. Liu, Z. Kira, Y. F. Wang, and J. Huang, *A closer look at few-shot classification*, *CoRR* **abs/1904.04232** (2019) [arXiv:1904.0423].

[27] M. Nickel, V. Tresp, and H. Kriegel, *A three-way model for collective learning on multi-relational data*, in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011* (L. Getoor and T. Scheffer, eds.), pp. 809–816, Omnipress, 2011.

[28] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, *Embedding entities and relations for learning and inference in knowledge bases*, in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[29] J. Wu, R. Xie, Z. Liu, and M. Sun, *Knowledge representation via joint learning of sequential text and knowledge graphs*, 2016.

[30] A. Neelakantan, B. Roth, and A. McCallum, *Compositional vector space models for knowledge base inference*, in *2015 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 22-25, 2015*, AAAI Press, 2015.

[31] W. Xiong, T. Hoang, and W. Y. Wang, *Deeppath: A reinforcement learning method for knowledge graph reasoning*, in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017* (M. Palmer, R. Hwa, and S. Riedel, eds.), pp. 564–573, Association for Computational Linguistics, 2017.

[32] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, *Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning*, in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[33] W. Chen, W. Xiong, X. Yan, and W. Y. Wang, *Variational knowledge graph reasoning*, in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)* (M. A. Walker, H. Ji, and A. Stent, eds.), pp. 1823–1832, Association for Computational Linguistics, 2018.

[34] X. V. Lin, R. Socher, and C. Xiong, *Multi-hop knowledge graph reasoning with reward shaping*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018* (E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, eds.), pp. 3243–3253, Association for Computational Linguistics, 2018.

[35] Y. Shen, J. Chen, P. Huang, Y. Guo, and J. Gao, *M-walk: Learning to walk over graphs using monte carlo tree search*, in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 6787–6798, 2018.

[36] M. Gardner, P. P. Talukdar, B. Kisiel, and T. M. Mitchell, *Improving learning and inference in a large knowledge-base using latent syntactic cues*, in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 833–838, ACL, 2013.

[37] M. Gardner, P. P. Talukdar, J. Krishnamurthy, and T. M. Mitchell, *Incorporating vector space similarity in random walk inference over knowledge bases*, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL* (A. Moschitti, B. Pang, and W. Daelemans, eds.), pp. 397–406, ACL, 2014.

[38] W. Y. Wang and W. W. Cohen, *Joint information extraction and reasoning: A scalable statistical relational learning approach*, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 355–364, The Association for Computer Linguistics, 2015.

[39] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon, *Representing text for joint embedding of text and knowledge bases*, in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015* (L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, eds.), pp. 1499–1509, The Association for Computational Linguistics, 2015.

[40] A. Nichol, J. Achiam, and J. Schulman, *On first-order meta-learning algorithms*, *CoRR* **abs/1803.02999** (2018) [arXiv:1803.0299].

[41] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, *Meta-learning with latent embedding optimization*, *CoRR* **abs/1807.05960** (2018) [arXiv:1807.0596].

[42] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural Computation* **9** (1997), no. 8 1735–1780.

[43] T. M. Mitchell, W. W. Cohen, E. R. H. Jr., P. P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. D. Mishra, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. A. Platanios, A. Ritter, M. Samadi, B. Settles, R. C. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling, *Never-ending learning*, *Commun. ACM* **61** (2018), no. 5 103–115.

[44] J. G. Carbonell and J. Goldstein, *The use of mmr, diversity-based reranking for reordering documents and producing summaries*, in *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia* (W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, eds.), pp. 335–336, ACM, 1998.

[45] R. Mihalcea and P. Tarau, *Textrank: Bringing order into text*, in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*, pp. 404–411, ACL, 2004.

[46] R. T. McDonald, *A study of global inference algorithms in multi-document summarization*, in *Advances in Information Retrieval, 29th European Conference on IR Research, ECIR 2007, Rome, Italy, April 2-5, 2007, Proceedings* (G. Amati, C. Carpineto, and G. Romano, eds.), vol. 4425 of *Lecture Notes in Computer Science*, pp. 557–564, Springer, 2007.

[47] Z. Cao, F. Wei, L. Dong, S. Li, and M. Zhou, *Ranking with recursive neural networks and its application to multi-document summarization*, in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.* (B. Bonet and S. Koenig, eds.), pp. 2153–2159, AAAI Press, 2015.

[48] R. Nallapati, F. Zhai, and B. Zhou, *Summarunner: A recurrent neural network based sequence model for extractive summarization of documents*, in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.* (S. P. Singh and S. Markovitch, eds.), pp. 3075–3081, AAAI Press, 2017.

[49] K. Al-Sabahi, Z. Zhang, and M. Nadher, *A hierarchical structured self-attentive model for extractive document summarization (HSSAS)*, *IEEE Access* **6** (2018) 24205–24212.

[50] Q. Zhou, N. Yang, F. Wei, S. Huang, M. Zhou, and T. Zhao, *Neural document summarization by jointly learning to score and select sentences*, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers* (I. Gurevych and Y. Miyao, eds.), pp. 654–663, Association for Computational Linguistics, 2018.

[51] X. Zhang, M. Lapata, F. Wei, and M. Zhou, *Neural latent extractive document summarization*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018* (E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, eds.), pp. 779–784, Association for Computational Linguistics, 2018.

[52] J. Pennington, R. Socher, and C. D. Manning, *Glove: Global vectors for word representation*, in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[53] Y. Nie and M. Bansal, *Shortcut-stacked sentence encoders for multi-domain inference*, in *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP, RepEval@EMNLP 2017, Copenhagen, Denmark, September 8, 2017* (S. R. Bowman, Y. Goldberg, F. Hill, A. Lazaridou, O. Levy, R. Reichart, and A. Søgaard, eds.), pp. 41–45, Association for Computational Linguistics, 2017.

[54] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, *A structured self-attentive sentence embedding*, in *International Conference on Learning Representations*, 2017.

[55] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, *Deep contextualized word representations*, in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)* (M. A. Walker, H. Ji, and A. Stent, eds.), pp. 2227–2237, Association for Computational Linguistics, 2018.

[56] J. Devlin, M. Chang, K. Lee, and K. Toutanova, *BERT: pre-training of deep bidirectional transformers for language understanding*, *CoRR* **abs/1810.04805** (2018) [arXiv:1810.0480].

[57] S. Subramanian, A. Trischler, Y. Bengio, and C. J. Pal, *Learning general purpose distributed sentence representations via large scale multi-task learning*, *CoRR* **abs/1804.00079** (2018) [arXiv:1804.0007].

[58] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil, *Universal sentence encoder for english*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018* (E. Blanco and W. Lu, eds.), pp. 169–174, Association for Computational Linguistics, 2018.

[59] L. Logeswaran and H. Lee, *An efficient framework for learning sentence representations*, in *International Conference on Learning Representations*, 2018.

[60] M. Pagliardini, P. Gupta, and M. Jaggi, *Unsupervised learning of sentence embeddings using compositional n-gram features*, in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)* (M. A. Walker, H. Ji, and A. Stent, eds.), pp. 528–540, Association for Computational Linguistics, 2018.

[61] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, *Self-taught learning: transfer learning from unlabeled data*, in *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007* (Z. Ghahramani, ed.), vol. 227 of *ACM International Conference Proceeding Series*, pp. 759–766, ACM, 2007.

[62] C. Doersch, A. Gupta, and A. A. Efros, *Unsupervised visual representation learning by context prediction*, in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1422–1430, IEEE Computer Society, 2015.

[63] P. Agrawal, J. Carreira, and J. Malik, *Learning to see by moving*, in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 37–45, IEEE Computer Society, 2015.

[64] X. Wang and A. Gupta, *Unsupervised learning of visual representations using videos*, in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 2794–2802, IEEE Computer Society, 2015.

[65] D. Okanohara and J. Tsujii, *A discriminative language model with pseudo-negative samples*, in *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech*

*Republic* (J. A. Carroll, A. van den Bosch, and A. Zaenen, eds.), The Association for Computational Linguistics, 2007.

[66] R. Collobert and J. Weston, *A unified architecture for natural language processing: deep neural networks with multitask learning*, in *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008* (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), vol. 307 of *ACM International Conference Proceeding Series*, pp. 160–167, ACM, 2008.

[67] J. Wu, X. Wang, and W. Y. Wang, *Self-supervised dialogue learning*, in *ACL 2019, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, The Association for Computational Linguistics, 2019.

[68] H. Lee, J. Huang, M. Singh, and M. Yang, *Unsupervised representation learning by sorting sequences*, in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 667–676, IEEE Computer Society, 2017.

[69] K. M. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, *Teaching machines to read and comprehend*, in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1693–1701, 2015.

[70] R. Nallapati, B. Zhou, C. N. dos Santos, Ç. Gülçehre, and B. Xiang, *Abstractive text summarization using sequence-to-sequence rnns and beyond*, in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016* (Y. Goldberg and S. Riezler, eds.), pp. 280–290, ACL, 2016.

[71] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 6000–6010, 2017.

[72] A. See, P. J. Liu, and C. D. Manning, *Get to the point: Summarization with pointer-generator networks*, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers* (R. Barzilay and M. Kan, eds.), pp. 1073–1083, Association for Computational Linguistics, 2017.

[73] C.-Y. Lin, *Rouge: A package for automatic evaluation of summaries*, *Text Summarization Branches Out* (2004).

[74] M. I. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena, *Show your work: Scratchpads for intermediate computation with language models*, *CoRR* **abs/2112.00114** (2021) [arXiv:2112.0011].

[75] G. Recchia, *Teaching autoregressive language models complex tasks by demonstration*, *Computing Research Repository* **abs/2109.02102** (2021). version 3.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 5998–6008, 2017.

[77] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *Exploring the limits of transfer learning with a unified text-to-text transformer*, *J. Mach. Learn. Res.* **21** (2020) 140:1–140:67.

[78] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. Chi, Q. Le, and D. Zhou, *Chain of thought prompting elicits reasoning in large language models*, *CoRR* **abs/2201.11903** (2022) [arXiv:2201.1190].

[79] S. Roy and D. Roth, *Solving general arithmetic word problems*, in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015* (L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, eds.), pp. 1743–1752, The Association for Computational Linguistics, 2015.

[80] K. Cobbe, V. Kosaraju, M. Bavarian, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, *Training verifiers to solve math word problems*, *CoRR* **abs/2110.14168** (2021) [arXiv:2110.1416].

[81] R. Nogueira, Z. Jiang, and J. Lin, *Investigating the limitations of the transformers with simple arithmetic tasks*, *CoRR* **abs/2102.13019** (2021) [arXiv:2102.1301].

[82] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev,

H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, *Palm: Scaling language modeling with pathways*, *CoRR* **abs/2204.02311** (2022) [arXiv:2204.0231].

[83] B. Kim, K. S. Ki, S. Rhim, and G. Gweon, *EPT-X: An expression-pointer transformer model that generates eXplanations for numbers*, in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4442–4458, May, 2022.

[84] A. Graves, G. Wayne, and I. Danihelka, *Neural turing machines*, *CoRR* **abs/1410.5401** (2014) [arXiv:1410.5401].

[85] P. He, X. Liu, J. Gao, and W. Chen, *Deberta: decoding-enhanced bert with disentangled attention*, in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[86] O. Vinyals, M. Fortunato, and N. Jaitly, *Pointer networks*, in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2692–2700, 2015.

[87] A. See, P. J. Liu, and C. D. Manning, *Get to the point: Summarization with pointer-generator networks*, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, July, 2017.

[88] D. Pomerleau, *ALVINN: an autonomous land vehicle in a neural network*, in *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]* (D. S. Touretzky, ed.), pp. 305–313, Morgan Kaufmann, 1988.

[89] S. Ross, G. J. Gordon, and D. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning*, in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011* (G. J. Gordon, D. B. Dunson, and M. Dudík, eds.), vol. 15 of *JMLR Proceedings*, pp. 627–635, JMLR.org, 2011.

[90] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur,

G. Gur-Ari, and V. Misra, *Solving quantitative reasoning problems with language models*, *CoRR* **abs/2206.14858** (2022) [arXiv:2206.1485].

[91] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et. al.*, *Transformers: State-of-the-art natural language processing*, in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

[92] S. Riedel, L. Yao, A. McCallum, and B. M. Marlin, *Relation extraction with matrix factorization and universal schemas*, in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 74–84, 2013.

[93] W.-t. Yih, M.-W. Chang, X. He, and J. Gao, *Semantic parsing via staged query graph generation: Question answering with knowledge base*, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1321–1331, Association for Computational Linguistics, 2015.

[94] M. Yu, W. Yin, K. S. Hasan, C. N. dos Santos, B. Xiang, and B. Zhou, *Improved neural relation detection for knowledge base question answering*, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pp. 571–581, 2017.

[95] M. B. Ring, *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

[96] S. Thrun, *Lifelong Learning Algorithms*, pp. 181–209. Springer US, Boston, MA, 1998.

[97] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[98] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, *Overcoming catastrophic forgetting in neural networks*, *CoRR* **abs/1612.00796** (2016) [arXiv:1612.0079].

[99] D. Lopez-Paz and M. Ranzato, *Gradient episodic memory for continual learning*, in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 6470–6479, 2017.

[100] M. McCloskey and N. J. Cohen, *Catastrophic interference in connectionist networks: The sequential learning problem*, in *Psychology of learning and motivation*, vol. 24, pp. 109–165. Elsevier, 1989.

[101] R. Ratcliff, *Connectionist models of recognition memory: constraints imposed by learning and forgetting functions.*, *Psychological review* **97** (1990), no. 2 285–308.

[102] J. L. McClelland, B. L. McNaughton, and R. C. O'reilly, *Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.*, *Psychological review* **102** (1995), no. 3 419–457.

[103] R. M. French, *Catastrophic forgetting in connectionist networks*, *Trends in cognitive sciences* **3** (1999), no. 4 128–135.

[104] D. Vrandečić and M. Krötzsch, *Wikidata: a free collaborative knowledgebase*, *Communications of the ACM* **57** (2014), no. 10 78–85.

[105] Y. LeCun, *The mnist database of handwritten digits*, *http://yann. lecun. com/exdb/mnist/* (1998).

[106] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, tech. rep., Citeseer, 2009.

[107] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, *icarl: Incremental classifier and representation learning*, in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5533–5542, 2017.

[108] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, *icarl: Incremental classifier and representation learning*, in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 5533–5542, IEEE, 2017.

[109] Anonymous, *Efficient lifelong learning with a-gem*, in *Submitted to International Conference on Learning Representations*, 2019. under review.

[110] X. Liu, M. Masana, L. Herranz, J. van de Weijer, A. M. López, and A. D. Bagdanov, *Rotate your networks: Better weight consolidation and less catastrophic forgetting*, *CoRR* **abs/1802.02950** (2018) [arXiv:1802.0295].

[111] H. Ritter, A. Botev, and D. Barber, *Online structured laplace approximations for overcoming catastrophic forgetting*, *CoRR* **abs/1805.07810** (2018).

[112] F. Zenke, B. Poole, and S. Ganguli, *Continual learning through synaptic intelligence*, in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3987–3995, 2017.

[113] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, *Error-driven incremental learning in deep convolutional neural network for large-scale image classification*, in *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), pp. 177–186, ACM, 2014.

[114] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, *Progressive neural networks*, *CoRR* **abs/1606.04671** (2016) [arXiv:1606.0467].

[115] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, *Pathnet: Evolution channels gradient descent in super neural networks*, *CoRR* **abs/1701.08734** (2017) [arXiv:1701.0873].

[116] Z. Chen, N. Ma, and B. Liu, *Lifelong learning for sentiment classification*, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 750–756, Association for Computational Linguistics, 2015.

[117] Z. Chen, *Lifelong machine learning for topic modeling and beyond*, in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pp. 133–139, Association for Computational Linguistics, 2015.

[118] L. Shu, B. Liu, H. Xu, and A. Kim, *Lifelong-rl: Lifelong relaxation labeling for separating entities and aspects in opinion targets*, in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 225–235, Association for Computational Linguistics, 2016.

[119] L. Shu, H. Xu, and B. Liu, *Lifelong learning crf for supervised aspect extraction*, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 148–154, Association for Computational Linguistics, 2017.

[120] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et. al.*, *Palm: Scaling language modeling with pathways*, *arXiv preprint arXiv:2204.02311* (2022).

[121] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, *et. al.*, *Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model*, *arXiv preprint arXiv:2201.11990* (2022).

[122] D. Su, Y. Xu, G. I. Winata, P. Xu, H. Kim, Z. Liu, and P. Fung, *Generalizing question answering system with pre-trained language model fine-tuning*, in

*Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pp. 203–211, 2019.

[123] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C. D. Manning, C. Ré, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. J. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. S. Kim, N. Guha, N. S. Chatterji, O. Khattab, P. Henderson, Q. Huang, R. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. F. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda, *Holistic evaluation of language models*, *arXiv* (2022).

[124] W. Wang, Z. Zhang, J. Guo, Y. Dai, B. Chen, and W. Luo, *Task-oriented dialogue system as natural language generation*, in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2698–2703, 2022.

[125] J. Qian and X. Yan, *Language model detoxification in dialogue with contextualized stance control*, *arXiv preprint arXiv:2301.10368* (2023).

[126] Z. Wang, M. Yang, C. Jin, J. Liu, Z. Wen, S. Liu, and Z. Zhang, *Ifdds: An anti-fraud outbound robot*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 16117–16119, 2021.

[127] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, *Captcha: Using hard ai problems for security*, in *Eurocrypt*, vol. 2656, pp. 294–311, 2003.

[128] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi, *Can ai-generated text be reliably detected?*, *arXiv preprint arXiv:2303.11156* (2023).

[129] M. Chew and H. S. Baird, *Baffletext: A human interactive proof*, in *Document Recognition and Retrieval X*, vol. 5010, pp. 305–316, SPIE, 2003.

[130] G. Mori and J. Malik, *Recognizing objects in adversarial clutter: Breaking a visual captcha*, in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 1, pp. I–I, IEEE, 2003.

[131] J. Yan and A. S. El Ahmad, *A low-cost attack on a microsoft captcha*, in *Proceedings of the 15th ACM conference on Computer and communications security*, pp. 543–554, 2008.

[132] R. Gossweiler, M. Kamvar, and S. Baluja, *What's up captcha? a captcha based on image orientation*, in *Proceedings of the 18th international conference on World wide web*, pp. 841–850, 2009.

[133] K. A. Kluever, *Evaluating the Usability and Security of a Video CAPTCHA*. Rochester Institute of Technology, 2008.

[134] H. Gao, H. Liu, D. Yao, X. Liu, and U. Aickelin, *An audio captcha to distinguish humans from computers*, in *2010 Third International Symposium on Electronic Commerce and Security*, pp. 265–269, IEEE, 2010.

[135] B. S. Saini and A. Bala, *A review of bot protection using captcha for web security*, *IOSR Journal of Computer Engineering* **8** (2013), no. 6 36–42.

[136] V. P. Singh and P. Pal, *Survey of different types of captcha*, *International Journal of Computer Science and Information Technologies* **5** (2014), no. 2 2242–2245.

[137] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, *arXiv preprint arXiv:1907.11692* (2019).

[138] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*, *arXiv preprint arXiv:1910.13461* (2019).

[139] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et. al.*, *Improving language understanding by generative pre-training*, *OpenAI blog* **1** (2018), no. 8 8.

[140] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, *Ernie: Enhanced language representation with informative entities*, *arXiv preprint arXiv:1905.07129* (2019).

[141] J. Zhang, H. Zhang, C. Xia, and L. Sun, *Graph-bert: Only attention is needed for learning graph representations*, *arXiv preprint arXiv:2001.05140* (2020).

[142] W. Su, X. Zhu, Y. Cao, B. Li, L. Lu, F. Wei, and J. Dai, *Vl-bert: Pre-training of generic visual-linguistic representations*, *arXiv preprint arXiv:1908.08530* (2019).

[143] W. Wang, L. Dong, H. Cheng, H. Song, X. Liu, X. Yan, J. Gao, and F. Wei, *Visually-augmented language modeling*, *arXiv preprint arXiv:2205.10178* (2022).

[144] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et. al.*, *Training language models to follow instructions with human feedback*, *arXiv preprint arXiv:2203.02155* (2022).

[145] E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, *Detectgpt: Zero-shot machine-generated text detection using probability curvature*, *arXiv preprint arXiv:2301.11305* (2023).

[146] A. Bakhtin, S. Gross, M. Ott, Y. Deng, M. Ranzato, and A. Szlam, *Real or fake? learning to discriminate machine from human generated text*, *arXiv preprint arXiv:1906.03351* (2019).

[147] S. Mitrović, D. Andreoletti, and O. Ayoub, *Chatgpt or human? detect and explain. explaining decisions of machine learning model for detecting short chatgpt-generated text*, *arXiv preprint arXiv:2301.13852* (2023).

[148] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions*, *Advances in neural information processing systems* **30** (2017).

[149] X. Zhao, Y.-X. Wang, and L. Li, *Protecting language generation models via invisible watermarking*, *arXiv preprint arXiv:2302.03162* (2023).

[150] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, *A watermark for large language models*, *arXiv preprint arXiv:2301.10226* (2023).

[151] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi, *Can ai-generated text be reliably detected?*, *arXiv preprint arXiv:2303.11156* (2023).

[152] J. Qian, H. Wang, Z. Li, S. Li, and X. Yan, *Limitations of language models in arithmetic and symbolic induction*, *arXiv preprint arXiv:2208.05051* (2022).

[153] Y. Elazar, N. Kassner, S. Ravfogel, A. Ravichander, E. Hovy, H. Schütze, and Y. Goldberg, *Measuring and improving consistency in pretrained language models*, *Transactions of the Association for Computational Linguistics* **9** (2021) 1012–1031.

[154] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou, *Self-consistency improves chain of thought reasoning in language models*, *arXiv preprint arXiv:2203.11171* (2022).

[155] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et. al.*, *Llama: Open and efficient foundation language models*, *arXiv preprint arXiv:2302.13971* (2023).

[156] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model." `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[157] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, *Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality*, March, 2023.